University College London

Computer Science

A thesis presented for the degree of Doctor of Philosophy in Computer Science

App Store Analysis for Software Engineering

January 25, 2017

Author:

William Martin

Supervisors:

Mark Harman

Yue Jia

Federica Sarro

Examiners:

Rachel Harrison

Walid Maalej

I, William Martin confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

App Store Analysis concerns the mining of data from apps, made possible through app stores. This thesis extracts publicly available data from app stores, in order to detect and analyse relationships between technical attributes, such as software features, and non-technical attributes, such as rating and popularity information. The thesis identifies the App Sampling Problem, its effects and a methodology to ameliorate the problem. The App Sampling Problem is a fundamental sampling issue concerned with mining app stores, caused by the rather limited 'most-popular-only' ranked app discovery present in mobile app stores. This thesis provides novel techniques for the analysis of technical and non-technical data from app stores. Topic modelling is used as a feature extraction technique, which is shown to produce the same results as n-gram feature extraction, that also enables linking technical features from app descriptions with those in user reviews. Causal impact analysis is applied to app store performance data, leading to the identification of properties of statistically significant releases, and developer-controlled properties which could increase a release's chance for causal significance. This thesis introduces the Causal Impact Release Analysis tool, CIRA, for performing causal impact analysis on app store data, which makes the aforementioned research possible; combined with the earlier feature extraction technique, this enables the identification of the claimed software features that may have led to significant positive and negative changes after a release.

Impact Statement

The work in this thesis seeks primarily to answer the question of "what makes successful apps successful?" through performing empirical analysis of app store data. The contributions work toward this goal, first establishing a feature extraction technique using topic modelling, and identifying the sampling issues present in mining app store data (publication at MSR 2015). This work looks at time series data, identifying the app releases that had the greatest subsequent effect on app success, and analysing common factors (publications at ICSE 2016 and FSE 2016). One of the outcomes of this work is the tool, CIRA, an implementation of causal impact analysis, a time series analysis technique. The secondary goal of this thesis is to help define the growing field of "app store analysis for software engineering", which it does through a comprehensive review of literature in the field (accepted for publication in the Transactions of Software Engineering journal).

Publications

The following papers were authored as part of this PhD.

First-Author Peer-Reviewed Papers

– W. Martin, M. Harman, Y. Jia, F. Sarro, and Y. Zhang, "The app sampling problem for app store mining," in *Proceedings of the 12th IEEE Working Conference on Mining Software Repositories*, MSR '15, 2015, pp. 123–133 *Citations at the time of writing:* 35

– W. Martin, "Causal impact for app store analysis," in *Companion Proceedings of the 38th International Conference on Software Engineering*, ICSE Companion '16. ACM, 2016 *Citations at the time of writing:* 3

– W. Martin, F. Sarro, and M. Harman, "Causal impact analysis for app releases in Google Play," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, FSE '16, 2016, to appear *Citations at the time of writing:* 5

– W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A survey of app store analysis for software engineering," *IEEE Transactions on Software Engineering*, 2016 *Citations at the time of writing:* 20

Technical Reports

– W. Martin, F. Sarro, and M. Harman, "Causal impact analysis applied to app releases in Google Play and Windows Phone Store," University College London, Tech. Rep., 2015, rN/15/07 *Citations at the time of writing:* 6

– W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A survey of app store analysis for software engineering," University College London, Tech. Rep., 2016, rN/16/02 *Citations at the time of writing:* 20

Co-Authored Peer-Reviewed Papers

In addition to the first-author papers listed above, I have also contributed to the following publications:

– F. Sarro, A. Al-Subaihin, M. Harman, Y. Jia, W. Martin, and Y. Zhang, "Feature lifecycles as they spread, migrate, remain and die in app stores," in *Proceedings of the Requirements Engineering Conference, 23rd IEEE International (RE'15)*. IEEE, 2015 *Citations at the time of writing:* 15

– A. Al-Subaihin, A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang, "App store mining and analysis," in *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*, DeMobile 2015. ACM, 2015, pp. 1–2 *Citations at the time of writing:* 3

– M. Harman, A. Al-Subaihin, Y. Jia, W. Martin, F. Sarro, and Y. Zhang, "Mobile app and app store analysis, testing and optimisation," in *Proceedings of the International Conference on Mobile Software Engineering and Systems*, MOBILESoft '16. ACM, 2016, pp. 243–244 *Citations at the time of writing:* 1

Technical Reports

– A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang, "App store analysis: Mining app stores for relationships between customer, business and technical characteristics," Tech. Rep., 2014, rN/14/10 *Citations at the time of writing:* 11

Acknowledgements

I'd like to send a big thanks to my supervisors Mark, Shin, Yue and Federica, for their guidance and support.

A big thankyou to my dedicated proof reading team, who are now experts in App Store Analysis, Lynn, Steve and Aja. Your attention to detail has never wavered.

And to Anne, your drive, motivation and passion for science inspire me.

To support this PhD, I'm very grateful to have been in receipt of funding from the DAASE project grant.

To everyone mentioned, without each of you none of this would have been possible. My heartfelt thanks.

Contents

Chapter 1

Introduction

App stores are a recent phenomenon: the largest, Apple's App Store and Google Play, were launched in 2008, and were joined by Blackberry World App Store in 2009, and Windows Phone Store in 2010, in addition to numerous smaller stores. The largest stores have, since 2008, both accumulated in excess of 1 million downloadable and rateable apps. Complementing their large collections of hosted apps, mobile app stores are extremely lucrative: the set of online mobile app stores were projected to be worth a combined 25 billion USD in 2015 [181], a number that is sure to grow in the coming years, as smartphones become ever more pervasive [69], even in developing countries [257].

The vast success of app stores has been driven by mass consumer adoption. Google announced that there were 1.4 billion activated Android devices in September 2015 [39]. Smartphones existed prior to the launch of these stores, but it was not until 2008 that users could truly exploit their extra computing power and resulting versatility through downloadable apps. In-house and even commercial applications had been available before the launch of app stores, but app stores had some differences: availability, compatibility, ease of use, variety, and user-submitted content [188].

It is the user-submitted content that fundamentally distinguishes app stores from the ad-hoc commercially available applications that existed beforehand. As a result, software engineering researchers have access to large numbers of software applications *together* with customer feedback and commercial performance data, unavailable in previous software deployment mechanisms. Furthermore, through readily available downloadable toolkits, users can write their own applications to make use of a smart device's hardware. They can subsequently publish their software in the central app store for users to download (and possibly pay for). This publication process is subject to the store's in-house review and certification policies but, in general, apps and app updates can be made available quickly (typically within hours/days).

Following the aforementioned inception of the major app stores Google Play and Apple App Store, and mass consumer adoption, there has followed a collection of innovative and insightful research papers centred on the study of mined collections of apps from app stores. This field of research is called "App Store Analysis", and it is the primary focus of this thesis. A comprehensive review of the literature is undertaken in Chapter 2, identifying the key research papers that have advanced the field, and tracking the growth of the field up until November 2015.

In Chapter 3, the methodology used throughout this thesis for data collection and analysis is introduced. We describe the statistical analysis techniques for dataset comparison and correlation analysis, and how they are used. We give an overview of the data collected to facilitate the studies in this thesis, as well as how it was collected, and any metrics extracted via information retrieval. We also give an overview of an unsupervised machine learning approach called topic modelling, and detail the text pre-processing techniques that are performed prior to its application.

The driving question throughout this PhD was "what makes successful apps successful", which is tackled from multiple perspectives throughout this thesis. The uniform hosting of apps by app stores makes them a good source for data extraction. We can extract technical information, such as the size and supported devices for each app, and non-technical information, such as descriptions. App descriptions can provide a large amount of information about apps that we would otherwise not be aware off. We can apply algorithms in order to extract the technical features that developers claim are present in an app, in addition to other information such as new or added content. In Chapter 4, we show that topic modelling can be used to extract technical features from app descriptions in a way that is analogous to n-gram based feature extraction. Performing correlation analysis on the mean metrics from each topic yields almost identical results to the same approach applied to features. An in depth look at the apps with 0 and 1 ratings reveals that they are special cases and must be treated carefully when conducting empirical studies.

Mining data from app stores can also present issues: app store owners may choose what data to make available, and in which order to present the apps. These choices can lead to sampling issues, such as the "App Sampling Problem", where app samples typically reflect only the most popular apps in a store, and only a cross section of the submitted user feedback. This is discussed in more detail in Chapter 5, where we define dataset types of varying completeness in order to investigate the issue, and conduct experiments to weight the potential effect of the problem on results. It is important that researchers become aware of the issue, and seek ways to ameliorate the problem.

Understanding what is present in apps at any given time is useful, but perhaps even more critical to understanding how apps are successful is to look at how they change over time. Apps that are hosted in app stores often adopt frequent releasing strategies for various reasons, such as stimulating user downloads and reviews, fixing bugs or adding new content. By mining app data over a period of time, it is possible to identify the points in time (the releases) at which significant increases or decreases in performance are realised. This is done using a technique called causal impact analysis, that trains a statistical model on the time series data of both the app in question, and a non-releasing control set, in order to detect where a significant change has occurred that may be a result of the new release.

In Chapter 6, we use this technique to identify the releases that have had the most impact on their respective apps' subsequent user rating success, and identify common factors among them. One such finding is that amongst the releases of non-free apps, it is the new releases of expensive apps that are most likely to cause a positive change. We introduce `CIRA`, a tool for applying causal impact analysis to app store datasets, and elicit the feedback from app developers of positive and negative impactful releases, who agree with the observed impacts, and express interest in learning from the results of these studies. Furthermore, we conduct an examination and comparison of the features present in impactful and non-impactful releases, and positive and negative releases, in order to reveal the features that are added, or removed, in apps that achieve better rating performance after their releases.

This thesis is structured as follows: Chapter 2 performs a comprehensive literature survey on the field of App Store Analysis for software engineering; Chapter 3 discusses the analysis techniques used throughout this thesis; Chapter 4 studies the use of topic modelling as a method analogous to feature extraction, for the analysis of textual app data; Chapter 5 presents the app sampling problem, as well as a review of literature related to app reviews, datasets used, effects of the app sampling problem and ways to ameliorate them; Chapter 6 introduces causal impact analysis, and the tool `CIRA` for performing this analysis on app store data; Chapter 7 presents studies using this analysis to identify the properties and features of app releases that correlate with the greatest subsequent impacts, both positive and negative. Chapter 8 concludes this thesis, and additional supplementary content can be found in Appendices B to D.

Chapter 2

Literature Review

This is to be published in TSE [186] (pre-print is available at the time of writing). The first author's contribution to this paper was to formulate the idea, implement and execute the experimentation and collect the results and analyse them; other authors of the paper contributed to research question formulation, result analysis and narrative write up.

App Store Analysis studies information about applications mined from app stores. App stores provide a wealth of information derived from users that would not exist had the applications been distributed via previous software deployment methods. App Store Analysis incorporates this non-technical information with technical information to learn trends and behaviours within these forms of software repositories. Findings from App Store Analysis have a direct and actionable impact on the software teams that develop for app stores, and have led to techniques for requirements engineering, release planning, software design, security and testing. This survey highlights and compares the areas of research that have been explored thus far, drawing out new directions future research could take to address open problems and challenges.

2.1 Introduction

This chapter provides a survey of literature that performs "App Store Analysis for Software Engineering" between 2000 and November 27, 2015. Although we do not expect app store analysis literature to date back prior to the launch of the major app stores in 2008, 2000 was chosen as the starting point to ensure that no relevant literature had been missed, such as literature focused on earlier platforms such as Nokia Widsets [222]. November 27, 2015 was chosen as the end point as the search queries were performed starting on November 28, 2015.

2.1.1 Contributions

The contributions of this study are as follows:

i) Formal definitions of apps, stores, and technical and non-technical attributes, which are used for App Store Analysis research.

ii) Growth patterns of App Store Analysis literature are studied both overall, and in each emergent subcategory.

iii) Analysis of the scale of app samples used, and discussion on how this is likely to progress in the future.

iv) Identification of some of the key ideas published in App Store Analysis, in addition to common aspects, trends and future directions, to help readers to understand the progression of the field overall.

2.1.2 Definitions

The following definitions help to clarify key components of App Store Analysis literature. They are used to find all of the relevant literature.

App: An item of software that anyone with a suitable platform can install without the need for technical expertise.

App Store: A collection of apps that provides, for each app, at least one non-technical attribute.

Technical attribute: An attribute that can be obtained solely from the software.

Non-technical attribute: An attribute that cannot be obtained solely from the software.

Examples of attributes are shown in Figure 2.1, based on the data that was collected to facilitate the studies in Chapters 5 and 6. As the diagram shows, some attributes are distinctly technical or non-technical in a boolean sense, but others lie in a grey area, depending on the precise interpretation of what can be obtained from software alone. Those in the grey box cannot be considered technical in the strictest sense of the definition, because they are not guaranteed to be obtainable solely from the software in all cases. These attributes can be both non-technical and technical, depending on how they are obtained. They are attributes that, in some cases, can be provided by the developer and not the app store, whilst attributes that are strictly non-technical may only be provided by an app store. For example, consider the 'author' attribute. In the case of Android software, the author can be obtained solely

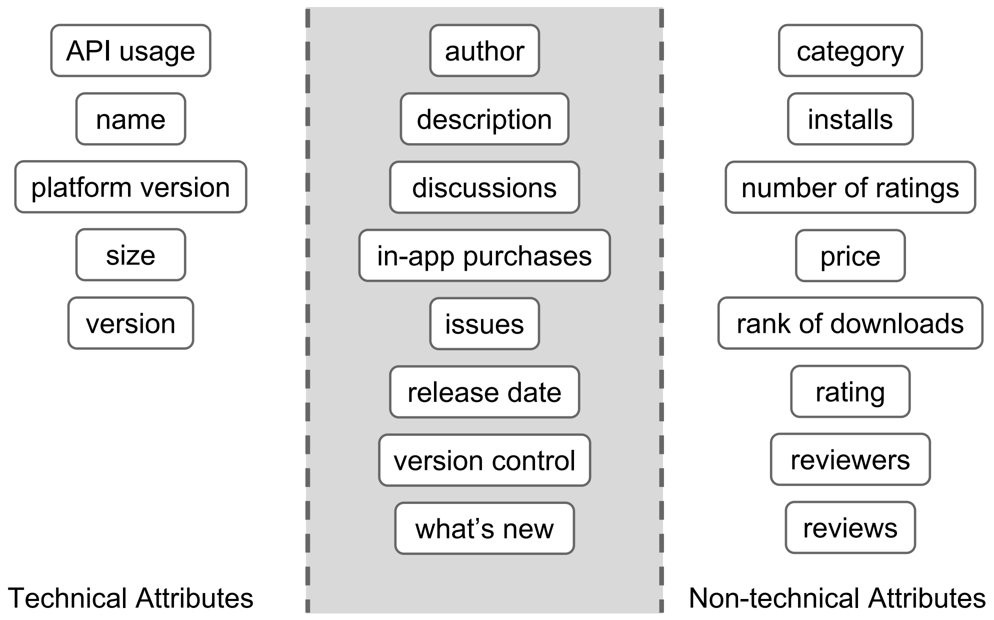


Figure 2.1: **Example attributes**[186]
Left: attributes which are strictly technical
Right: attributes which are strictly non-technical
Centre grey area: attributes which may be in either category

from the distributed `apk` file. However, in the case of a compiled `C` binary such as a simple "hello world" program, the author cannot be obtained directly from the binary file. The 'author' attribute therefore belongs in the grey area. The size of a `C` binary can be obtained, and so this attribute is technical; one cannot obtain the price from either of these example files, and so this is a non-technical attribute.

This definition of App Store may seem simplistic. However, at the time of writing, app stores serve as more than just collections of apps; they enable more developers than ever to produce and distribute content, and enable a communication channel between users and developer via reviewing systems. Therefore, the definition is aimed at inclusivity. In only 7 years since the launch of the two biggest app stores, there are already over 180 papers devoted to their study, and each of these stores has well over 1 million apps each. As this rapid development shows, the concept of apps and app stores is very likely to evolve over the coming years. It is our aim to encompass this evolution as best as possible through the stated definitions, in the hope that future surveys will be able to build upon this work and the App Store Analysis literature to come.

2.1.3 Overview

This survey is structured as follows: Section 2.2 describes the process used to find the included literature; Section 2.3 breaks down the growth trends in non-technical research compared with technical-only research, and Section 2.4 breaks down the growth of scale of apps used; key ideas in each subfield of app store analysis are identified in Section 2.5.

The following App Store Analysis subfields are defined, based on the literature gathered through the process explained in Section 2.2: "API Analysis", which is discussed in Section 2.6; "Feature Analysis", which is discussed in Section 2.7; "Release Engineering", which is discussed in Section 2.8; "Review Analysis", which is discussed in Section 2.9; "Security", which is discussed in Section 2.10; "Store Ecosystem", which is discussed in Section 2.11; and "Size and Effort Prediction", which is discussed in Section 2.12. An inclusion checklist for future app store analysis authors is outlined in Section 2.13.

Closely related work that does not meet the inclusive criteria, but are highly related, are discussed in Appendix A.

2.2 Literature Search

In this section, the process used to find literature is described, including scope, search terms and repositories and lessons learned for future app store analysis surveyors.

2.2.1 Scope

App Store Analysis literature encompasses studies that perform analysis on a collection of apps mined from an App Store. We are particularly interested in studies that combine technical with non-technical attributes, as these studies pioneer the new research opportunities presented by app stores. However, studies are also included whose authors use app stores as software repositories, to validate their tools on a set of real world apps, or by using specific properties such as the malware verification process apps go through before being published in the major app stores.

This survey is not a Systematic Literature Review (SLR). The area of App Store Analysis is still developing, and at the time of writing does not have a well-defined body of literature, with the exception of the subset of app review analysis, which

Khalid et al. have helped to identify through their survey [141]. Because there is not a well defined body of literature for app store analysis as a whole, and no prior surveys, research questions can be chosen and asked. This study therefore aims to define, collect and curate the disparate literature, arguing and demonstrating that there does, indeed, exist a coherent area of research in the field that can be termed "App Store Analysis for Software Engineering". We hope that this will prove to be an enabling study for future SLRs in this area.

The following *inclusion criteria* are applied:

i) The paper is related to software engineering, and may have actionable consequences for software users, developers or maintainers.

ii) The paper is related to mobile app stores, concerning the use of collections of apps or non-technical data gathered from one or more app stores.

The following *exclusion criteria* are applied:

i) The paper focuses on mobile app development but does not extend to collections of apps nor to app stores.

ii) The paper uses an arbitrary collection of apps to test a tool, but the collection was not mined from an app store, and the study does not extend to app stores (and could not clearly be extended to apply to app stores).

2.2.2 Search Methodology

In order to collect all relevant literature to date that meets the scope defined in Section 2.2.1, a systematic search was performed for the terms defined below, from each repository (also defined below). Unique papers are collected into a table, and a decision is made based on the inclusion and exclusion criteria in three stages:

Title: Publications are removed that are clearly irrelevant from their titles.

Abstract: The abstract is inspected and publications are removed that are clearly irrelevant according to the scope defined in Section 2.2.1.

Body: Results are read fully and a judgement is made on whether the paper a) meets the key requirements on what is defined as "app store analysis" in our scope, or b) is very relevant to the field and so should be included as "expanded literature", to put the main literature into context. Papers matching the requirements of a) or b) are included in this survey.

A summary of the number of papers found through the search, as well as the number of papers accepted at each stage of validation, can be found in Table 2.1. All of the references for papers discussed in this survey are available in an online repository [246].

Search Repositories

A search was performed in each of the following repositories for papers to include in the study: Google Scholar, Scopus, JSTOR, ACM, IEEE and arXiv.

Terms

Searches were performed for the following terms and phrases, to encompass the sub-fields of App Store Analysis that emerge from the literature: "App Store", mining, API, feature, release, requirements, reviews, security, and ecosystem. Searches were performed for the following specific queries, where terms joined by an 'AND' must appear, and phrases in quotes must appear verbatim:

"app store analysis"

"app store analysis" AND mining

"app store analysis" AND mining AND API

"app store analysis" AND mining AND feature

Searches were then performed for the following more general searches to ensure that no relevant literature was missed from the survey:

"app store" AND analysis AND API

"app store" AND analysis AND API AND mine

"app store" AND analysis AND feature AND mine

"app store analysis" AND mining AND requirements

"app store analysis" AND mining AND release

"app store analysis" AND mining AND reviews

"app store analysis" AND mining AND security

"app store analysis" AND mining AND ecosystem

The threat of missing papers is mitigated by conducting searches for "app store analysis" AND "mining" and also each of the names of each of the major subfields of App Store Analysis literature. Since, by the definition in Section 2.1.2, app store analysis research uses collections of apps, this should encompass much of the field. Snowballing was also performed, which further helps to mitigate the threat of potentially

Table 2.1: **Specific search query results**[186]

	“app store analysis”	“app store analysis” AND mining	“app store analysis” AND mining AND API	“app store analysis” AND mining AND feature	“app store analysis”	“app store analysis” AND mining	“app store analysis” AND mining AND API	“app store analysis” AND mining AND feature
	Google Scholar				**IEEE**			
Hits	35	17	9	13	3	40	13	13
Inspect	35	17	9	13	3	40	13	13
Title	15	13	8	12	3	8	8	8
Abstract	13	13	8	12	3	7	4	4
Body	12	13	8	12	3	5	4	4
	ACM				**JSTOR**			
Hits	7	1,146	295	231	0	36	4	13
Inspect	7	1,146	295	231	0	36	4	13
Title	4	69	44	31	0	0	0	0
Abstract	3	57	27	22	0	0	0	0
Body	3	44	26	17	0	0	0	0
	arXiv				**Scopus**			
Hits	0	81	28	10	1	128	21	1
Inspect	0	81	28	10	1	128	21	1
Title	0	4	1	0	1	128	21	1
Abstract	0	4	1	0	0	13	6	0
Body	0	4	1	0	0	11	4	0

missing papers. However, the potential threat of missing papers is a threat to the validity of any survey, including this one.

2.2.3 Snowballing

In addition to the repository searches specified in Section 2.2.2, snowballing [296] is also performed on the included studies. To do this, the studies cited by each included study are inspected, as well as the publications that subsequently cited the included study, using Google Scholar and ACM. By performing this process in addition to repository keyword searching, we reduce the risk that relevant literature is omitted from this survey.

2.2.4 Search Results

Search results can be found in Tables 2.1 and 2.2. The tables indicate the number of hits each query generates, the number of these that were available to be inspected, the number of titles and subsequent abstracts and paper bodies that were accepted as valid. Table 2.1 indicates more specific queries run in multiple paper repositories,

Table 2.2: **General search query results**[186]

	“app store” AND analysis AND API	“app store” AND analysis AND API AND mine	“app store” AND analysis AND feature AND mine	“app store analysis” AND mining AND requirements	“app store analysis” AND mining AND release	“app store analysis” AND mining AND reviews	“app store analysis” AND mining AND security	“app store analysis” AND mining AND ecosystem
	Google Scholar							
Hits	3,130	409	1,040	12	9	15	9	9
Inspect	1,000	409	1,000	12	9	15	9	9
Title	87	35	37	12	9	14	8	9
Abstract	61	23	33	12	9	14	8	9
Body	52	21	32	12	9	14	8	9

and Table 2.2 indicates the more general queries run only in Google Scholar. In the case of Google Scholar, only the top 1,000 results were accessible to inspect at the time of search due to the repository’s limitations.

An overlap was found between search queries performed, and thus the total number of discovered papers through search queries was fewer than suggested by the sum of the bottom rows in Table 2.1. Many papers were discovered through snowballing, which do not appear in the Table.

A summary of the included literature in Tables 2.4 to 2.10. Histograms depicting the growth of publications studied on App Store Analysis for software engineering can be found in Figures 2.2 to 2.4, which show the split between technical-only and technical and non-technical research, the split between different subfields identified as subsections in this survey, and the split between scale of studies in terms of the number of apps used, respectively. Numbers in the box plots indicate the number of studies in each subgrouping, and boxes without numbers had one study. A breakdown of these studies in each sub-field that emerged from the literature is also presented in Figure 2.5.

The earliest reported study is 2010. This is likely because the App Stores that propelled mobile app usage to become widely adopted were launched in 2008. Yet, it is interesting that studies incorporating technical with non-technical app store in-

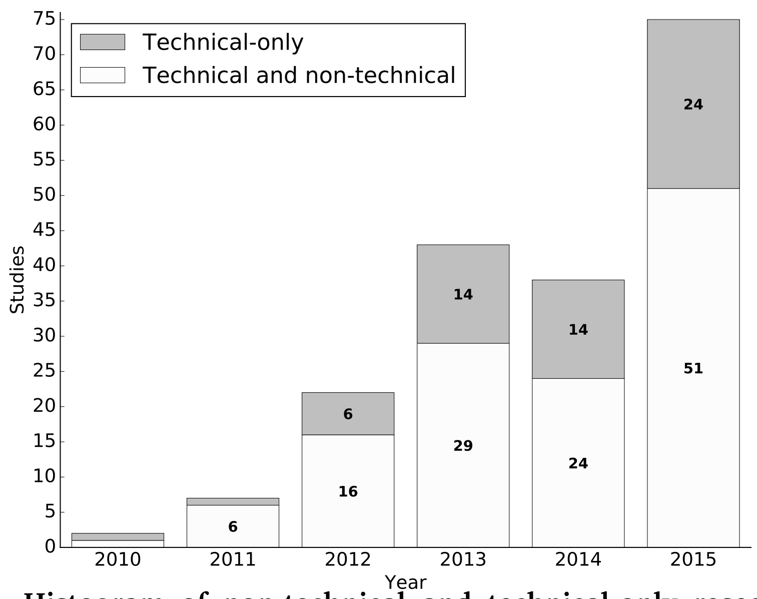


Figure 2.2: **Histogram of non-technical and technical-only research papers** showing the period from 2010 to November 27, 2015[186]

formation did not emerge until two years later. Papers were collected until November 27, 2015.

2.2.5 Lessons Learned

As can be seen from Table 2.1, for some queries, there were large drops in the number of papers upon inspection of their title or abstract, when performing the more general searches on Google Scholar: searches for “app store” with many of the combinatoral words resulted in several thousand papers which may have mentioned “app store” only once. We found that searching for “app store analysis” as a phrase narrowed the results down a lot, but did miss some relevant papers.

Searches that included “mining” as a keyword did encompass much of app store analysis research due to the focus on collections of apps that meets our app store definition. However, we found that the snowballing technique was crucial in the literature search, because paper discovery through many of the paper repositories used could not be relied upon to find all relevant papers; in a growing field, terms of reference are not fully stabilized. Future surveyors are encouraged to visit the App Store Analysis paper repository [246], which can assist in the discovery of app store analysis literature.

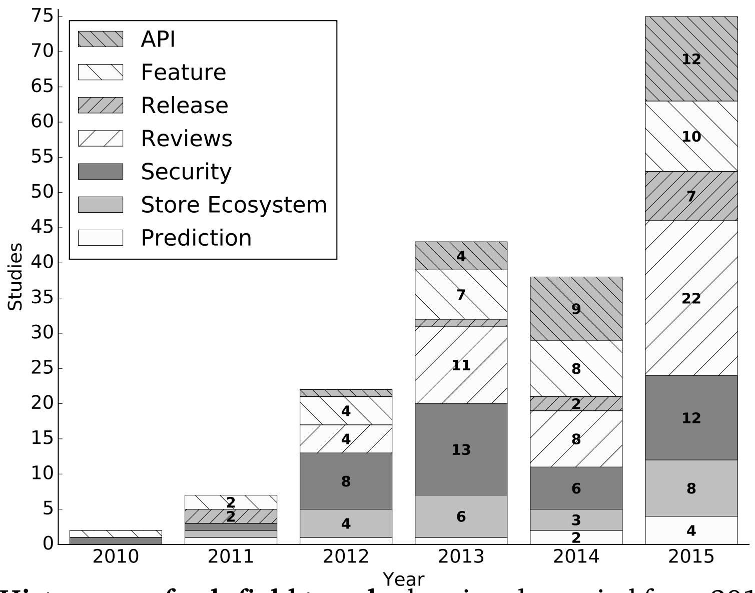


Figure 2.3: **Histogram of sub-field trends** showing the period from 2010 to November 27, 2015[186]

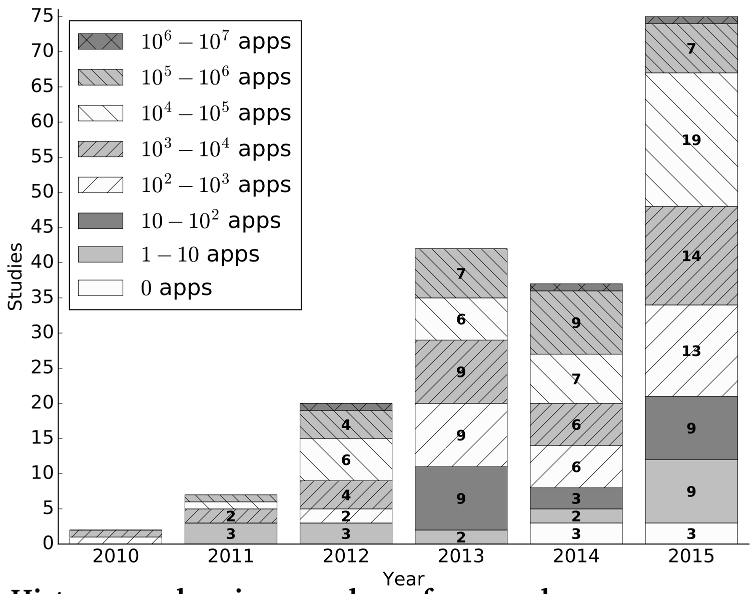


Figure 2.4: **Histogram showing number of research papers grouped into app quantity ranges** showing the period from 2010 to November 27, 2015[186]

2.3 Non-Technical Research

While software engineering deals primarily with code, it is not confined to deal with strictly technical sources of information. One can combine data from multiple (technical and non-technical) sources, and app stores provide a wealth of such information. There are 124 of 184 (68%) papers included in this study that incorporate non-technical information mined from app stores in order to either infer technical

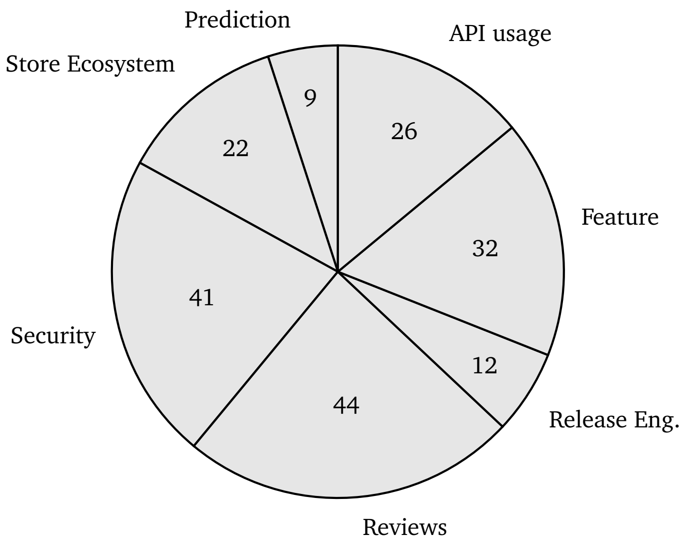


Figure 2.5: **Pie chart showing overall sub-field distribution** showing the period from 2010 to November 27, 2015[186]

Table 2.3: **Number of research papers studying each app quantity range** from 2010 to November 27, 2015[186]

No. Apps Range	Papers	No. Apps Range	Papers
0	6	$[10^3, 10^4)$	36
$[1, 10)$	19	$[10^4, 10^5)$	38
$[10, 10^2)$	21	$[10^5, 10^6)$	28
$[10^2, 10^3)$	31	$\geq 10^6$	3

attributes (such as features), or to extract useful information such as bug reports and feature requests from users.

The histogram in Figure 2.2 shows that the number of studies incorporating non-technical information grew year-on-year, with the notable exception of 2014. It is unknown what caused the apparent drop in literature in 2014, but with the exception of this outlier there was linear growth in App Store Analysis research between 2010 and 2015.

2.4 Scale of Studies

In order to discuss the number of apps that are studied by research papers, we first need to define a set of ranges. Papers are assigned to app quantity ranges in ascending powers of 10, according to the number of apps that they consider. The ranges that we assign, and the number of research papers that study them, are shown in Table 2.3.

The median number of apps used in the considered literature is 1,679, and the mean is 44,807. This result shows that half of the papers study fewer than 2,000 apps, but the other half study a quantity of apps several orders of magnitude larger. This is reflected in Figure 2.4, where the range $[10^4, 10^5)$ is shown to grow and in 2015 represents almost half of the app usage literature.

Figure 2.4 shows growth in the the studies using between 10^4 and 10^5 apps from 2011 to 2015, and this result is reflected in studies using between 10^5 and 10^6 apps as well, up to 2014. It is important to note that we did not have complete data for 2015, so this result is subject to change. Studies using smaller scales of apps show an uncertain change in frequency, indicating that most studies in the future are likely to continue using over 10^4 apps. We anticipate larger studies in the future, based on the growth of App Store Analysis literature, the increasing quantity of apps studied, and of course the growing app stores themselves.

2.5 Key Ideas Timeline

A timeline depicting the key ideas is shown in Figure 2.6. This highlights the launch of major app stores studied, as well as the first studies in each subsection.

The timeline includes studies that have advanced the field of App Store Analysis in some way (based on the qualitative judgement of the author of this thesis), or introduced influential ideas into their respective subsection and are therefore well-cited in this subsection.

2.6 API Analysis

Papers that extract the API usage from app APKs or source code, and combine this information with non-technical data are discussed in this section, and are summarised in Table 2.4. All API analysis literature studied apps from the Android platform only. This may be due to the availability of tools which can be used to decompile the apps and extract their API calls, which are freely available and can be applied to downloaded app binaries. It is perhaps surprising that such analyses have not also been performed on the Apple platform, iOS, since the store was launched in 2008.

2008	Apple App Store launched
	Google Play launched (as Android Market)
2009	Blackberry World App Store launched
2010	[253] Shabtai et al. extracted feature information to differentiate between Tools and Games categories (**Feature**)
	[31] Bläsing et al. used the Android Market as a software repository for testing their APK analyser (**Security**)
	Windows Phone Store launched
2011	[260] Syer et al. compared Blackberry and Android platforms using feature equivalent apps (**Store ecosystem**)
	[151] Lee et al. analysed deployment strategies for optimising download rank (**Release engineering**)
	[113] Henze and Boll studied release times in the Apple App Store for game deployment (**Release engineering**)
	[252] Sethumadhavan discussed function point analysis for mobile apps (**Size and Effort Prediction**)
2012	[96] Goul et al. analysed reviews in order to facilitate requirements engineering (**Reviews**)
	[47] Chandy and Gu mined 6,319,661 reviews from the Apple App Store for spam classification (**Reviews**)
	[109] Harman et al. connected non-technical, technical & business aspects; extracted technical features from text descriptions (**Feature**)
	[240] Ruiz et al. studied class reuse and inheritance in Google Play (**API**)
	[311] Zhu et al. studied the problem of mobile app classification (**Feature**)
2013	[280] Vision Mobile found 72% of developers dedicated to Android; iOS and Android developers earn twice that of developers using other platforms
	[158] Lim and Bentley simulated an app store ecosystem (**Store ecosystem**)
	[124] Iacob and Harrison automatically analysed app reviews to identify feature requests and bug reports (**Reviews**)
	[313] Zhu et al. studied ranking fraud in App stores (**Security**)
2014	[238] Ruiz et al. investigated the effect of ad libraries on app ratings (**API**)
	[95] Gorla et al. performed malware detection through API usage/description cluster outliers (**Feature**)
	[281] Vision Mobile paid and with-ads models almost tied in revenue and developer share
2015	[191] McIlroy et al. studied update frequency of Google Play apps (**Release engineering**)
	[177] Malavolta et al. investigated hybrid mobile apps from technical and user perspectives (**Reviews**)
	[219] Park et al. studied the mobile app retrieval problem (**Reviews**)
	[247] Sarro et al. studied feature migration between apps (**Feature**)
	[141] Khalid et al. surveyed app store review literature (**Reviews**)
	[78] Ferruci et al. compared approaches for size and effort prediction in mobile apps (**Size and Effort Prediction**)

Figure 2.6: **Key ideas timeline** for App Store Analysis literature. The primary area of study is suffixed in parentheses[186]

Table 2.4: **Chronological summary of API-related literature**[186]

Authors [Ref], Year	**Venue**	**No. apps**
Ruiz et al. [240], 2012	ICPC	4,323
Linares-Vásquez et al. [164], 2013	FSE	7,097
Shirazi et al. [241], 2013	EICS	400
Minelli and Lanza [195], 2013	ICSM	20
Minelli and Lanza [196], 2013	CSMR	20
Ruiz et al. [238], 2014	IEEE Soft.	236,245
Hao et al. [107], 2014	MobiSys	3,600
Dering and McDaniel [71], 2014	MILCOM	450,000
Linares-Vásquez et al. [167], 2014	MSR	24,379
Ruiz et al. [237], 2014	IEEE Soft.	208,601
Linares-Vásquez [163], 2014	ICSE comp.	0
Viennot et al. [276], 2014	SIGMETRICS	1,107,476
Bartel et al. [19], 2014	IEEE Soft. Eng.	1,421
Zhang et al. [303], 2014	WiSec	10,311
Borges and Valente [35], 2015	PeerJ C. S.	396
Bavota et al. [22], 2015	IEEE Soft. Eng.	5,848
Kim et al. [145], 2015	ASE	350
Khalid et al. [137], 2015	IEEE Soft.	10,000
Watanabe et al. [293], 2015	SOUPS	200,000
Zhou et al. [309], 2015	WiSec	36,561
Wan et al. [287], 2015	ICST	398
Wang et al. [288], 2015	ISSTA	105,299
Syer et al. [261], 2015	Soft. Qual.	5
Azad [16], 2015	Masters Thesis	950
Wang et al. [289], 2015	UbiComp	7,923
Seneviratne et al. [250], 2015	WiSec	4,114
	Mean	93,298
	Median	5,086

This might be because iOS binaries are only available for the intended platforms, and cannot be downloaded to, or used from a desktop computer without an Apple Developer account, which is not free. Even with such an account, app binaries or source code would be needed, and neither are freely available due to a) copyright on binaries and b) many iOS apps being paid-for apps. Due to these difficulties, it is uncertain whether it will be possible for future studies to extract API information from iOS apps; in fact, it may become harder since the move (in iOS version 9) to developer-submitted LLVM [172] intermediate representation binaries, that are then compiled for specific platforms by Apple. The Windows Phone platform is relatively recent, and we may start to see API analysis studies utilising this platform; the Google Play store launched in 2008 (as Android Market), but it was not until 2012 that App Store Analysis literature studied API usage in the store.

All API analysis literature has, hitherto, studied apps from the Android platform. There is large range in the number of apps considered, from 0 apps to over 1 million.

API analysis literature can be decomposed into "API Usage", "Class Reuse and Inheritance", "Faults" and "Permissions and Security". There is some overlap between the latter subsection and Section 2.10 on Security. Nevertheless, the literature discussed in this section is collected together and discussed here because it directly analyses API usage. Two papers included in this section relate to energy usage [165, 287], although much of this field of research relates only indirectly to app stores. For those who wish to learn more on the subject, we point the reader to the recent paper by Hindle [114].

2.6.1 API Usage

This subsection concerns studies into the usage of APIs, which can be extracted from Android APK files.

Borges and Valente [35] used association rule mining to infer API usage patterns, using a dataset of 396 open source Android apps. For their study, the authors extended `APIMiner` [199] to mine usage patterns and instrument API documents with extracted usage patterns. They reported a study over 17 months, during which instrumented Android documentation was made publicly available, and received approximately 78,000 visits. Shirazi et al. [241] extracted the API usage with regards to user interface (UI) elements and layout, and compared statistics between the 21 different categories of the Google Play store that existed in 2012.

Wan et al. [287] explored energy hotspots in apps by transforming their UIs and producing a ranked list of UI components by energy consumption. The authors tested their approach on 398 apps mined from Google Play. Azad [16] studied apps mined from Google Play and F-droid, and produced tools to inspect API usage and suggest similar APIs based on Stackoverflow discussions; score the similarity of apps; identify the degree to which apps have copied the source code of open source projects; and detect license violations. Tian et al. [269] extracted API information and evaluated apps in terms of code complexity, API dependency, API quality, as well as a number of other factors, in order to train features to distinguish high from low rated apps.

2.6.2 Class Reuse and Inheritance

This subsection concerns studies into the reuse of code and classes, and use of inheritance in Android apps.

In 2012 Ruiz et al. [240] studied class reuse and inheritance in 4,323 Android apps mined from 5 categories in Google Play. Of these, 217 apps were found to contain exactly the same set of classes as another app in the same category. The study was later extended to 208,601 apps by Ruiz et al. [237] in 2014. More evidence of substantial code reuse was found, and the authors concluded that app developers benefit from increased productivity but risk dependence on the quality of the code they reuse.

In 2013 Minelli and Lanza presented a visual analytics web tool for studying repositories of apps [195, 196]. The tool analyses snapshots of apps throughout their version history, using an interactive graphical user interface. Following their subsequent study on 20 free and open source Android apps, the authors found that 3rd party API code is often (incorrectly) committed along with the app code, instead of including the corresponding 3rd party `jar` files. Excluding 3rd party code, most apps were found to have comparatively small code-bases. Additionally the authors found little use of inheritance in Android apps, and much duplication. Viennot et al. [276] introduced the `PlayDrone` Google Play crawler, which they used to store daily data on 1.1M apps and decompile 880k free apps. The authors found that native libraries are heavily used in popular apps, and that approximately a quarter of free apps are duplicates of other apps. They found that paid apps account for just 0.05% of downloads, and the top 10% of most popular apps account for 96% of total downloads as of June 23, 2013.

Linares-Vásquez et al. [167] decompiled and analysed 24,379 APKs from Google Play, and found that the 82% of detected clones replicate 3rd party libraries. Zhang et al. [303] proposed `ViewDroid`, an app plagiarism detection system that uses view transition graphs as “birthmarks” to capture app behaviour, in order to detect clones in the presence of code obfuscation. Apps mined from Google Play were used as a false negative set. In a related study, Kim et al. [145] scan API invocations to identify plagiarised applications, in a more sophisticated approach than similarity detectors that scan code, as it handles code obfuscation. Wang et al. [288] proposed `WuKong`, a two-phase Android clone detection system that first filters third-party libraries to increase detection speed. The authors tested the system on 105,299 Android apps and found zero false positives.

2.6.3 Faults

This subsection concerns studies into faults that are found to be present in Android apps, typically through automated procedures.

Linares-Vásquez et al. analysed the effect of fault and change-prone core Google APIs on app ratings [164]. This is an important study as it combines technical API information with non-technical information in the form of average user reviews, in order to assess the impact that API usage can have. Fault and change prone APIs were found to be used more frequently by poorly-rated apps. Conversely, popular apps used APIs that were found to be less susceptible to faults and changes. The paper presents an analysis of 7,097 randomly selected free apps with $>$ 10 reviews. Changes and faults were measured as the number of API changes and bug fixes, respectively, to the particular associated core libraries.

Building on the work by Linares-Vásquez et al. [164], Linares-Vásquez also presented an approach for a recommendation system for Android app developers [163], to help them to prepare for platform updates and avoid breaking changes and introducing bugs. The authors extended their API analysis work to identify APIs that have a high energy usage [165], but this study did not combine non-technical app store information.

Bavota et al. [22] investigated how the number of changes and faults present in APIs used affected apps' ratings. Their results showed an inverse correlation between the popularity of apps and the number of faults and changes in APIs they used. That is, low rated apps were found to use APIs that are more fault- and change-prone than highly rated apps. Bavota et al. surveyed 45 Android developers who confirmed this relationship from anecdotal experience. These studies combined technical (API usage) with non-technical (user ratings) information to highlight best practice for API usage in Android development.

Syer et al. [261] studied the effect of platform independence on source code quality, finding that the more defect prone source files also depend more heavily on the platform. The authors therefore suggest prioritising platform-dependent source files for unit testing, as a quality assurance strategy. In 2015, Khalid et al. [137] performed static analysis on 10,000 free Google Play apps, and found that 3 categories of `FindBugs` warnings occur more frequently in lower rated apps. The categories 'bad practice', 'internationalisation' and 'performance' had more warnings in lower-rated apps, suggesting that these areas are the ones developers should focus on to achieve better rating performance.

2.6.4 Permissions and Security

This subsection concerns studies into the permissions usage, and related security concerns in Android apps.

In 2013 Peiravian and Xingquan [220] used API calls and permissions data to train their malware classifier, which they trained and validated on 1,260 malware samples and 1,250 benign samples, using cross-validation. Hao et al. [107] studied the insertion of UI handlers into app code. They published the `PUMA` tools which makes UI automation programmable, and enables researchers to analyse correctness properties of apps. They tested the tool on a set of 3,600 apps downloaded from Google Play. Dering and McDaniel [71] downloaded a set of 700,000 app binaries from 450,000 free apps on Google Play and analysed library and permission usage. They found a strong correlation between the number of libraries used and the number of permissions requested by the apps, leading to the conclusion that libraries tend to have specific use cases that require additional permissions from the user. This finding presents a security concern: is each library doing what it is supposed to, and does it need this permission? In conjunction with the finding by Book et al. [34], this suggests that library usage is a significant security concern, since libraries often make use of existing permission privileges, and also increase the number of permissions requested.

Ruiz et al. studied the effect of advertisement libraries on app ratings [238]. They combined non-technical rating information with the extracted technical information showing advertisement library usage to perform the study. Advertisement libraries query their host server at regular intervals to fetch advertisements for display, and this interval determines the “advertisement fill rate”. Multiple libraries are often used to obtain higher fill rates in order to increase revenue. From a sample of 236,245 apps, the authors found no evidence of a correlation between rating and the number of advertisement libraries. However, certain APIs were found to have low median ratings from apps that used them. The authors state that this is due to intrusive behaviours, such as recording entered passwords.

Gorla et al. [95] trained a one-class support vector machine [178] on API usage information in order to identify outliers in trained clusters for security purposes. Bartel et al. [19] showed that off-the-shelf static analysis is insufficient for permission-protected API methods, and investigated alternatives, which they tested on 1,421 apps downloaded from two Android markets. Watanabe et al. [293] found, from analysing the description and API usage of 200,000 Android apps, that there is disparity between their descriptions and requested permissions. This is due to a combination of factors: unnecessary permissions requested by app building frameworks, or developers that use similar manifests for multiple app projects; secondary functionality that is not mentioned in descriptions; and the use of 3rd party libraries. In a related study, Zhou et al. [309] mined a set of 36,561 Android apps, and proposed the tool `CredMiner` which is focused on decompilation and program slicing. They identified over 400 apps that leaked developer user-names and passwords, required for the program to execute normally.

Wang et al. [289] decompiled 7,923 apps from Google Play and mined features from the decompiled code and variable names. They trained a machine learning classifier on labelled instances of the apps using location and contact information, in order to identify the way in which sensitive information is used. Seneviratne et al. [250] studied 275 free and 234 paid Android apps, and found that paid apps collect personal information, in the same way as free apps do. 60% of the paid apps collected personal information, compared to 85% in free apps. The authors subsequently showed that 20% of 3,605 collected Android apps were connected to more than three trackers.

2.7 Feature Analysis

Papers that extract feature information from either technical or non-technical sources of information are discussed in this subsection, and are summarised in Table 2.5. These research papers study a wide range of platforms: Android, iOS, Nokia Widsets, Blackberry and Windows Phone. In addition, the publications investigate a large number of apps: the minimum is 3 and the maximum is 600,000.

Features have been extracted from app descriptions, API usage, manifest files, decompiled source strings, categories and permissions.

Papers in this section show that it is possible to extract feature information from sources other than source code or requirements lists. Additionally, many different methods are used for extraction and categorisation of features, including natural language processing, topic modelling and clustering. The work shows that analysis of app collections can be augmented with meaningful technically-oriented information, mined from freely-available app store pages.

Many studies in Section 2.9 are relevant to this subfield of research (in particular the studies discussed in Sections 2.9.3 and 2.9.4 which discuss the extraction of feature requests and causes of complaints). However, those studies are not discussed in this section because they use user reviews to extract this information, whereas the studies discussed in this section do not.

Feature Analysis literature is broken down into “Automated Classification”, “Clustering”, “Lifecycles”, “Recommendation”, “Store Success” and “Verification”.

Table 2.5: **Chronological summary of feature-related literature**[186]
a: Apple App Store
b: Blackberry
g: Google Play or other Android stores
n: Nokia (or Widsets) platform
s: Samsung (Android)
w: Windows Phone

Authors [Ref], Year	**Store**	**Venue**	**No. apps**
Shabtai et al. [253], 2010	g	CIS	2,285
Chen and Liu [49], 2011	a	iConference	102,337
Coulton & Bamford [59], '11	n	MobileHCI	3
Harman et al. [109], 2012	b	MSR	32,108
Sanz et al. [242], 2012	g	CCNC	820
Teufl et al. [265], 2012	g	MobiSec	130,211
Zhu et al. [311], 2012	n	CIKM	680
Mokarizadeh et al. [198],'13	g	WEBIST	21,065
Teufl et al. [264], 2013	g	Sec. & Com. Netw.	443
Lulu and Kuflik [25], 2013	g	IUI	120
Bhattacharya et al. [29],'13	g	CSMR	24
Yin et al. [302], 2013	a	WSDM	5,661
Lin et al. [161], 2013	a	SIGIR	7,116
Ihm et al. [127], 2013	g	CGC	10
Iacob et al. [126], 2013	g	BCS-HCI	161
Iacob et al. [125], 2014	g	MobiCASE	270
Kim et al. [146], 2014	a	Service Business	100,830
Finkelstein et al. [79], 2014	b	Tech. report	42,092
Yang et al. [300], 2014	g	Tech report	26,703
Zhu et al. [312], 2014	n	TMC	680
Zhu et al. [314], 2014	g	KDD	170,753
Jiang et al. [129], 2014	g	INTERNETWARE	150
Zhu et al. [310], 2014	a	IEEE Cybernetics	15,045
Vakulenko et al. [271], 2014	a	ICIS	600,000
Lin et al. [162], 2014	a	SIGIR	6,524
Maalej and Nabil [174], 2015	a,g	RE	1,140
Guzman et al. [102], 2015	a,g	ESEM	7
Guzman et al. [103], 2015	a,g	ASE	7
Sarro et al. [247], 2015	b,s	RE	54,983
Berardi et al. [27], 2015	a,g	SAC	5,993
Svedic [259], 2015	a	PhD Thesis	60
Seneviratne et al. [251],'15	g	WWW	232,906
Tong et al. [270], 2015	g,w	JCST	10,000
Wang et al. [289], 2015	g	UbiComp	7,923
He et al. [110], 2015	g	Big Data	122,875
Tian et al. [269], 2015	g	ICSME	1,492
Nayebi and Ruhe [207],'15	g	PeerJ C.S.	241
Lulu and Kuflik [26], '15	g	MOB INF SYST	6,633
		Mean	52,084
		Median	6,579

This section has an overlap with Section 2.10, in the cases where features are used to help detect anomalies or verify app functionality.

2.7.1 Automated Classification

This subsection concerns the automated classification of apps, using machine learning classifiers that are trained using feature information. The source of feature information varies between studies.

Shabtai et al. [253] extracted feature information from the manifest, `XML` files, API calls and methods used from a set of 2,285 Google Play apps. They trained a classifier on the features to differentiate between Tools and Games categories, as a proof of concept that malware detectors could be trained in the same way. In 2012 Sanz et al. [242] trained machine learning classifiers to predict app categories, using extracted features. The features used for prediction were strings extracted from the decompiled app code, requested permissions, rating, number of ratings and app size. They tested the approach on 820 apps and found a peak AUC (area under ROC curve) of 0.93 using the Bayesian TAN classifier [81].

Zhu et al. [311, 312] studied the problem of mobile app classification in the Nokia Store. The authors mined 680 apps, and experimented by classifying apps using data from web search and from device logs from users of the apps. Their approach outperformed other classification techniques, and enabled them to automatically classify a given app onto a predefined category of Apple's App Store taxonomy. In 2015 Berardi et al. [27] built on this work, by constructing a classifier using features mined from app descriptions, categories, names, ratings and file sizes. They trained the classifier using a support vector machine for each of 50 classes, and used the BM25 weighting scheme [235] on the features. Users manually classified 5,993 apps mined from Apple App Store and Google Play, to act as the training (cross validation) set for the classifier.

Jinh et al. [132] used the features: numbers of app installs, number of reviews, category and rating score, in conjunction with features based on information flow, for their machine learning classifier for rating app security risk. Wang et al. [289] extracted features from decompiled `Java` code, from their collection of 7,923 apps mined from Google Play. They used the extracted features to train classifiers for predicting how ‘location’ and ‘contact’ information is used, with 85% and 94% accuracy, respectively.

2.7.2 Clustering

This subsections concerns studies that perform app clustering using feature information.

Teufl et al. [265] mined 130,211 apps from Google Play and performed clustering on both app descriptions and requested permissions, as part of their activation patterns malware detection approach. They later extended this work [264] to propose a first-step malware detection method using links between description terms and security permissions to identify suspicious outliers. In 2013 Mokarizadeh et al. [198] performed clustering on 21,065 apps, mined from Google Play, after applying topic modelling on app descriptions. They found that the resultant clustering was very different from the app’s assigned categories, and apps in the same category often had dissimilar description topic distributions. Mokarizadeh et al. also performed correlation analysis and found that users download free apps more frequently, and that downloads correlate with the number of ratings an app has received.

Lulu and Kuflik [25] performed clustering on 120 apps mined from Google Play, comparing their description-based clustering with the app store’s pre-assigned categories. They found that descriptions provide good clustering features, and present the method as the basis of an app recommendation system. The authors later built on this work [26], by extracting features from 6,633 app descriptions and enriching them with information mined from the web, found by searching for the app name. They used the enriched features to provide an installed-app recall interface, supported by functionality-based categorisation. The interface was validated by performing a user study with 40 participants, who were able to find apps faster and found the categorisation more intuitive, when compared with a reference “smart launcher” interface [88]. Kim et al. [146] mined 100,830 apps from Apple App Store, and extracted feature keywords from their descriptions using natural language processing. They clustered apps using the extracted features, and re-categorised them using the resulting clusters.

2.7.3 Lifecycles

This subsection concerns the tracking of features between multiple apps.

Sarro et al. [247] proposed a theoretical characterisation of feature lifecycles in app stores, to help app developers to identify trends and to find undiscovered requirements. In order to investigate app feature migratory and non-migratory behaviours in current app stores, they mined features from app descriptions using the techniques in the earlier work [109], and used the proposed theory to empirically analyse the migratory and non-migratory behaviours of 4,053 non-free features from Samsung and Blackberry stores. The results revealed that features generally migrate to a category that has similar characteristics. However, there are also a few features that migrate to apparently non-related categories. The early identification of these features may allow developers to find undiscovered requirements. The authors also found that approximately one third of features were intransitive (they neither migrate nor do they die out over the period studied), and such features exhibited significantly different behaviours with regard to important properties, such as their price. Being aware of which are the intransitive features in a given category may support developers in identifying crucial ('must-have') requirements for their apps.

2.7.4 Recommendation

This subsection concerns recommender systems that are trained using feature information.

Yin et al. [302] proposed the Actual Tempting (AT) model to perform app recommendation for users. The model incorporates latent tempting parameters. Take for example two apps, "a" and "b". The AT model incorporates the number of users who own app "a" and subsequently download app "b", and the number who do not download "b" after owning "a". The model also uses feature overlap information, measured by performing topic modelling on app descriptions and computing the topic overlap between each pair. The authors find that the AT approach outperforms collaborative filtering and case-based reasoning in their initial experiments.

Lin et al. [161] used topic modelling on the Twitter messages of users that follow an app's Twitter feed, in order to generate latent groups related to the app. The groups are then used as part of a recommendation system, in order to help remove the problem of cold start in app recommendation based on other metadata. The system was tested on 7,116 apps mined from Apple App Store and the authors found that it outperformed recommendation using app descriptions. However, in 2014 Lin et al. [162] used topic modelling on app descriptions in order to produce a recommendation system. The model is semi-supervised and incorporates app version information using different weights corresponding to update types: so that newer app versions can be recommended when they add a certain feature to the description. Resultant topics are weighted based on their category in the app store to provide a recommendation. The model was trained on 6,524 apps mined from the Apple App Store.

Zhu et al. [310] mined the daily top 300 free and top 300 paid apps from Apple App Store charts from February 2, 2010 to September 17, 2012, collecting information on 15,045 apps in total. They used popularity information to construct a Popularity-based Hidden Markov Model (PHMM), to encode trend and other latent factors. The authors state that this can be used in a variety of ways, including app recommendation, review spam detection, and demonstrate its usefulness in ranking fraud detection. Zhu et al. [314] built an app recommendation system using a combination of technical information (device permissions requested) and non-technical information (app popularity). They tested the system on 170,753 apps mined from Google Play to show its scalability. However, the system received no human-based evaluation of its recommendations.

Valulenko et al. [271] performed topic modelling on a set of 600,000 app descriptions mined from Apple App Store. They used the resultant topics to suggest categories, and to improve and augment existing categorisation approaches used in app stores. He et al. [110] trained a system for targeting users for advertising, with a dataset containing app install data on a per-user basis, consisting of 122,875 apps from the Huawei App Store. The authors reported a higher click rate than current targeting approaches. Nayebi and Ruhe [207] extracted feature information from 241 Google Play apps, and used crowd-sourcing to assign user value to each of the features. The authors used the approach for service portfolio planning [3].

2.7.5 Store Success

This subsection concerns studies that analyse the success of apps in their hosting app store. Success is measured as rating, as well as the total number of ratings or rating frequency, downloads or download rank and bug reports.

In 2011, Coulton and Bamford [59] conducted a case study on games created for the WidSets platform, an earlier app store that targets Nokia phones (including non-smartphones). Their findings are transferable to modern app stores: high download numbers are required in order to gain active users, and popular features such as chat can increase popularity and the proportion of active users. Chen and Liu [49] collected 102,337 apps from Apple App Store, and observed no correlation between download rank and rating, from a sample of the top 200 most popular apps.

Harman et al. [109] introduced app store mining as an MSR (Mining Software Repositories) problem. They mined app information and performed correlation analysis on price, downloads, and rating. Correlations analysis was performed in both app and feature space, where features were extracted using natural language processing techniques from app descriptions, and results showed that under most conditions there is a strong correlation between rating and downloads (popularity). The proposed approach can be applied to different app stores by modifying the data extraction and parsing phases to accommodate the different app store structure and data representations. The authors later extended this work [79], finding that free apps have higher ratings than non-free apps, with a medium effect size. They also carried out a developer survey on the extracted features, who found them meaningful, and were able to successfully detect the extracted features over randomly generated features.

Bhattacharya et al. [29] presented an empirical study of 24 open source Android apps from multiple categories, with the aim of defining metrics of bug report quality and developer involvement. The authors showed how the bug-fix process is affected by differences in bug lifecycles. Security bug reports were found to be of higher quality, but the associated bugs are fixed more slowly. The scale of the study was large as all apps had more than 1,000 ratings, 100,000 downloads and 200 bug reports. The authors found that bug report quality correlates with description length but not app rating.

Ihm et al. [127] conducted a study on 10 popular apps in the Google Play store, analysing the correlations between app downloads in the store and external metrics. The authors found a strong positive correlation between the number of downloads in the store and the number of registered users on the app's respective websites, and a strong correlation between the number of downloads and the app website (inverse) download rank. Jiang et al. [129] conducted a user survey on 50 app descriptions in order to identify the attributes most important to the quality of the description. A support vector machine was trained on the resultant attributes and tested on a sample of 100 descriptions, finding an accuracy of 0.62. The findings showed that quick overviews are the most effective form of app description, and the study contains further heuristics on good description styles.

In a longitudinal study on 60 paid iOS apps, Svedic [259] found that ratings and reviews can impact sales ranks. The study found that higher, more stable ratings lead to users associating the app with high quality, and the app sales increased as a result. Tian et al. [269] studied 1,492 high and low rated apps from Google Play, and identified the features which most accurately differentiate apps with high rating from those with low rating. The authors used technical features, such as code complexity and API usage, with non-technical information such as the category and the number of images displayed on the app store page. The most important features for differentiating high from low rated apps were the size of the app and the number of images on store page. The target SDK version was also an influential feature, suggesting that high rated apps are updated more frequently and use more modern features of the Android operating system.

2.7.6 Verification

This subsection concerns the verification of claimed features in app descriptions, matched to the performance or code calls in the app software.

Yang et al. [300] introduced the APPIC framework, which extracts main theme tag words from Android description and permission files. It does this using LDA and Partially Labelled Dirichlet Allocation (PLDA), for the purpose of identifying misleading app descriptions. It uses an app's permissions file to establish whether its description makes claims consistent with its functionality, and whether it resides in the appropriate category. The method was tested on 207,865 apps from Google Play, and was manually evaluated on a subset of 1,000 apps. The authors found that their method achieved (average) 88.1% category accuracy, and 76.5% permissions accuracy.

Watanabe et al. [293] found that apps often contain secondary functionality that is not mentioned in their descriptions. In a study of 232,906 apps, Seneviratne et al. [251] trained a machine learning classifier on app features in order to detect spam apps. The features used for the classifier were numeric statistics about an app's description. The authors labelled apps that were removed from the store and established potential reasons for removal. Apps that were likely to have been removed due to being spam (the majority of those removed) were then used to train a boosting classifier in order to identify potential spam.

Tong et al. [270] proposed the `App Generative Model` (AGM) topic model, for extracting semantically coherent app features from descriptions, using term co-occurrence statistics. The AGM model resulted in lower perplexity [286] (a fitness function for probabilistic models, that measures the trained model's log-likelihood of generating a held-out test set; the lower the better), than the most commonly used model, LDA. However, the model precision was evaluated only against TF.IDF, and not LDA or similar topic models such as the weighted topic model [194]. Nevertheless, the study shows the importance of accurate feature discovery and representation, and can help lead to future studies using extracted features.

2.8 Release Engineering

This section discusses papers that focus on app releases or release strategies, which are summarised in Table 2.6. The results in Table 2.6 show that there were two papers published in 2011 that tackled this issue, one in 2013, and then a steady increase prior to November 27, 2015. Release studies typically require time series data, in order that the changes made to apps in their releases can be recorded. Due to the 2015 spike in release engineering studies, we expect the trend to continue and contribute to the growing numbers of App Store Analysis literature. As can be seen in Table 2.6, the stores studied are split almost equally into Apple and Google, but there are no release studies in Blackberry or Windows Phone Store. The scale of

Table 2.6: **Chronological summary of release engineering-related literature**[186]
a: Apple App Store
g: Google Play or other Android stores

Authors [Ref], Year	Store	Venue	No. apps
Lee and Raghu [151], 2011	a	AMCIS	3,168
Henze and Boll [113], 2011	a	MobileHCI	24,647
Datta and Kajanan [67], 2013	a	CloudCom-Asia	3,535
Lee and Ragu [152], 2014	a	JMIS	7,579
Ruiz et al. [239], 2014	g	IEEE Soft.	120,981
Guerrouj et al. [100], 2015	g	SANER	154
Comino et al. [55], 2015	a,g	Tech report	1,000
McIlroy et al. [191], 2015	g	ESE	10,713
Gui et al. [101], 2015	g	ICSE	21
Carbunar and Potharaju [41], 2015	g	ASONAM	160,000
Alharbi and Yeh [6], 2015	g	MobileHCI	24,436
		Mean	29,772
		Median	5,557

the past studies in this section is relatively small, ranging from 21 to 160,000; this scale is not surprising, given the difficulty of mining longitudinal data for a large number of data points.

Release Engineering literature has featured Apple and Google platforms but not yet Blackberry, Samsung or Windows. The scale in studies has been small, most likely due to the difficulty in obtaining time series data.

Release Engineering literature is broken down into “Content”, “Success” and “Strategy” subsections.

2.8.1 Content

This subsection concerns studies into the content of app releases.

The 2014 study by Ruiz et al. investigated the updates made to update advertisement libraries [239]. They found that over 12 months, almost half of the 5,937 apps with multiple updates had an advertisement library update. Approximately 14% of advertisement updates contained no changes to the app’s code, indicating the effort involved in keeping advertisement libraries updated. Gui et al. found, from 21 apps in Google Play with frequent releases, that 23% of their releases contained ad-related changes [101].

The findings of Guerrouj et al. [100] indicate that high code churn in releases correlates with lower ratings. Alharbi and Yeh [6] tracked the design patterns used by 24,436 Android apps over a period of 18 months. They found that depreciated patterns are sometimes adopted after they are depreciated, and that new pattern adoption rates are low. By tracking the app descriptions, they found that authors sometimes update the app descriptions to reflect changes to their design patterns. They believe that this shows that descriptions are used as a communication channel between developers and users. The authors report on apps that start and stop using certain design patterns. An interesting future research direction might be to record the migration of these “design features” using the app feature migration terminology of Sarro et al. [247].

2.8.2 Success

This subsection concerns studies into the relationship between app releases and the store success of apps, as measured by rating, downloads or popularity.

Moller et al. [204] studied the installation behaviour of users with recently updated apps, in a security related study. Lee and Raghu [152] studied the factors that affect an app’s likelihood of staying in the top (most popular) charts in the Apple App Store. They found that free apps were more likely to ‘survive’ in the top charts, and that frequent feature updates were the most important factor in ensuring their survival, along with releasing in smaller categories. The authors also found that high volumes of positive reviews improved an app’s likelihood of survival.

Carbunar and Potharaju [41] conducted a longitudinal study on 160,000 Google Play apps mined daily over a 6 month time period in 2012. They found that at most 50% of apps were updated in each category, and that there is an issue of “stale apps” affecting aggregated statistics on large populations. The authors also found that a few developers dominated the total download counts, that productive developers did not have many popular apps, and that there was no correlation between price and downloads.

2.8.3 Strategy

This subsection concerns releasing strategies for app deployment and updates.

In 2011 Lee et al. [151] mined app information from the top 300 iOS apps in all 21 categories free and paid, mining at least 3,168 apps. They analysed developer diversification through publishing apps in multiple categories and in both free and paid, and found a positive relationship between download rank and app portfolio diversification. The study incorporates technical (download rank) with non-technical information (category, price), in order to identify actionable findings for app developers.

Henze and Boll [113] analysed release times and user activity in the Apple App Store, and concluded that Sunday evening is the best time for deploying games. Their study also found that version updates were an effective strategy for raising an app's rank in the store. Datta and Kajanan [67] studied review counts from the Apple App Store, and found that apps receive more reviews after deploying updates on Thursdays.

In 2014 Lin et al. [162] incorporated version information in their app recommendation system, in order to ensure that apps are recommended if they add new features to new versions. Comino et al. [55] studied the top 1,000 apps in Apple App Store and Google Play. They found that for iTunes, increased numbers of app releases are more likely when the app is performing badly, and that releases can boost downloads. Neither finding held true for Google Play, however, which the authors speculate might be a symptom of "excessive updating" caused by a lack of quality control from Google.

McIlroy et al. [191] studied update frequencies in the Google Play store, after mining data about 10,713 mobile apps. They found that only 1% of the studied apps received more than one update per week, and only 14% were updated in a two-week period. The authors also found that rating was not affected by update frequency. Nayebi and Ruhe [207] combined app features with values gained from crowd-sourcing as an approach to app service portfolio planning.

Table 2.7: **Chronological summary of reviews-related literature**[186]
a: Apple App Store
b: Blackberry
g: Google Play or other Android stores

Authors [Ref], Year	**Store**	**Venue**	**No. apps**
Hoon et al. [120], 2012	a	OzCHI	17,330
Vasa et al. [274], 2012	a	OzCHI	17,330
Chandy and Gu [47], 2012	a	WebQuality	3,090
Goul et al. [96], 2012	a	HICSS	9
Ha et al. [105], 2013	g	CCNC	59
Oh et al. [212], 2013	g	CHI	24,000
Hoon et al. [119], 2013	a	Tech report	17,330
Iacob and Harrison [124], 2013	g	MSR	270
Galvis Carreño et al. [83], 2013	g	ICSE	3
Khalid [138], 2013	a	ICSE	20
Fu et al. [82], 2013	g	KDD	171,493
Chen et al. [51], 2013	a,g	WWW	5,059
Pagano and Maalej [214], 2013	a	RE	1,100
Hoon et al. [118], 2013	a	OzCHI	25
Iacob et al. [126], 2013	g	BCS-HCI	161
Iacob et al. [125], 2014	g	MobiCASE	270
Khalid [140], 2014	a	IEEE Soft.	20
Chen et al. [50], 2014	g	ICSE	4
Cen et al. [45], 2014	g	PIR	6,938
Guzman and Maalej [104], 2014	a,g	RE	7
Khalid et al. [139], 2014	g	FSE	99
Wano and Iio [292], 2014	a	NBIS	500
Erić et al. [75], 2014	a	QIP	968
Khalid et al. [141], 2015	g	IJITCS	0
Gao et al. [84], 2015	g	SOSE	4
McIlroy et al. [192], 2015	a,g	ESE	12,000
Cen et al. [44], 2015	g	SIAM	12,783
Vu et al. [284], 2015	g	ASE	3
Vu et al. [283], 2015	g	CoRR	95
Malavolta et al. [176], 2015	g	MS	11,917
Malavolta et al. [177], 2015	g	MOBILESoft	11,917
Park et al. [219], 2015	g	SIGIR	43,041
Panichella et al. [218], 2015	a,g	ICSME	7
Palomba et al. [215], 2015	g	ICSME	100
Moran et al. [201], 2015	g	FSE	14
Gomez et al. [92], 2015	g	MOBILESoft	46,644
Maalej and Nabil [174], 2015	a,g	RE	1,140
Pérez [278], 2015	g	Masters Thesis	4
Khalid et al. [142], 2015	-	IJIEEB	0
Gu and Kim [99], 2015	g	ASE	17
Guzman et al. [102], 2015	a,g	ESEM	7
Guzman et al. [103], 2015	a,g	ASE	7
McIlroy et al. [193], 2015	g	IEEE Soft.	10,713
Liang et al. [154], 2015	a	IJEC	139
		Mean	9,594
		Median	161

2.9 Review Analysis

Literature discussed in this section concerns the study of app reviews; a summary of discussed literature can be found in Table 2.7. The results in Table 2.7 show that the majority of studies focus on the Google Play store, with a minority focusing on Apple App Store, and one paper studying Blackberry store. App review centred literature first emerged in 2012, and has subsequently gained significant (and increasing) interest and activity: Figure 2.3 shows that there are greater numbers of requirements/reviews literature each year. We hypothesise that this is due to the tenure of the stores, and the progression of the field. One avenue of research that has not been attempted is the study of reviews in the Windows Phone Store, which was launched in 2010 but has not achieved the widespread success of Google Play and Apple App Store, in the competitive market.

Review Analysis literature mostly studies Apple and Google stores, inviting future comparison with Windows and other store reviews.

Review Analysis literature is broken down into “Classification”, “Content”, “Requirements Engineering”, “Sentiment”, “Summarisation” and “Surveys and Methodological Aspects of App Store Analysis”. Many early works have focused on the content of reviews in 2012-2013, before advancing to sentiment in 2013-2014, and requirements and summarisation in 2015.

2.9.1 Classification

This subsection both concerns the classification of user reviews, and studies that have trained classifiers on user reviews.

Chandy and Gu [47] mined 6,319,661 reviews from 3,090 apps in the Apple App Store. After manually labelling a subset of the mined reviews as spam or not spam, the authors trained both a supervised decision tree and unsupervised latent class analysis to identify spam reviews. The unsupervised method achieved higher accuracy, and took into account factors such as average rating of a user, and number of apps rated.

Chen et al. [51] compared the maturity ratings of 1,464 equivalent apps between the Apple App Store and Google Play, and taking the Apple store ratings as the accurate ratings, the authors found that 9.7% of the Android apps were underrated and 18.1% were overrated. The authors also studied a sample of 729,128 reviews from 5,059 Google Play game apps, and trained a classifier on the sets of app descriptions, user reviews, and iOS maturity ratings, to automatically verify app maturity ratings. Ha et al. [105] manually examined 556 reviews mined from 59 Google Play apps, in order to classify them into topics and sub-topics based on content. They found that most information in reviews concerns the quality of the app, and not security or privacy concerns.

Cen et al. [45] devised an approach to identify the Comments with Security / Privacy Issues (CSPI) from a set of mined Google Play app reviews. The authors later built upon this work, using reviews in order to rank the security risk of apps, by detecting security labels in a crowd-sourcing approach [44]. Using `AndroGuard` [9] scores as a ground truth, the authors found that their tool outperformed other metrics for ranking app security risk, half of which incorporated user reviews and half of which relied on declared permissions.

Gomez et al. [92] used an unsupervised machine learning approach in order to identify apps that may contain errors, using 1,402,717 reviews mined from 46,644 apps. The authors used the error information in addition to permissions used by the apps, in order to construct a ranked recommender system to analyse app permissions, for app store moderators. Guzman et al. [103] also developed an ensemble of machine learning classifiers in order to classify user reviews. They tested this system on 4,550 reviews mined from 7 apps in the Google and Apple app stores, and achieved a precision of 0.74 and recall of 0.59 on a manually labelled set of 1,820 reviews.

2.9.2 Content

This subsection concerns studies into the content of user reviews, and how store success metrics associate with certain review content.

Hoon et al. [120] and Vasa et al. [274] collected a dataset containing 8.7 million reviews from the Apple App Store and analysed the reviews and vocabulary used. In 2013 Hoon et al. analysed 8 million reviews from Apple App Store [119]. They found that the majority of mobile apps reviews are short in length, and that rating and category influences the length of reviews. The majority of studied apps received under 50 reviews in their first year. Half of the apps analysed decrease in the user assessment of quality, denoted by rating, over time. The authors suggested that user expectations are changing rapidly towards apps, and that developers must keep up with demand to remain competitive.

Iacob et al. [126] studied how the price and rating of an app influence the type and amount of user feedback that it receives through reviews. The authors selected 3,279 reviews for the study, from which they identified 9 classes of feedback: positive, negative, comparative, price related, request for requirements, issue reporting, usability, customer support, versioning. From the selected apps, there was a roughly equal split of positive type reviews with feature/issue type reviews, with very few other types such as negative or price related.

Khalid et al. [139] studied the devices used to submit app reviews, in order to determine the optimal devices for testing. Palomba et al. [215] study the Google Play reviews from 100 open source Android apps, and link the reviews to code changes. They found that a mean of 49% of review requests are implemented in new releases, and that the apps with changes more directly implementing the content of user reviews improve their ratings with new releases. In order to bridge the gap between software attributes and user reviews, Hoon et al. [117] developed an ontology of words used to describe software quality attributes in app reviews.

McIlroy et al. [193] studied responses to reviews from 10,713 Google Play apps, finding that most developers do not respond to reviews. However, in the cases where a response occurred, 38.7% of users were found to subsequently change their ratings, resulting in a median increase in individual user ratings of 20%. A summary of mobile app user feedback classification can be found in the study by Maalej et al. [285].

2.9.3 Requirements Engineering

This subsection concerns the extraction from user reviews of bug reports, user requests and other information that can be used for requirements engineering. The extracted information of course relates not only to reviews, but to the app as a whole.

Oh et al. [212] developed a review digest system which they tested on 1,711,556 reviews mined from 24,000 Google Play apps. They automatically categorised reviews into bug reports, functional requests and non-functional requests, and produced a digest featuring the most informative reviews in each category.

Iacob and Harrison [124] presented an automated system (MARA) for extracting and analysing app reviews in order to identify feature requests. The system is particularly useful because it offers a simple and intuitive approach to identifying requests. The authors used 161 apps and 3,279 reviews for manually training linguistic rules. Additionally, 136,998 reviews were used for an evaluation, that found that 23.3% of reviews contain feature requests. As an extension to the MARA system they had previously introduced [126], Iacob et al. [125] introduced a set of linguistic rules for identifying feature requests and bug reports in order to help facilitate app development.

Wano and Iio [292] analysed the textual content of 856 reviews from 500 apps in the Japanese App Store, and found that the review styles differed between apps in different categories. In a large scale study, Erić et al. [75] studied the star ratings of 48 million reviews from 968 popular free and paid Apple apps. They found that the reviews were mostly positive, and there were significant differences in the distributions between categories, and also between free and paid. Free apps were found to have more reviews but a lower mean rating, and higher standard deviation. Due to the higher numbers of reviews for free apps, which may give an app credibility, the authors argue that in-app purchasing revenue models are a good way to make money for developers, especially if used as a 'teaser' for a paid version.

Park et al. [219] developed AppLDA, a topic model designed for use on app descriptions and user reviews, that discards review-only topics. This enables developers to inspect the reviews that discuss features present in the app descriptions. The authors tested the system on 1,385,607 reviews mined from 43,041 apps. Panichella et al. [218] presented a system for automatically classifying user reviews based on a predetermined taxonomy, in order to support software maintenance and requirements evolution. They verified the system on a manually labelled truth set of 1,421 sentences extracted from reviews, and achieved 0.85 precision and 0.85 recall when training the system on language structure, content and sentiment features. Maalej and Nabil [174] produced a classification method identifying bug reports and feature requests from user reviews. The authors found that upwards of 70% precision and 80% recall can be obtained using multiple binary classifiers, as an alternative to a single multiclass classifier. They also found that the commonly used NLP techniques, stopword removal and lemmatisation, can negatively affect the performance of this classification task.

Moran et al. [201] propose the `FUSION` system, that performs static and dynamic analysis on Android apps, in order to help users complete bug reports. The system focuses on the steps to reproduce a bug, using dynamic analysis to walk through Android system events. Khalid et al. [142], argue that app store reviews can be used for “crowdsourcing” [180]. They argue that users are inadvertently performing crowdsourcing when they review apps, solving the following problems: requests for potential features, suggestions for developer action, recommendations for other users, and issue reporting.

2.9.4 Sentiment

The works discussed in this subsection have incorporated sentiment in their study of reviews. Sentiment describes a user’s views or opinions, typically as positive or negative in the following studies, and is extracted from reviews using ‘positive’ sentiment words such as ‘good,great,love’, or ‘negative’ sentiment words such as ‘bad,hate,terrible’.

In 2012 Goul et al. [96] published the earliest work to study online app store reviews. The authors performed sentiment analysis on 5,000 Apple App Store reviews in order to facilitate requirements engineering. Galvis Carreño and Winbladh [83] extracted user requirements from comments using the ASUM model [133], a sentiment-aware topic model. Initial results showed that the method aids requirements summation with significantly less effort than manual identification, but do not return all possible requirements. Hoon et al. [118] gathered a set of 29,182 short reviews of up to 5 words, from the top 25 Health & Fitness apps in the Apple App Store. They analysed the reviews and found they are mostly made up of sentiment words, and match the star rating of the review closely.

Khalid [138, 140] manually categorised 6,390 negative reviews from a sample of 20 free iOS apps, and reported the most frequent causes of complaints. The apps had combined over 250,000 reviews, and so 6,390 reviews is a statistically representative sample at the 95% confidence level. The authors carried out a manual analysis of the 6,390 reviews, finding that 11% of samples concerned complaints about a recent update. Users were most dissatisfied by issues relating to invasion of privacy and unethical behaviour, while hidden cost was the second most negatively perceived complaint. Pagano and Maalej [214] gathered a sample of 1.1 million reviews from the Apple App Store in order to provide an empirical summary of user reviewing behaviour. They found that most feedback is provided after releases, that positive feedback is often associated with highly downloaded apps, and that negative feedback is often associated with less downloaded apps and often does not contain user experience or contextual information.

In 2014 Chen et al. [50] produced a system for extracting the most informative reviews, placing weight on negative sentiment reviews. Guzman and Maalej [104] studied user sentiments towards app features from a multi-store sample, which also distinguished differences of user sentiments in Google Play from Apple App Store. Guzman et al. [102] developed a tool called `DIVERSE`, that extracts key reviews specific to a queried feature. `DIVERSE` groups together reviews with similar sentiments about the same feature in order to condense the information. The authors tested their tool on the dataset used in their earlier study [104]. Liang et al. [154] performed MultiFacet Sentiment Analysis (MFSA) on user reviews from 139 apps mined from Apple App Store. They reported that opinions on product quality form a larger portion of reviews, but opinions on service quality have a bigger effect on sales.

2.9.5 Summarisation

This subsection concerns the summarisation of large samples of user reviews, using automated tools.

A large sample was used in the 2013 study by Fu et al. [82], in which the authors analysed over 13 million Google Play reviews for summarisation. They designed a system called `WisCom` that enables summarisation of reviews at a per-review, per-app or per-market level. This tool can be useful for large-scale overviews of competitor apps, or for gathering information about a market. The weakness of the system is the need for a large complete sample of reviews to be mined first, and the associated mining difficulties. However, the `WisCom` system enables summarisation of 'complaint' or 'praise' reviews over time, and can produce accurate results given a complete sample in a fixed time period i.e. 6 months, so long as the inherent sample bias is taken into account. The authors found that there was a large difference between free and paid apps, and that paid apps have an associated 'complaint' type about price that free apps do not.

In 2015 McIlroy et al. [192] studied reviews in Google Play and Apple App Store, and developed an automated labelling scheme that can identify multiple elements to reviews that could be beneficial to stakeholders. For example, a review might contain a feature request and a bug report, and so a label for each type would be applied to it. Gao et al. [84] proposed `AR-Tracker`, a similar tool to `AR-Miner` [50], that automatically collects user reviews of apps and ranks them in order to optimise the representation of the review set, in terms of frequency and importance. Pérez [278] mined and labelled 160 user reviews from 5 Google Play apps in order to train a review categorisation tool, that identifies feature requests and bug reports. The tool was evaluated on 400 labelled reviews and achieved 0.78 accuracy.

Malavolta et al. [176, 177] analysed 3 million reviews from 11,917 Google Play apps, and produced a summary of user perceptions about 445 hybrid apps [112] compared with native apps. The authors found that hybrid mobile apps receive similar ratings to native apps, but native apps have been reviewed on average 6.5 times more. They plan to replicate the work using multiple stores and a small set of cross-platform apps to compare their perception across different platforms. Vu et al. [283, 284] developed `MARK`, a system that identifies keywords in sets of reviews in order to assist summarisation and search. The method is one of several summarisation approaches that are applied to reviews.

Gu and Kim [99] proposed `SUR-Miner`, a review summarisation and categorisation tool, that they evaluated on 2,000 sentences from reviews of 17 Google Play apps. The tool is intended for use by developers, and produces a visualisation of the reviews. The authors surveyed the developers of the studied apps, of whom 28 out of 32 agreed that the tool is useful. In the Google Play store it is possible for developers to respond to reviews, which can lead to users changing their rating.

2.9.6 Surveys and Methodological Aspects of App Store Analysis

Khalid et al. [141] reviewed recent literature in app store review analysis, and made several suggestions that could improve the app reviewing process for both users and developers. They suggested that the process could be improved by assigning categories to reviews, and adding sort and filtering functionality based on the assigned category, helpfulness and star rating. The authors also suggest that adding a user reply feature would assist the developers to get the highest quality reviews.

2.10 Security

Studies relating to app security are discussed in this section, and are summarised in Table 2.8. The results in Table 2.8 show that the number of studies grew year on year until 2013 and then remained stable. A large proportion of these papers do not combine technical with non-technical attributes. Instead, they used properties such as the validation that highly rated apps have received, through being downloaded, used, and highly rated by many users. Much of the dedicated security research in this section used the Google Play store. Specifically, there are no studies using Blackberry or Apple, and just one study used Windows. Much of the security-related literature used the property that popular Google Play apps can generally be assumed non-malware, since they are scanned prior to being hosted in the store (by `Google Bouncer` [5]), and they have large user bases. Many studies in this section used large collections (>10,000 apps) of benign apps to help distinguish between benign and malicious behaviour.

The number of apps used ranges from 1 to 1,165,847 (which also happens to be the largest study in this survey).

App security has been well-studied in the literature, and perhaps warrants a survey of its own. There are many studies on mobile app security that used app stores in a less direct way than those discussed in this section, some of which are mentioned in Appendix A.2. Additionally, literature in Section 2.6.4 has identified potential risks associated with permissions, beyond the more direct security threats discussed in this section.

Table 2.8: **Chronological summary of security-related literature**[186]
a: Apple App Store
g: Google Play or other Android stores
w: Windows Phone

Authors [Ref], Year	Store	Venue	No. apps
Blasing et al. [31], 2010	g	MALWARE	150
Batyuk et al. [21], 2011	g	MALWARE	1,865
Potharaju et al. [226], 2012	g	ESSoS	158,000
Moller et al. [204], 2012	g	LARGE	1
Chia et al. [52], 2012	g	WWW	19,344
Gibler et al. [86], 2012	g	TRUST	24,350
Grace et al. [97], 2012	g	WiSec	100,000
Crussell et al. [63], 2012	g	ESORICS	75,000
Peng et al. [221], 2012	g	CCS	500,000
Zhu et al. [316], 2015	g	ICICS	5,685
Bakar et al. [15], 2013	g	ACSAT	5,000
Stevens et al. [258], 2013	g	MSR	10,300
Book et al. [34], 2013	g	CoRR	114,000
Sanz et al. [243], 2013	g	Cybernet. Syst.	333
Sanz et al. [245], 2013	g	SECRYPT	333
Sanz et al. [244], 2013	g	NSS	333
Wang et al. [291], 2013	g	DBSec	272,774
Crussell et al. [64], 2013	g	ESORICS	265,359
Gibler et al. [87], 2013	g	MobiSys	265,359
Peiravian and Xingquan [220], 2013	g	ICTAI	1,250
Chakradeo et al. [46], 2013	g	WiSec	14,888
Pandita et al. [216], 2013	g	SEC	581
Zhu et al. [313], 2013	a	CIKM	15,045
Liu et al. [170], 2014	w	NSDI	51,150
Crussell et al. [65], 2014	g	MobiSys	165,426
Gorla et al. [95], 2014	g	ICSE	32,136
Ham and Lee [106], 2014	g	IJCCE	10
Bhoraskar et al. [30], 2014	g	SEC	1,010
Qu et al. [231], 2014	g	CCS	45,811
Zhu et al. [315], 2015	a	TKDE	15,045
Schütte et al. [248], 2015	g	ConDroid	10,000
Mutchler et al. [203], 2015	g	MoST	998,286
Avdiienko et al. [14], 2015	g	ICSE	2,866
Ma et al. [173], 2015	g	COMPSAC	22,555
Vigneri et al. [277], 2015	g	CoRR	5,000
Yang et al. [299], 2015	g	ICSE	633
Lageman et al. [150], 2015	g	MILCOM	417
Deng et al. [70], 2015	a	CCS	2,019
Zhang et al. [305], 2015	g	CCS	100
Huang et al. [122], 2015	g	SEC	16,000
Chen et al. [48], 2015	g	SEC	1,165,847
		Mean	106,933
		Median	14,888

Security literature is broken down into “Faults”, “Malware”, “Permissions”, “Plagiarism”, “Privacy” and “Vulnerability” subsections.

2.10.1 Faults

This subsection concerns studies that detect faults in Android and Windows apps, that can be potential security concerns.

Ravindranath et al. [233] used a sample of apps mined from Windows Phone Store to run their greybox fault detection tool. They found that 1,138 of the sample of 3,000 apps had failures. Liu et al. [170] presented their `DECAF` system for detecting advertisement placement and layout violations, which can indicate advertisement fraud. They tested the system on 51,150 Windows apps for tablet or phone, and plan to extend it to detect more types of rule violation. The `DECAF` system was used by Microsoft Advertising in 2013 to prompt developers to comply with layout rules.

Crussell et al. [65] presented `MAdFraud`, a system that detects advertisement fraud in the form of requesting ads while the application is in the background, and in the form of simulating user clicks on advertisements. They tested the system on 165,426 apps gathered from Google Play and a separate security company, and found that 30% of apps made advertisement requests while running in the background, and 27 apps simulated user clicks on advertisements. Deng et al. [70] introduced their `iRiS` system, which performs static analysis on iOS apps in order to detect suspicious apps that may violate Apple’s terms of service. The authors detected 146 apps from a sample of 2019, that accessed sensitive user information through use of private APIs.

2.10.2 Malware

This subsection concerns studies into Android malware.

In 2010 Bläsing et al. [31] used the top 150 free Google Play apps to test their static and dynamic APK analyser. They tested these apps against 1 known malware app which was shown to be an outlier, establishing that their approach has the potential for malware detection. Peng et al. [221] proposed an app risk rating system trained on metadata from name, category and set of permissions. The system was trained on a set of 378 malware apps and evaluated on almost 500,000 apps mined from Google Play. Zhu et al. [316] proposed an approach to malware detection that uses permission and description information to detect abnormal permission sets. They evaluated the system on 5,685 apps mined from Google Play and found some words that have a large effect on permission validity; they also tested the system on known malware and found that it was able to successfully detect it as such.

Chakradeo et al. [46] created an app malware triaging tool call `MAST`, that they trained on known malware, and a set of 14,888 apps mined from Google Play (that were assumed to be benign). Peiravian and Xingquan [220] trained a malware classifier using 1,250 samples of known malware, and 1,250 samples of benign apps mined from Google Play. They trained the classifier using information on the permissions requested and the API calls made by the apps.

Sanz et al. [245] used cosine similarity between the sets of features declared in Android manifest files, in order to detect anomalies that might suggest the presence of malware when compared with a benign set. Sanz et al. later trained machine learning classifiers to distinguish between sets of known malware and 333 benign apps mined from Google Play [244, 243]. Similarly, Wang et al. [291] proposed `DroidRisk`, an app trained on sets of known malware and assumed benign software mined from Google Play. `DroidRisk` rates the security risk of other apps in order to help prevent users from installing malware unknowingly. Apps mined from Google Play were assumed benign, as Google's tool `Google Bouncer` [5] is run to detect malware and remove it from the store.

As a means of detecting potentially malicious apps, Gorla et al.[95] performed topic modelling on app descriptions, and then applied K-means clustering to the results to form distinct clusters. Utilising API information from the app manifest, the authors trained a one-class support vector machine (SVM) [178] on each cluster to detect outliers in terms of API usage, which may suggest the presence of malware. This approach was later extended by Ma et al. [173] who used known malware and benignware to train their model, and reported improvements on the resultant precision, recall and F-measure.

Avdiienko et al. [14] extracted information flow data from apps in order to train a benign-trained malware classifier. The classifier was trained on 2,950 of the most popular Google Play apps, which are assumed benign as their download rank is in the top 100 in each of 30 categories. In this way, the authors combined non-technical information (download rank) with extracted technical information (information flow) to detect malware. The system reported high precision on sets of known malware from the Genome project [308] and VirusShare database [279]. In a similar way, the 2013 study by Sanz et al. [243] trained machine learning classifiers to separate known malware and benign apps mined from Google Play. The 2014 study on identifying malicious apps using system call events, by Ham and Lee [106], also used apps from the Google Play Games category as a benign set, against which to compare.

Lageman et al. [150] generated feature sets to be used for classification of malware and benignware, from runtime log datasets of 419 malware apps and 417 mined benign apps. They tested the feature set and achieved a true positive rate of 90% with a Random Forest classifier [90]. In the largest app study to date, Chen et al. [48] ran their `DiffCom` system on 1,165,847 apps mined from Google Play and third party Android stores. `DiffCom` detects malware, including zero-day malware, without prior knowledge of malware, using a simple comparison with known apps in the corpus. The system was tested on a sample of 50,000 apps and achieved a false positive rate of 0.04 and false negative rate of 0.06. When run on the entire dataset, `DiffCom` detected 127,429 instances of malware and 20 likely instances of zero-day malware.

2.10.3 Permissions

This subsection concerns studies into permissions misuse and potential security risks in Android apps from requested permissions.

In 2013 Awang Abu Bakar and Mahmud [15] mined 5,000 apps from the Google Play store and analysed their permissions. They found statistically significant (though weak) Spearman's rank correlation coefficients of 0.13, 0.24 and -0.13 between (technical) the number of permissions asked for and (non-technical) the price, download rank and rating, respectively. They highlight the top permissions requested by apps, and found that 40% of the apps requested the phone's status and identity, a source of sensitive information. Stevens et al. [258] mined 10,300 apps from several Android stores including Google Play and applied the permissions analysis tool `Stowaway` [13] that can detect declared and used permissions. The authors found that 44% of apps in their sample contained at least one unnecessary permission, and computed a Spearman's rank correlation coefficient of 0.72 between the number of search results for a permission on Stackoverflow, and the number of times it is found to be misused. Book et al. [34] studied library permissions on 114,000 apps mined from the Google Play store, showing libraries bundled with apps lead to old versions being included. Increasingly advertisement libraries take advantage of app permissions presenting a potential security risk, which the authors argue should be solved by the app store or privacy legislation.

Pandita et al. [216] presented the `WHYPER` system for automatically extracting the reason a permission is used from the description. They evaluated the system using 581 apps mined from Google Play, which were manually labelled by the authors. The authors tested the system on the permissions address book, calendar and audio recording, and achieved an average precision of 82.2% and recall of 81.5%. In a related study, Qu et al. [231] introduced `AutoCog`, a tool for checking the fidelity between app descriptions and requested permissions. The authors tested the system on 45,811 Google Play apps, and achieved a precision of 92.6% and a recall of 92.0% when detecting 11 permissions.

The findings by Dering and McDaniel [71] suggest that library usage presents a security risk due to permissions usage. This is discussed in more detail in Section 2.6.

2.10.4 Plagiarism

This subsection concerns studies that detect plagiarism in Android apps.

In 2012 Potharaju et al. [226] conducted a study on 158,000 free Android apps, identifying apps that are likely to be plagiarised in order to spread malware. The authors found that the 29.4% of apps with the most permissive rights are most likely to spread malware, and that non-technical information such as category, number of downloads and publishing day can increase the initial spread of the malware. Crussell et al. [63] introduced the tool `DNADroid`, which they used to identify 141 cloned apps in the Google Play store, from a mined set of 75,000 apps. The authors then introduced the tool `AnDarwin`, which decompiles apps and compares them to detect clones [64]. They detected 4,295 cloned apps using this approach from a mined set of 265,239 apps. This dataset is used in the study by Gibler et al. [87], who investigated the effects of application plagiarism on developers.

Zhu et al. [313, 315] mined ranking, rating and review data from 15,045 apps from the Apple App Store. They detected outliers using hypothesis tests in order to find potentially fraudulent apps. They took a unique approach to the issue with app ranks (only the top apps in Google Play, Windows Phone Store and Apple App Store have download ranks), in that they termed the period in which an app has a rank as a 'leading event' and consecutive events as a 'leading session'. Several authors used API information to detect plagiarised apps [145, 288, 303], which are discussed in more detail in Section 2.6.

2.10.5 Privacy

This subsection concerns studies into privacy risks on the Android platform, and in Android apps.

In 2011 Batyuk et al. [21] used the top 1,865 free Google Play apps to test their static APK analyser, which detected that 167 apps accessed private identifiers, presenting a security risk. 114 of these apps wrote the information after reading it, which might indicate that the apps contain spyware. The work has since been extended into a static analysis tool called `Androlyzer` [66]. Chia et al. [52] evaluated the ratings of apps from Facebook, Chrome and Google Play, as a means of warning against privacy risks. They found a strong correlation between popularity and the number of ratings apps receive, but no correlations between permissions sought and privacy risk, nor rating. This result shows that ratings are not an effective indicator of the privacy of apps, and new suspicious apps are not likely to receive many ratings that could serve as potential warnings for future users.

Gibler et al. [86] mapped Android API calls to privacy information, and performed static analysis to identify apps where private data is leaked. Using their tool, `AndroidLeaks`, they analysed 24,350 apps from Google Play and third party stores, and found 2,342 apps with privacy leaks. Grace et al. [97] introduced `AdRisk`, a static analysis tool for identifying potential privacy risks associated with advertisement libraries. From their study on 100,000 apps mined from Google Play, the authors found that 52,067 apps use advertisement libraries, of which 31% use more than one. The authors remarked that the majority of 100 studied advertisement libraries were found to collect personal information.

Vigneri et al. [277] used a set of 5,000 apps mined from Google Play, on which they performed dynamic execution to determine network usage. They focused, in particular, on network activity to URLs which they claimed could present privacy or security risks, such as those associated with tracking, spyware or malware. Network activity was compared both within category and overall, in order to identify apps with suspiciously high activity. The authors noted that a high proportion of apps, even those with high ratings and download ranks, downloaded a large number of advertisements. Huang et al. [122] presented their `SUPOR` system, that detects privacy information entry fields as potential privacy or security risks using static analysis. They evaluated the system on 16,000 apps mined from Google Play, obtaining a precision of 0.973 and a recall of 0.973, with a false positive rate of 0.087. The cases of entry fields found include national ID, username, password, credit card and health data.

2.10.6 Vulnerability

This subsection concerns studies that analyse potential security vulnerabilities from a variety of sources.

Moller et al. [204] studied the update behaviour of users following recent updates, finding from a case study that approximately half of users did not update their app for at least a week after the update. The authors argue that this could lead to users continuing to run vulnerable software even after a fix is available.

In 2015, Zhang et al. [305] argued that the descriptions given to apps contain insufficient security information. The authors presented the `DescribeMe` system, which generates security-centric descriptions using static analysis. They performed a user study using Amazon's Mechanical Turk [7], on a set of 100 apps, asking whether the generated descriptions are readable and whether they can reduce the rate at which users download malware. The generated descriptions achieved a 4% lower readability rating than the original human-written descriptions, but decreased the malware download rate by 39%. Yang et al. [299] used 633 apps mined from Google Play as the benign set to test their tool for distinguishing between malicious and benign apps. They found that the intent of security accesses is more related to whether an app is malicious than the type of security-sensitive resources that it accesses.

Schütte et al. [248] tested their dynamic analysis tool `ConDroid` on the top 10,000 free Google Play apps and found 172 apps suffered from a logic bomb vulnerability, by selectively executing code sections that use vulnerable APIs. Mutchler et al. [203] took a snapshot of 1,172,610 Google Play apps. They found that 998,286 of these apps used `WebView`, indicating that the apps use an embedded `WebView` in some way. The authors searched for several known vulnerabilities and found that 28% of the studied apps had at least one of these vulnerabilities. As a result, the authors propose a set of API changes to mitigate such threats. In a similar study Bhoraskar et al. [30] mined 1,010 apps from Google Play and used static analysis and partial app rewriting to check for known security issues in third party components. They found that 13 of the 200 apps using the Facebook SDK were vulnerable to known attacks, and 175 of 220 children's apps potentially collected information, in violation of the US Children's Online Privacy Protection Act [56].

2.11 Store Ecosystem

This discusses literature that focuses on a store's ecosystem, or the differences between stores. This literature is summarised in Table 2.9.

The scale of studies in this section ranges from 1 to 1,164,489, with a median of 848.

Store Ecosystem literature is broken down into "Inter-store", "Intra-store", "Recommendation" and "Simulation" subsection.

2.11.1 Inter-store

This subsection concerns studies on differences between stores.

In 2011, Syer et al. [260] studied the different code practices between app stores, by selecting 3 pairs of feature-equivalent apps from Android and Blackberry. The authors analysed the source code, code dependencies and code churn of these apps, and found that Android apps are generally smaller but rely heavily on the platform. Conversely, Blackberry apps are larger and rely heavily on 3rd-party APIs. In order to reach the largest customer base developers need to cater for each platform, and so the authors remarked that it is therefore easier to develop for Blackberry and port to Android than vice versa. Syer et al. [262] later compared development prac-

Table 2.9: **Chronological summary of ecosystem-related literature**[186]
a: Apple App Store
b: Blackberry
g: Google Play or other Android stores
w: Windows Phone
*: over 500 simulated apps (final values not specified in the paper)
**: 500,000 simulated apps
***: 1,250,000 simulated apps

Authors [Ref], Year	**Store**	**Venue**	**No. apps**
Syer et al. [260], 2011	b,g	SCAM	3
d'Heureuse et al. [72], 2012	a,b,g,w	MCCR	1,164,489
Jung et al. [135], 2012	a	Market Lett	1,189
Garg and Telang [85], 2013	a	MIS	1,223
Lim and Bentley [157], 2012	a	GECCO	**
Lim and Bentley [156], 2012	a	ALIFE	**
Lim and Bentley [158], 2013	a	CEC	**
Zhong & Michahelles [307],'13	g	SAC	191,301
Petsas et al. [223], 2013	g	IMC	316,143
Syer et al. [262], 2013	g	CASCON	15
McDonnell et al. [190], 2013	g	ICSM	10
Cocco et al. [53], 2014	a	MWIS	*
Wenxuan and Airu [294],'14	a,g,w	ICDMW	736,377
Ng et al. [210], 2014	g	COMPSAC	506
Liu et al. [168], 2015	g	WSDM	6,157
Ruiz et al. [197], 2015	g	IEE Soft.	10,150
Joorabchi et al. [134], 2015	a,g	ISSRE	14
Gómez et al. [91], 2015	g	ICSE NIER	1
Askalidis [12], 2015	a	CoRR	162
Xie and Zhu [297], 2015	a	WiSec	179,353
Corral and Fronza [57], 2015	g	MOBILESoft	100
Lim et al. [159], 2015	a	TEVC	***
		Mean	144,844
		Median	848

tices between 15 Android apps and 5 traditional desktop and server applications. They found that mobile apps are most similar to `Unix` utilities, in terms of smaller code bases and small development teams. However, they also report that mobile apps suffer from greater numbers of defects and slower fix times than the studied traditional applications.

In 2012 d'Heureuse et al. [72] mined 1,164,489 total apps from Apple, Blackberry, Google and Windows app stores at regular intervals over a period of 3 months, in order to perform cross store comparison and also to study growth over time. The authors found that the smaller stores (Blackberry and Windows) had similar rates of growth to the larger stores (Apple and Google), at 2% in the two month time period studied. The smaller stores (Blackberry and Windows) were found to be the most expensive, and all stores displayed a similar power-law curve in price, with many cheap and free apps. Apps that appeared in multiple markets were on average 7.15 MB larger in the Apple store, and were a similar size in the 3 other stores.

Petsas et al. [223] analysed the downloads of 316,143 apps from 4 third-party Android app stores. They found that 10% of the apps accounted for at least 70% of total downloads in the stores, and that user downloads followed a clustering type behaviour, where subsequent app downloads were usually in the same category. The authors also found that popularity followed a power-law distribution against app price, for paid apps. Ng et al. [210] looked into the safety of third-party Android stores by downloading the top apps from Google Play and 20 other third-party Chinese Android app stores. They compared the APKs to check whether they are the same as the official releases, and ranked the severity level of differences. The authors concluded that the third party app stores studied cannot be trusted, as the proportion of apps which do not match their official releases is high, as are the corresponding difference severity levels.

In 2015 Ruiz et al. [197] conducted a longitudinal rating study on 10,150 apps over the period of 12 months. They argued that the Amazon style rating system, in which ratings are accumulated over the lifespan of an app, is too slow to adapt to changes in apps, whose performance is determined by their current release. The current Google Play rating system makes it more difficult for an app to increase its rating with a strong release than, for example, the Apple App Store rating system.

Joorabchi et al. [134] introduced CheckCAMP, a tool that checks for inconsistencies between Android and iOS versions of the same app. The authors tested the tool on 7 open source apps and 7 industry apps, and validated their results with a user study, finding an F-measure of 1.0 on the open source apps and an F-measure of 0.92 on the industry apps.

There are sources of non-technical information that replicate information found on app stores, but provide a more accessible means to gather the data. For example, the study by Syer et al. [261] uses information on the number of downloads from AppBrain, a replication of the number of installs bracket on Google Play (eg. 1,000,000 - 5,000,000 installs appears on AppBrain as 1,000,000+). Ihm et al. [127] combined download information on 10 social networking apps from Google Play with the number of registered users on their respective websites. The authors found a strong correlation between the two metrics.

2.11.2 Intra-store

This subsection concerns studies into the inner workings of individual stores. This includes ranking behaviours, updates to apps and to the platform, source code quality and promotions.

Jung et al. [135] assessed the differences between free and paid apps on Apple's Korean App Store. They found that customer ratings were more critical to the survival of free apps, and there was also a benefit from getting an early entrant in markets. In 2013 McDonnell et al. [190] studied 10 apps using source code from github [89]. The Android platform was shown to be evolving fast with an average of 115 API updates per month, due to which 28% of Android references were out of date, and the median lag time to update to support a new API was found to be 16 months. The APIs used most were the ones updated most frequently, yet interestingly API updates were more defect prone than other changes in client code.

Apps in Google Play do not have accessible information on their total number of downloads, other than 'ranges', such as the range 50-100. Zhong and Michahelles [307] analysed the distributions of download ranges and ratings from 191,301 Google Play apps. They found that a small number of very popular 'blockbuster' apps account for the majority of app downloads, and also have high ratings indicating customer satisfaction. Paid apps achieved more success when they were cheaper, but expensive professional apps had a disproportionally high numbers of downloads. The authors concluded that developers can break into the higher download ranking positions by fulfilling a niche market. Garg and Telang [85] compared paid app demand in the Apple App Store, using download ranks. They found that the top ranked paid app was downloaded 120-150 times more than the 200th ranked app.

Askalidis [12] studied the effects of sales promotions in the Apple App Store on 162 apps. They found that rival apps were able to benefit from a promotion, so long as their promoted price is cheaper than their competition. They authors also found that sales where apps become free, or have easily redeemable digital discounts, were the most successful. Sales were shown to have mixed effects on the ratings of apps. Gómez et al. [91] proposed an app store feature of automatically patching defective apps, which they demonstrated by automatically fixing a defective app mined from Google Play. Xie and Zhu [297] investigated the practice of promoting apps through buying positive reviews, via illicit "underground" services. The authors registered on 8 such app promotion sites and exposed approximately 30,000 promoted apps. Their tool, `AppWatcher` was used to collect information from 179,353 randomly selected iOS apps, from which they mined 9,399,014 reviews. The authors reported on differences between datasets of promoted and random apps.

Corral and Fronza [57] studied 100 open source apps that are available on the Google Play store. They performed correlation and regression analyses between source code quality metrics and the store performance metrics: number of downloads, number of reviewers and average rating. The authors found no strong correlation and no strong regression coefficients, rejecting their initial hypothesis that source code quality plays a role in app success.

2.11.3 Recommendation

This subsection concerns studies into recommender systems using app store information.

In 2014 Wenxuan and Airu [294] used information on the number of downloads and numbers of reviews, as well as the number of apps downloaded by, and reviewed by, participating users. This data was used as part of a recommendation system called Interoperability-Enriched Recommendation (IER), which enables them to recommend similar apps to a user in the Windows Phone Store using data mined from 736,377 Google Play, Apple App Store, and Amazon App Store apps. Liu et al. [168] also studied app recommendation systems, by incorporating the level of privacy that the app needs as well as user interests. They evaluated their approach using 6,157 apps mined from Google Play, and found that their recommender performed better when treating each app function with different privacy allowances. They use the rating distribution over their dataset as the motivation for modelling user preference with a Poisson distribution.

2.11.4 Simulations

This subsection concerns simulations of app stores.

Lim and Bentley simulated the app store ecosystem using an agent-based evolutionary model, in order to experiment with different publicity strategies [156, 157]. The authors modelled apps with infectious properties, that could spread after being downloaded by a user. They found that an ‘app epidemic’ is most likely to occur when the app appears on the ‘new apps’ chart. The authors then used the model to explore different ranking algorithms [158]. They simulated users, and experimented with alternating time periods for updating the ‘new apps’ chart, and the degree to which historical performance factors into the ‘top apps’ chart. The study found that the top apps chart performs best in terms of overall downloads by incorporating fresh apps, and for this to work it needs to incorporate less historical performance data (also found later by Ruiz et al. [197]).

Lim et al. later simulated the ecosystem from a user’s perspective [159], using collected usage information from over 10,000 participants [155]. They modelled developer strategies such as ‘innovator’ (who produces apps with random features) and ‘copycat’ (who copies the app) [159]. They found that ‘optimiser’ (who improves on the original ‘innovator’ apps) and ‘copycat’ working together led to the best overall fitness, provided they represented a low proportion of the overall modelled developer population. Cocco et al. [53] extended the model used by Lim and Bentley, and investigated additional ranking algorithms and user behaviour. They explored store ranking algorithms, and found that a 1% chance of a new app appearing in the top charts leads to the highest downloads-to-browse ratio.

Table 2.10: **Chronological summary of size and effort prediction literature**[186]

Authors [Ref], Year	Venue	No. apps
Sethumadhavan [252], 2011	ISMA	6
Preuss [228], 2012	The IFPUG Guide to IT & Software Measurement	1
Preuss [227], 2013	ICEAA	1
van Heeringen and van Gorp [272], 2014	IWSM-MENSURA	0
Abdullah et al. [1], 2014	ICOS	0
D'Avanzo et al. [68], 2015	SAC	8
Francese et al. [80], 2015	SEAA	23
Ferrucci et al. [77], 2015	SEAA	13
Ferrucci et al. [78], 2015	PROFES	13
	Mean	7
	Median	6

2.12 Size and Effort Prediction

Papers that predict size or effort based on the functionalities offered by an app are discussed in this section, and are summarised in Table 2.10. Many of the papers mine apps from Google Play, and compare the resultant predicted size with the actual size reported in the store and/or LOC (number of Lines Of Code) of the apps.

The scale of size and effort prediction studies is relatively small but, since the field has witnessed strong growth in 2015, it seems likely that the scale of studies will grow in the future.

In 2011 Sethumadhavan [252] discussed the application of Function Point Analysis (FPA) to Android applications, pointing out that compared with traditional desktop applications, mobile apps contain limited functionality, and often functionality is merely a wrapper to system functionality. Preuss [227, 228] then showed how FPA can be used for the estimation of the cost of a mobile app, using the approach on a case study Android application. In 2014 van Heeringen and van Gorp [272] discussed how to use the `COSMIC` method [58] to measure the functional size of apps. Abdullah et al. [1] discussed using the `COSMIC` method to estimate game apps, using an intermediate representation of required assets and functionality in the `Unity3D` game engine.

In 2015 D'Avanzo et al. [68] applied the `COSMIC` approach to 8 Google Play apps, and applied linear regression to the functional point size in order to estimate the code size. By applying leave-one-out cross validation, the authors showed that the approach can accurately estimate code size based on functionality alone, once trained. Francese et al. [80] used linear regression to estimate the development effort needed, and the number of GUI components, based on requirements alone. The authors found, from a study on 23 Android applications, that the estimates were accurate when trained on source code metrics such as classes, files and LOC. Ferrucci et al. [77] applied the `COSMIC` approach to 13 Android applications, showing that functional size is strongly correlated with app size, and that it can be used to accurately estimate the bytecode size of the app. Ferrucci et al. [78] later compared the related approaches by D'Avanzo et al. [68] and van Heeringen and van Gorp [272] on their dataset of 13 Android apps. They found that both functional size results were correlated with multiple app size measures, but that the approach presented by D'Avanzo et al. [68] was more accurate.

2.13 Checklist for Future App Store Analysis Authors

This survey has reported on the general content of studies, as well as the scale of apps used, and the store used. In future surveys it may be possible to synthesise more information from future literature. Such richer analysis, facilitated by richer data reporting, could lead to new insights and directions in the field of App Store Analysis.

To help facilitate this, we present the following checklist as recommendations for data to include in future studies, to help facilitate future studies such as SLRs:

App Stores used to gather collections of apps.

Total number of apps used in the study.

Breakdown of free / paid apps used in the study, including information regarding in-app-purchases where possible.

Categories used, with breakdown of app counts in each category.

Indication of whether API usage was extracted from the studied apps to facilitate the study.

Indication of whether code was needed from apps to facilitate the study.

Indication of whether open source apps were used exclusively for all of part of the study.

Total number of reviews used, if any.

Description of ratings and user feedback categories, including trends and response ranges.

Details of statical analysis techniques that were used in the study.

2.14 Threats to Validity

Internal validity: Our internal validity may be affected by missing relevant papers that constitute "app store analysis", or by using inclusion criteria that exclude relevant literature. To help mitigate this risk, we include influential and related literature that does not meet the inclusion criteria in Appendix A, and take this literature into account when making our conclusions.

External validity: Our external validity may be affected by the uniqueness of current mobile app store data. Conclusions drawn about the research presented in this chapter may not generalise well to other software systems, or to future app stores. To mitigate this risk, we make our definitions for app store, technical and non-technical attributes as general as possible, to be inclusive in order that conclusions may extend beyond the current versions of mobile apps.

Construct validity: Our construct validity may be affected by our carrying out the search queries and snowballing we describe. It is possible that papers were missed through this procedure. To mitigate this risk, we were as systematic as possible; while not able to carry out a "systematic literature review" due to there not yet being a well defined body of "app store analysis" literature, we repeated search queries on as many literature repositories as possible, and performed snowballing extensively.

Conclusion validity: Our conclusion validity may be affected by our understanding of the work presented, which incorporates a wide range of software engineering disciplines, and by missing relevant work. We contacted the authors we cite in order to ensure we has not missed relevant work, or mis-quoted their research.

2.15 Conclusions

We have surveyed the published literature in App Store Analysis for software engineering, and identified the key sub-fields of App Store Analysis to date: "API analysis", "feature analysis", "release engineering", "review analysis", "security analysis", "store ecosystem comparison", and "size and effort prediction". Newer sub-fields such as "release engineering" and "size and effort prediction" have shown strong growth in 2015, suggesting that they might eventually overtake other smaller sub-fields such as "store ecosystem".

The scale of app samples used in studies has increased: in 2015 the number of studies using between 10,000 and 100,000 apps was approximately three times that of 2014. We have observed the emergence of new areas of App Store Analysis, and the progression from conceptual ideas to practical empirical studies that apply and refine them.

Many aspects of app data have been used as features in empirical app store analysis, in the "feature analysis" subsection. One of the potential feature sources is app descriptions, yet research to date has looked at n-gram mining methods, and topic extraction has been used only for summarisation. We look at the potential solution of feature extraction using topic modelling in Chapter 4.

Through our own app store mining, we have noted that app store data availability is incomplete, and this is true for review data as well. This creates an inherent sampling bias when mining app and review data for analysis. The effect of this bias on results is explored in Chapter 5.

The literature has shown in increase in the field of release engineering, and has looked at the content of releases relating to ad updates. However, no literature to date has looked at the aspects of releases which might relate more directly to success, such as features. This problem is explored in Chapter 7.

In keeping with the checklist proposed in Section 2.13, we give details of the app stores used, as well as app and review quantity, and metrics used.

Chapter 3

Methodology

This chapter describes the methods, tools and data used throughout this thesis.

3.1 Statistical Analysis

Experiments are carried out with importance placed on achieving results that are statistically significant. Due to the large number of apps available for analysis, experimental results are not skewed by use of small datasets.

For correlation analysis, Spearman's Rank correlation is used. This results in a rho value and a p-value. The rho value indicates the strength of the correlations, where 1.0 shows that the compared datasets are entirely linearly dependent (as set a increases, set b increases by a proportional amount), -1.0 shows that the compared datasets are negatively linearly dependent (as set a increases, set b decreases by a proportional amount), and 0.0 shows there is absolutely no correlation whatsoever. The p-value indicates the chance of observing the rho value given that there is, in fact, no correlation. Interesting correlations therefore have high absolute rho values and low p-values; if either the rho value is close to zero, or the p-value is much higher than zero, then there is little evidence for any correlation.

We compare distributions using a two-tailed unpaired non-parametric Wilcoxon test [295], that tests against the Null-hypothesis that the result sets are sampled from the same distribution. We also compare result sets using Vargha and Delaney's $\hat{A}_{12}$ effect size comparison test [273], which results in a value between 0 and 1, that tells us the likelihood that one measure will yield a greater value than the other. The p value is the probability that one would observe the difference in median values, given that there is, in fact, no difference in the two distributions from which the releases are drawn. It is a conditional probability, usually used to reject the Null hypothesis (that there is no difference).

3.2 Data

Data from app stores is readily available through their online web pages. In recent years there has been a transition from static to dynamic content, where data on web pages is loaded on demand and usually as a result of the user clicking a button. This development has made automated data mining more difficult, but is solved through the use of frameworks that load an in-memory browser, such as `spynner` [230]. These frameworks are able to load a dynamic page and proceed to click on buttons as a user would in order to load the data. There are still limitations on how much content can be dynamically loaded without introducing overheads on the system, i.e. available memory to store the in-memory web pages.

Highly successful stores such as Google Play and the Apple App Store make compelling arguments to be focal to future studies, as do emerging stores such as Windows Phone Store. To support the studies in this thesis, snapshots of apps were mined from Blackberry World App Store, Google Play and Windows Phone Store. Due to the time taken to mine Blackberry, and larger snapshots from Windows, these stores were mined fortnightly, whilst Google and smaller Windows snapshots were mined on a weekly basis. Table 3.1 shows details of the data mined up to August 1, 2016.

Data mining was carried out using a combination of a dynamic browsing framework to fetch dynamic content (`spynner` [230]), as was used in the collection script by Harman et al. [109], and a more traditional web page fetching library to fetch static content (`urllib2` [229]). Data availability is limited, and so the amount of data collected varies per store. The app collection process is described in Section 3.3. Blackberry app reviews have also been mined as part of the study in Chapter 5 [183], and more detail on this process can be found in Section 3.3.1.

3.3 Mining Process

We use a process for mining apps that is based on the 4-step approach used by Harman et al. [109], shown in Figure 3.1:

Data Mining Phase 1 (Extraction of App List): We use the `spynner` [230] framework to continually load more apps to the 'most popular' list, but available memory becomes a limiting factor as the page size grows rapidly, when a large

Table 3.1: **App collection summary** as of August 1, 2016

Store	**Subgroup**	**Start**	**Schedule**	**Latest**	**Snapshots**	**Median Apps**
Blackberry	Most popular	Feb'14	Fortnightly	Oct'15	45	23,162
Google	Most popular overall	Feb'14	Weekly	Aug'16	128	1,080
Google	Most popular in (sub)categories	Nov'14	Weekly	Aug'16	83	17,079
Google	All previously collected	Feb'15	Weekly	Aug'16	83	60,008
Windows	Most popular overall	Mar'14	Weekly	Aug'16	123	1,974
Windows	Most popular in (sub)categories	Dec'14	Fortnightly	Aug'16	43	22,783

number of apps are available. On a machine with 4GB of RAM, the script is able to capture a list of between 20k and 24k apps before the framework runs of out memory.

Due to its loading of dynamic content, `spynner` is susceptible to memory limitations and delays, particularly if the page it is loading is large. This, combined with the high app numbers present, limits our ability to mine all available apps from the Blackberry store. It is unknown how to get around this problem to effectively mine the entire store due to its size.

In all cases there are limitations on what can be mined, which is related to what is contained on the pages, how pages are loaded, and available tools. Blackberry allows all apps to be loaded through a dynamic ordered list, through which we load up to 24,000 app links, and mine data from those pages. Google allows the top 540 paid and 540 free apps to be loaded, all of which are mined. Windows allows up to the top 1000 paid and 1000 free apps to be loaded, all of which are mined.

Data Mining Phase 2 (Raw App and Review Data Download): We visit each app in the list extracted by Phase 1, and download the `HTML` page which holds the app metadata. For Blackberry and Windows pages, the `spynner` framework is used, and for Google `urllib` [229] is used.

Data Mining Phase 3 (Parsing): We parse the raw data for a set of unique searchable signatures, each of which specifies an attribute of interest. These patterns enable me to capture app information such as price, rating, category, description, while other information such as the rank of downloads and the identifier were

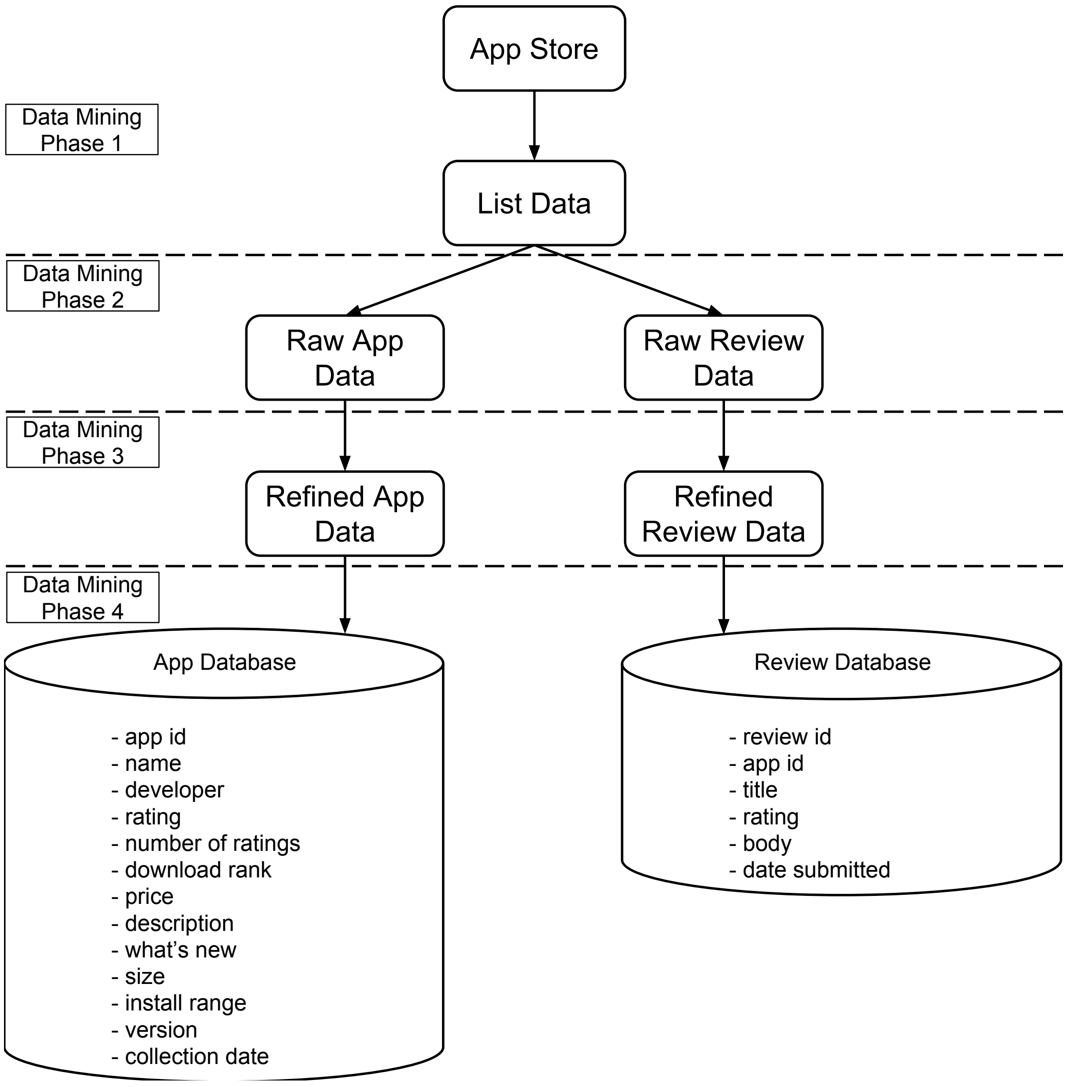


Figure 3.1: **App and Review mining process** showing the 4-phase approach described in Section 3.3

established in Phase 1. For example, the unique searchable signature for Blackberry rating is `awwsProductDetailsContentItemRating`.

Data Mining Phase 4 (Database): We store the parsed app data in an app database. The approach is applicable to any app store, with sufficient changes to the sections parsing the web pages. However in other app stores, the initial list popularity information from Phase 1 would need to be adapted, as other stores do not provide a global popularity ranking, but instead separate free and paid, or by category.

Table 3.2: **App review data summary**

Store	Snapshot	Median reviews	Total apps	Total reviews
Blackberry	140205	19	13,171	3,036,601
Blackberry	140306	2,425	489	1,329,938
Blackberry	140903	10	16,301	3,202,398

3.3.1 User Reviews

Review data can be loaded from Blackberry World App Store via dynamic lists. The complete dataset is available, and it is for this reason that it is used in the study in Chapter 5. An overview and discussion of the availability of reviews from other stores can also be found in Chapter 5.

Review data is mined from Blackberry by repeatedly requesting more reviews via `spynner`. Reviews are loaded in sets of up to 14, depending on their length, until either the memory is exceeded and the framework crashes, or the soft limit is reached in mining code. To prevent crashes, we set this limit to load 4,000 reviews per app, which consumes between 2GB and 3GB of system memory. An overview of mined review data from Blackberry World App Store is presented in Table 3.2.

3.3.2 Persistent List Collection

Since February 2015, we have aggregated a list of all unique links captured from the Google Play store, for weekly collection of all valid apps. Figure 3.2 shows the number of total unique links, and valid working links mined weekly between February 2015 and August 2016. The valid links mined is always lower than the total links, due to apps dropping out of the store. This may be due to a number of factors:

1) Google removes apps when they are flagged as inappropriate or infringe on copyright.

2) Authors remove apps for a number of reasons not always clear. See for instance the case of `Flappy Bird`, an app that was reported to have been removed from the store by the author because it garnered too much media attention [23].

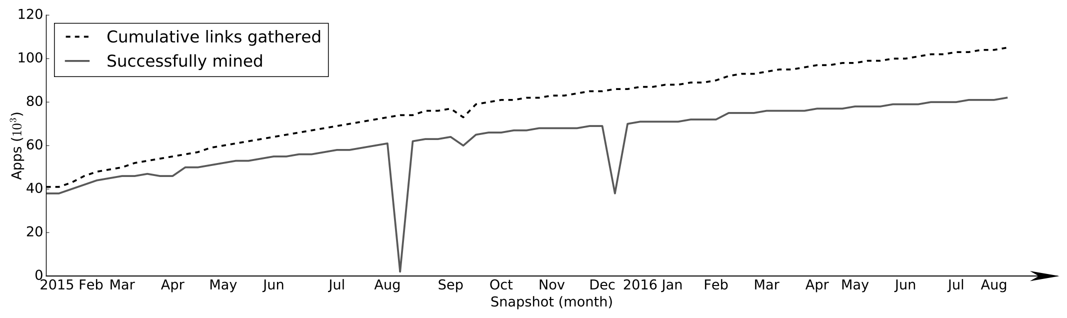


Figure 3.2: **Persistent Google data summary** from February 2015 to August 2016

There are two noticeable dips in Figure 3.2, that were caused by changes in the app store page schema. Since these changes were unexpected, it took a week to adapt the mining scripts to mine complete snapshots in each case.

3.3.3 Time Series Datasets

The following datasets are used in the time series studies on app releases in Sections 7.3 and 7.4:

Strongly popular Google: Mined between July 2014 and July 2015, used by Section 7.3 and the published papers [182, 184]. This dataset consists of 307 apps.

Strongly popular Windows: Mined between July 2014 and July 2015, used by Section 7.3 and the published papers [182, 184]. This dataset consists of 726 apps.

Weakly popular Google: Mined between February 2015 and February 2016, used by Sections 7.4 and 7.5 and the published paper [185]. This dataset consists of 38,858 apps.

3.4 Metrics

The metrics used throughout this thesis are defined in Table 3.3. Metrics are extracted from mined app store data on a per-app basis. The success metrics can be used to monitor and compare how well apps are performing relative to other apps in their store or category.

3.5 Text Preprocessing

Throughout this thesis, the mined qualitative data including app description text and review body text, is treated to the pre-processing steps shown in Figure 3.3. These steps are taken to reduce the impact of potentially unwanted (stop)words,

Table 3.3: **App metrics** summary

Metric	**Description**
(P)rice	The amount a user pays for the app in local currency (in this case GBP) in order to download it. This value does not take into account in-app-purchases and subscription fees, thus it is the 'up front' price of the app.
(L)ength of description	The length in characters of the description, after first processing the text as detailed in Section 3.5, and further removing whitespace in order to count text characters only.
Success Metric	**Description**
(R)ating	The average of user ratings made of the app since its release.
(D)ownload rank	Indicates the app's popularity. The specifics of the calculation of download rank vary between App Stores and are not released to the public, but one might reasonably assume the rank approximates popularity, at least on an ordinal scale. The rank is in descending order, that is it increases as the popularity of the app decreases: from a developer's perspective, the lower the download rank the better.
(N)umber of ratings	The total number of ratings that the app has received. A rating is the numerical component of a review, where a user selects i.e. a number from 1 to 5.
(NW) Number of ratings per week	The number of ratings that the app has received since the previous snapshot, which is taken a week earlier.

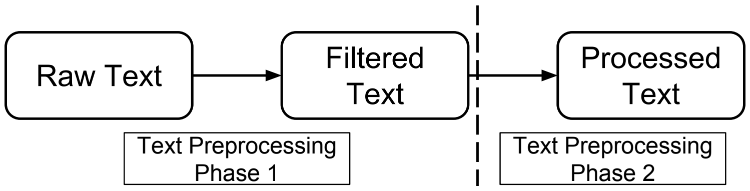


Figure 3.3: **Text pre-processing process** discussed in Section 3.5

and combine multiple forms of the same word. However, there is a risk that this can reduce the classification performance [174].

Text Preprocessing Phase 1 (Filtering): Text is filtered for punctuation, URLs, 1-letter words and the list of words in the English language `stopwords` set in the Python NLTK data package[1]. It is then cast to lower case.

Text Preprocessing Phase 2 (Lemmatisation): Each word is transformed into its 'lemma form' using the Python NLTK `WordNetLemmatizer`. This process homogenises singular/plural, gerund endings and other non-germane grammatical details, yet words retain a readable form i.e. faster $\rightarrow$ fast. The process is similar to stemming but produces more readable results, which helps with evaluation of the resultant topics.

3.6 Topic Modelling

Topic modelling is used throughout the literature in app analysis for categorisation [124, 300], summarisation [50, 82], data extraction [83, 104], and API threat detection [95]. It has also been used recently in in the MSR community for improvements in error reporting [40, 148], among many other applications. This section describes the topic modelling process used in the study, and defines terms used later on.

Topic modelling algorithms such as latent Dirichlet allocation (LDA) [32] take as input a set of unstructured textual documents, a fixed number of underlying topics, K, an α Dirichlet prior and a β Dirichlet prior distribution (in this case assumed symmetrical). We compute the topics using the Mallet framework [8], which performs parallelised Gibbs sampling [209], and delivers as output the distribution of word likelihoods for each topic, ϕ, and the distribution of topic likelihoods for each document, θ.

[1]`nltk.corpus.stopwords.words('english')`

Table 3.4: **Topic modelling parameter choices**

Parameter	**Setting**
α	50 (default)
β	0.01 (default)
Gibbs sampling iterations	500

ϕ and θ are probabilistic multinomial distributions sampled from the Dirichlet distribution, and each ϕ_k and θ_m sum to 1. In addition, LDA uses the bag of words model, that ignores the ordering and placement of text within a document, and treats each word as a separate random sample from a topic.

Topic Modelling Phase 1: The text from each document (app description or user review) is processed as described in Section 3.5. This text is fed into a modified version of Latent Dirichlet Allocation (LDA) [32], which enhances performance through parallelisation [209]. This is applied using Mallet [8], a Java tool for applying machine learning and language processing algorithms.

Topic Modelling Phase 2: The output of LDA includes a list of topic likelihoods for each document (app description or user review). From this, we compute a list for each topic, of the documents for which the topic-document likelihood exceeds a threshold.

Mallet [8] is run on the processed text with the parameters specified in Table 3.4. In the case of the Dirichlet priors, which affect the sparsity of the resulting per-topic word distributions and per-document topic distributions, we opt to go with the defaults as these generally work well in the context of natural language [217], and optimising the topic results are out of the scope of this study. The number of sampling iterations was selected as 500, since our results converged early and showed very little change (measured by incremental reductions in perplexity) after 100 iterations.

As with the number of topics, we trialled multiple topic threshold values ranging from 0.1 to 0.005, and selected 0.02. Using this setting, the resultant mean number of topics said to have generated an app came to a manageable figure of 5.62, and a manual inspection of 10 sampled apps showed an appropriate level of summarisation of the app description.

3.7 TF.IDF

TF.IDF [179] finds ‘top terms’ in release text: each term in each document is given a score of TF (Term Frequency) multiplied by the IDF (Inverse Document Frequency). The IDF is equal to the log of the size of the corpus divided by the number of documents in which the word occurs.

Chapter 4

App Feature Extraction using Topic Models

App Stores provide a wealth of technical and non-technical information on app pages. This non-technical information may be used in conjunction with technical data to provide insights on software, that were previously not possible. In particular, it is possible to extract claimed software features, a method that has been previously studied using an n-gram feature extraction approach [109]. This study explores the option of using topic modelling to extract topics, which can then be treated in a way analogous to features, that offers greater flexibility due to the use of configurable settings such as number of topics and topic probability thresholds. We perform empirical analysis on these topics using app ratings, download ranks and prices, and find that the approach produces very similar results to the feature based approach on the same dataset.

4.1 Introduction

Non-technical app store data, in the form of app descriptions, has been combined with technical information, in order to analyse empirical app store data. This was done by first extracting claimed software features using a greedy n-gram algorithm [79]. The greedy n-gram algorithm works by singling out the list of claimed features (as sentences) that are present in a Blackberry app description, and extracting bi- and tri-grams from them. These n-grams (“featurelets”) are then clustered via a greedy algorithm, and the resultant clusters (“features”) with more than one featurelet are used.

An alternative approach is to use topic modelling [32], an unrelated, but widely studied language processing technique. Using this technique we can extract topics, and apply the same experiments to them as we would to features. Topic modelling is described in more detail in Section 3.6.

We replicate the study by Harman et al. [109] using a topic modelling approach in place of their n-gram feature extraction method, showing that topics are analogous to features. Empirical analysis using these alternative methods yields the same results. Furthermore, we apply a sliding window approach based on the number of reviews an app has received, in order to identify and filter out the bias caused by zero-rated apps in the dataset, and extract more 'signal'.

4.2 Findings

The findings of this study are as follows:

i) Topics are analogous to features: we show that topics are analogous to features, and can be used to analyse claimed features mined from app descriptions, and other free-text content.

ii) Zero rated apps should be treated separately from other apps: we identify the bias that zero-rated apps add to observed trends, by performing correlation analysis after filtering for the number of reviews an app has, with a sliding window.

4.3 Definitions

Feature: a set of terms that describe something about the app. Features are extracted from a developer's list in their app's description, typically functionality or a property.

Topic: a probabilistic distribution over a finite set of terms. Terms that co-occur in multiple documents in a corpus (and are therefore usually related) have high probabilities together in the same topic, so by applying a threshold probability value we can obtain a set of related terms, or a theme.

4.4 Data

The same data is used for this experiment as for the study by Harman et al. [109]. This dataset consists of 32,108 paid apps and 9,999 free apps, mined in September 2011.

4.5 Research Questions

This section explains the research questions explored in order to determine the viability of a topic-modelling based feature extraction approach.

RQ1: How much do correlations from the extracted topics differ from those of features?

We explore correlations between Price, Rating and Download rank (an inverse measure of popularity), and compare the results with those for the apps and features published previously [109]. We apply the Wilcoxon Rank-Sum test [295] to investigate the similarity of results between each of features, apps and topic results. Specifically, we test against the Null hypothesis that the mean rank of a pair of sets (of P, R or D between features and topics) is the same, i.e. there is no difference between the two methods. Additionally, we qualitatively inspect a sample of 5 of the extracted topics to verify that they have meaning.

RQ2: What is the chance of producing a similar topic correlation in each category at random?

In the same way as with the the feature study [109], the metric values for a topic are computed as the mean over the apps that share the topic. This will show whether 'pseudo topics', random sets of apps, can replicate the correlations found. To answer this question we construct pseudo topics by randomly sampling sets of topics for each app.

4.6 Application of LDA

LDA, and the process used to train the model, is described in detail in Section 3.6. We perform the following steps to obtain results:

1) LDA is trained on the set of app descriptions as described in Section 3.6; a mixture of topics generates each app description.

2) Metrics for topic price, rating and downloads rank are computed from the mean of the set of apps generated by a topic.

The settings specified in Table 3.4 were used. After trialling multiple values of K (number of topics) ranging from 10 to 3,000, we select 100 as a balance between representative topics and granularity based on the size of the dataset: 10 topics were

Table 4.1: **RQ1: Significance test results**
P,R: Price / Rating correlation
P,D: Price / Download rank correlation
R,D: Rating / Download rank correlation

	Wilcoxon Rank-Sum p-values		
	P,R	**P,D**	**R,D**
Features / Topics	0.2796	0.3351	0.2304
Apps / Topics	0.2931	0.2201	0.1802

too general, whilst over 500 topics resulted in very specific topics, and an increasing quantity of low-quality topics [194].

4.7 Results

In the following results, P,R denotes Price/Rating correlation, P,D denotes Price/Downloads correlation and R,D denotes Rating/Downloads correlation.

RQ1: How much do correlations from the extracted topics differ from those of features?

The topic-correlation results can be found in Table 4.2. Topic-correlation results on 32,108 non-free apps from the Blackberry World App Store showed the same correlations as the feature-correlation method. These result show very similar correlations to those in the original study, that performed feature-based correlation analysis. This finding confirms the rather surprising finding that there is no correlation between either price and rating, or price and download rank of apps. As expected, there is a strong correlation between rating and download rank (popularity).

We qualitatively inspect 5 random topics in order to verify they have meaning. The top 5 terms from each of these topics is given below:

```
post package dhl ups tracking
theme icon scroll dock included
email send one use application
natural disaster earthquake tsunami always
image photo effect color filter
```

Table 4.2: **RQ1: Topic-correlation results table** showing Spearman's rank correlation coefficients for results with $p < 0.05$
P,R: Price / Rating correlation
P,D: Price / Download rank correlation
R,D: Rating / Download rank correlation

Name of Category	Number of Non-free Apps	Number of topics	Topic Correlation			Feature Correlation			App Correlation		
			P,R	P,D	R,D	P,R	P,D	R,D	P,R	P,D	R,D
Reference & eBooks	11,584	100	0.83	0.84	0.89	0.06	-0.29	0.77	0.09	-0.13	0.32
Themes	10,936	100	0.41	0.37	0.97	0.15	0.02	0.90	0.16	-0.15	0.81
Games	2,604	100	-0.46	0.14	0.60	0.25	0.39	0.83	-0.10	0.01	0.76
Utilities	1,362	100	-0.13	-0.10	0.93	-0.06	-0.07	0.92	-0.10	0.03	0.77
Entertainment	908	100	-0.18	-0.24	0.83	-0.31	0.06	0.47	-0.17	0.21	0.81
Travel	764	100	0.36	0.26	0.90	0.38	0.29	0.95	0.04	0.02	0.75
Health & Wellness	626	100	-0.63	-0.68	0.92	-0.12	-0.11	0.88	-0.28	0.26	0.85
Education	576	100	-0.05	-0.08	0.76	-0.28	-0.45	0.82	-0.10	0.05	0.83
Productivity	503	100	-0.25	-0.20	0.87	0.03	0.08	0.88	0.01	-0.08	0.73
Music & Audio	499	100	0.52	0.60	0.92	-0.31	-0.28	0.76	0.42	-0.33	0.76
Photo & Video	393	100	0.31	0.36	0.95	-0.33	-0.25	0.91	0.02	-0.06	0.82
Business	350	100	0.11	0.07	0.88	0.20	0.17	0.76	0.02	-0.03	0.83
Maps & Navigation	245	100	0.02	0.07	0.81	0.20	0.06	0.87	-0.06	-0.01	0.78
Sports & Recreation	239	100	0.13	0.17	0.81	0.01	0.37	0.76	-0.10	0.02	0.77
Finance	193	100	0.37	0.37	0.86	0.19	0.31	0.40	0.33	-0.43	0.81
IM & Social Networking	150	99	0.02	0.15	0.77	-0.52	-0.42	0.61	-0.21	-0.02	0.63
News	73	99	0.08	0.07	0.74	-0.25	-0.32	0.82	0.07	-0.16	0.79
Weather	58	99	0.18	0.19	0.68	-0.38	-0.26	0.38	0.07	-0.12	0.54
Shopping	45	88	-0.04	-0.01	0.76	0.60	-0.36	0.26	0.26	-0.21	0.67
All Categories	32,108	100	-0.22	-0.13	0.97	0.07	-0.09	0.89	0.10	-0.12	0.79

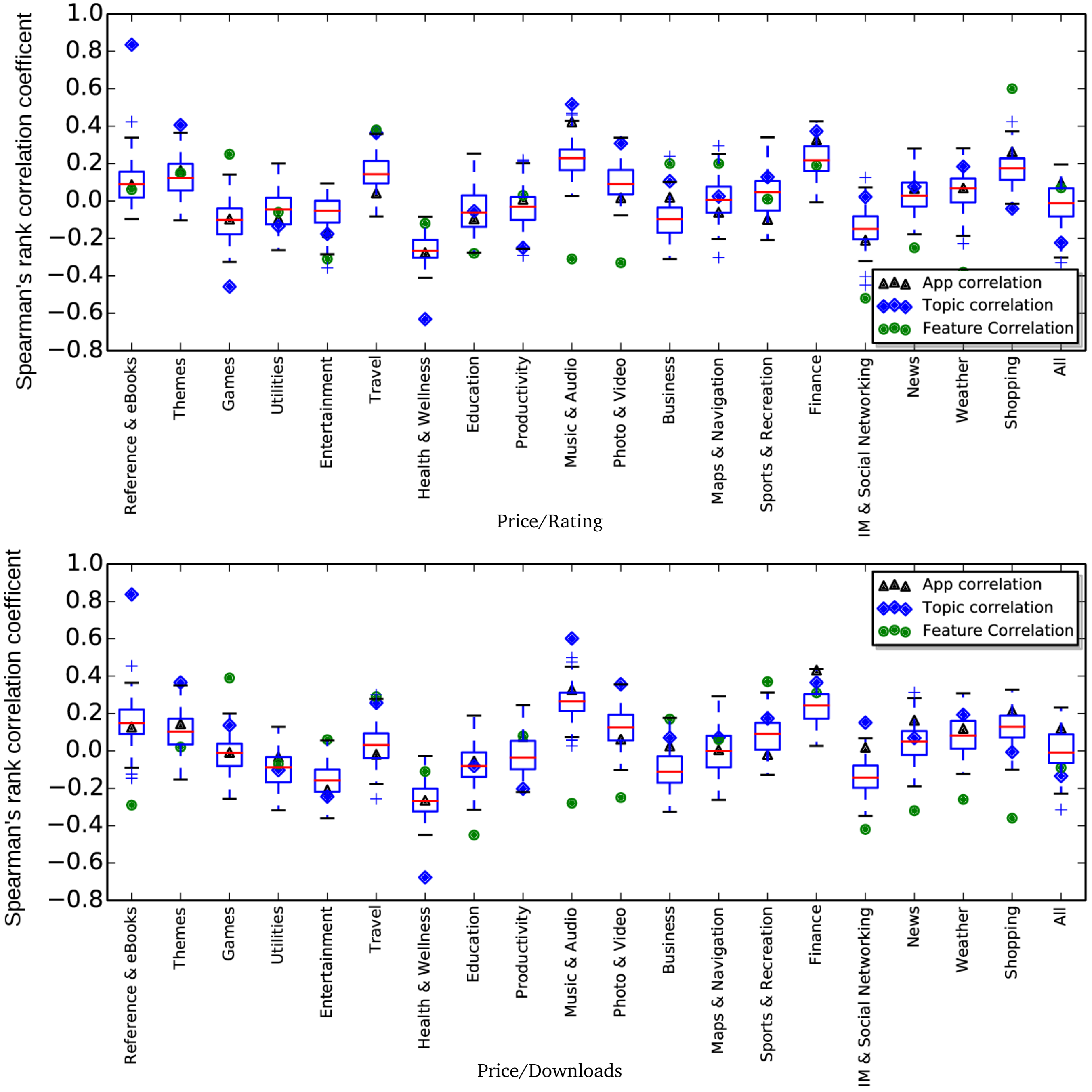


Figure 4.1: **RQ2: Comparison of Spearman's rank correlation coefficients** including box-and-whisker plots of pseudo topics

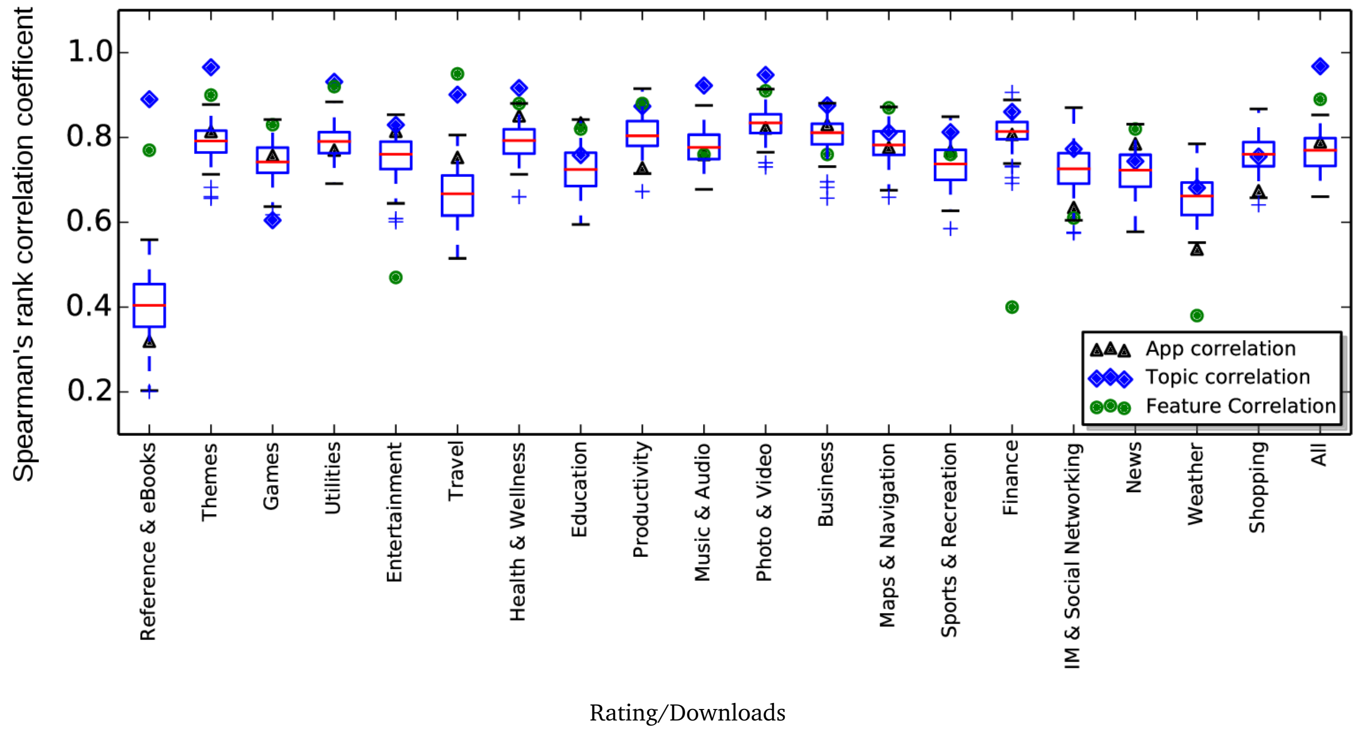


Figure 4.1: **RQ2: Comparison of Spearman’s rank correlation coefficients** including box-and-whisker plots of pseudo topics

By the qualitative judgement of the author of this thesis, these topics have meaning, and appear similar to the n-gram features stated in the earlier study [79].

Results of the Wilcoxon test are shown in Table 4.1. Each of the results comparing the populations of feature-R,D to topic-R,D show a high p-value, indicating that the Null-hypothesis that mean ranks are the same cannot be rejected.

RQ2: What is the chance of producing a similar topic correlation in each category at random?

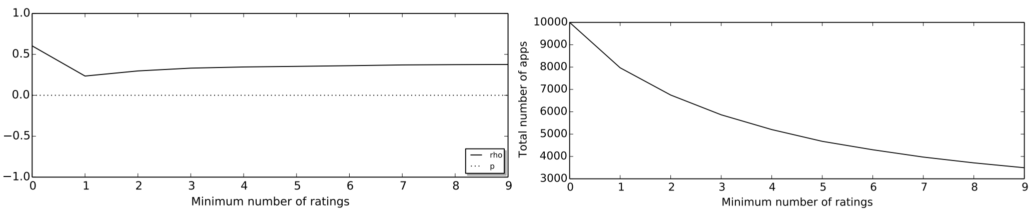
We sampled 6 random 'pseudo topics' for each document out of 100 potential pseudo topics, as 5.62 is the mean number of 'top topics' above the threshold value in the results from RQ1. Results for P,R, P,D and R,D correlations compared between topics, features, apps and pseudo topics are shown in Figure 4.1. The box-and-whisker plots show the Spearman's rank correlation coefficients of pseudo topic distributions for each category, and app, topic and feature results are plotted as triangles, diamonds and circles, respectively. Plus symbols in the plot show outliers from the pseudo topic distributions.

These results show that pseudo topics often produce similar correlations to app level correlation analysis between price, rating and download rank. However, topic- and feature-based correlation analysis identifies stronger correlation results in almost all categories.

The topic-correlation method is potentially faster than the n-gram approach as the topic training algorithm is heavily optimised, due to its widespread use and improvements such as parallelisation. However, the artifacts produced by this method cannot be used in the same way as those produced by the feature-correlation method: the topics generated are influenced by a random element and are subject to change during the generative sampling process, so if more data is added, the topics will change; conversely, the identified features are trigrams which will not change with new data, and as a result their presence or lack thereof can be tracked and studied.

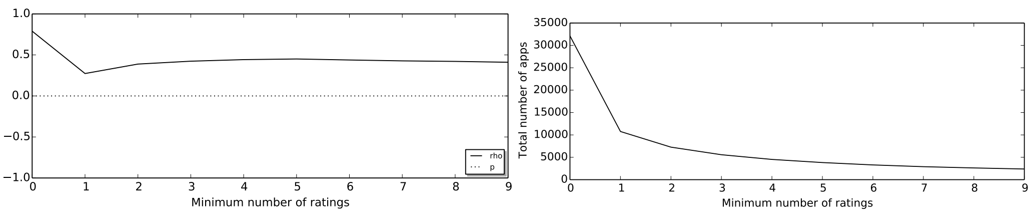
4.8 The Problem of Zero Rated Apps

The results in Table 4.2 show a decline in the strength of the ranked Rating/rank of Downloads correlation, as observed in the original study [109]. We explored the



Rating/Download correlation App counts

Figure 4.2: **Sliding window correlation graphs for free apps**



Rating/Download correlation App counts

Figure 4.3: **Sliding window correlation graphs for paid apps**

issue and found that a large number of zero rated apps are present in the dataset, as shown by Figures 4.2b and 4.3b. We investigate further, using a sliding window mechanic where we increment the minimum number of ratings as a filter, and compute the results each time. Results for individual categories can be found in Appendix B.

The results in Figures 4.2 and 4.3 show that there is a significant correlation between rating and download rank when all reviews are considered. However, as we begin to be more selective of apps, choosing those that have a minimum number of 1 review, the correlation drops out and is no longer significant (the 'dip' in Figures 4.2 and 4.3). Interestingly, as we raise the minimum number of reviews further, the correlation increases and almost reaches the peak value again (with $minimumN = 0$).

This 'flattened tick shape' can be seen on almost all rating/download rank correlation graphs. The exceptions are the categories 'Reference and eBooks' and 'Themes', which do not exhibit the same relationship due to the large number of apps containing content and not functionality, i.e. a book or a graphical theme, but no application. This pattern is consistent among free and paid app correlation graphs between rating and download rank. The same cannot be said for paid app graphs for price/rating and price/download rank correlations.

The causes for the 'flattened tick shape' are the large numbers of specific cases. For $minimumN = 0$, a large number of apps are considered that have no rating, leading to a strong correlation as apps with no rating have low download rank. In the case of $minimumN = 1$, there is more evidence on which to perform correlation analysis, however in many categories the large number of apps with 1 rating can negatively affect the correlation coefficient. For $minimumN > 1$, there is sufficient evidence to perform correlation analysis with confidence that the results are representative of 'normal' apps: ones that have been used and reviewed at least a handful of times.

The strong R,D correlation results show that apps which are downloaded more are generally considered to be better, in most cases. Conversely, the insignificant correlations between price/rating, and between price/download rank, tell us that in most cases with non-free apps, price does not bear a relation to the likelihood of users downloading apps, or to their satisfaction with apps after using them.

4.9 Threats to Validity

Internal validity: Our internal validity may be affected by the use of the Blackberry dataset. However, the generalised process of feature extraction using topic modelling extends to any free text app descriptions, and it is the aim that the method is more generic than n-gram feature extraction.

External validity: Our external validity may be affected by the use of the most popular apps in the store at the time of mining, and the use of Blackberry data. Findings, therefore, may not extend to other stores or samples. However, the methods we use can be applied to other samples as future work.

Construct validity: Our construct validity may be affected by the replication of the same experiments as n-gram features were used for. Our finding that both methods achieve similar results does not necessarily mean that the methods achieve similar features. We mitigate this threat by inspecting several topics and features to verify that similar features are obtained.

Conclusion validity: Our conclusion validity may be affected by our qualitative human assessment of the features and topics extracted. However, our methodology can be applied more widely to assess extract features or topics from any dataset.

4.10 Conclusions

We have shown that topic modelling can be used successfully as an alternative to the feature n-gram detection approach used in the paper by Harman et al. [109]. This study has shed extra light on the observed trends within the Blackberry data, and led to a more in depth study of the relationships in the data present once it is filtered according to number of reviews. The results gained from the topic modelling approach were not different in nature to the feature-based approach.

Due to the configurable nature of topic models, they could be used where tuning for better representational performance is desired. The configurable nature could potentially lead to difficulties in interpretation, however: where topics are too general, it may be difficult to extract meaning from them. Through careful tuning, however, they can be as fine grained as n-gram features. Features remain a better option to use for applications such as tracking migration: while topics will change over time, as corpus of app descriptions changes, features will not. But topics can be better applied to very large corpora, enabling the user to configure the number of topics to a manageable amount. The topic modelling approach offers the ability to link developer descriptions with other textual data, such as user reviews; this technique is later used in Chapter 5.

Future work might use the topic modelling approach as either an alternative to the feature-based approach, or as a additional method for validation. Other topic models or inference methods for training the model might also be considered, such as the weighted topic model [194] or externally weighted model [189].

Chapter 5

The App Sampling Problem for App Store Mining

This work was published in MSR 2015 Proceedings [183]. The first author's contribution to this paper was to formulate the idea, implement and execute the experimentation and collect the results and analyse them; other authors of the paper contributed to research question formulation, result analysis and narrative write up.

Many papers on App Store Mining are susceptible to the App Sampling Problem, which exists when only a subset of apps are studied, resulting in potential sampling bias. We introduce the App Sampling Problem, and study its effects on sets of user review data. We investigate the effects of sampling bias, and techniques for its amelioration in App Store Mining and Analysis, where sampling bias is often unavoidable. We mine 106,891 requests from 2,729,103 user reviews and investigate the properties of apps and reviews from 3 different partitions: the sets with fully complete review data, partially complete review data, and no review data at all. We find that app metrics such as price, rating, and download rank are significantly different between the three completeness levels. We show that correlation analysis can find trends in the data that prevail across the partitions, offering one possible approach to App Store Analysis in the presence of sampling bias.

5.1 Introduction

Data availability from app stores is prone to change: between January and September 2014 we observed two changes to ranked app availability from the Windows store: a major change, increasing the difficulty of mining information by refusing automated `HTTP` requests; and a minor change, extending the number of apps available. Google and Apple stores enabled mining of a large number of ranked apps during 2012 [82, 119], but far fewer are currently available. At the time of mining in February 2014, we observed that the availability of app data was incomplete in Google, Windows and Apple, but not in Blackberry.

This study is concerned with reviews: user-submitted app evaluations that combine an ordinal rating and text body. Table 5.1 shows the availability of reviews at the time of writing, as well as a summarisation of the quantity of data. The availability of reviews presents a challenge to researchers working on App Store Review Mining and Analysis, because it affects generalisability. If it is only possible to collect sets of the most recent data, i.e. from Google Play and Windows Phone Store, then it is only possible to answer questions concerning this most recent data. For instance, the research question "*What is the type of feedback most talked about in app reviews?*" could not be answered using a data subset, but could be reasonably answered by collecting a random sample from all published reviews.

Work in App Store Review Mining and Analysis has analysed app review content and sentiment [104, 119, 126, 138, 214], extracted requirements [83, 124] and devices used [139], and produced summarisation tools [50, 82]. It is apparent, however, that the literature on App Store Review Mining and Analysis, with the exception of three related studies by Hoon et al. [119, 120, 274] and one by Fu et al. [82], uses only a subset of all reviews submitted (at the times of collection). We therefore question whether the data provided is sufficient to draw reasonable conclusions.

As a way of exploring the issue empirically, we assess the level of representation an app review subset can provide. For this, a full set of user reviews is needed, and so we study the Blackberry World App Store, where the full published history is available.

5.2 Contributions

The contributions of this study are as follows:

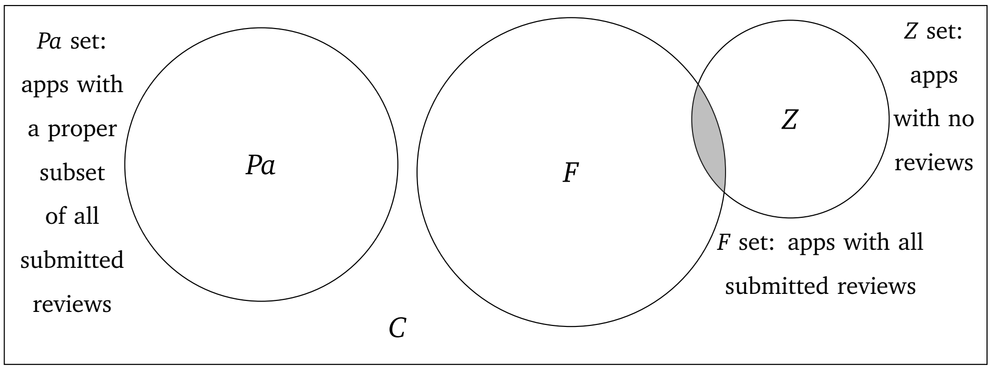
i) We survey the literature in the field of App Store Review Mining and Analysis, and identify the prevalent problem of partial datasets.

ii) We present empirical results that highlight the pitfalls of partial datasets and consequent sampling bias.

iii) We mine and analyse a dataset of apps and user requests, using manually vali-

Table 5.1: **Review availability** in February 2014[183]

Store	**Apps**	**Most reviews**	**Review availability**	**Method of access**
Apple	1,200,000	600,000	Last release	RSS feed
Blackberry	130,000	1,200,000	Full	Web
Google	1,300,000	22,000,000	2,400	Web
Windows	300,000	44,000	36	Web


Pa set: apps with a proper subset of all submitted reviews
Pa
F
Z
Z set: apps with no reviews
F set: apps with all submitted reviews
C

C set: all available apps at the time of mining

Figure 5.1: **Venn diagram showing the datasets we define**[183]

dated automatic extraction.
iv) We illustrate how the trends identified by correlation analysis are unaffected by this sampling bias, in our dataset.

5.3 Definitions

We define the following sets, shown in Figure 5.1, in order to describe different levels of data completeness. These sets refer to the data collected in a sample, at the time the sample is taken, and enable us to refer to their completeness.

Set unit: an app and it's *collected* reviews, in the dataset in question.

(*C*) Complete: The set of apps and all their available reviews at the time of mining. This set does not need to contain all of the apps in the store, but for all the apps it contains, it must also contain all of their available reviews.
(*Pa*) Partial: The set of apps and their reviews for which the dataset contains only a proper subset of the reviews submitted to the App Store at the time of mining.

(F) Full: The set of apps and their reviews for which the dataset contains all of the reviews submitted to the app store at the time of mining.

(Z) Zero: The set of apps that have no reviews.

(A) All: All mined data in a snapshot: ($A = Pa \cup F \cup Z$).

If the mined dataset contains all available reviews for all mined apps then $A = F = C$. If the mined dataset contains *half* of the reviews for all mined apps then $A = Pa \subset C$ and $F = \emptyset$. If the mined dataset contains all available reviews for *some* mined apps and only a subset of available reviews for other mined apps (such as the dataset used in this study) then $A = Pa \cup F \cup Z$ and $A \subset C$.

5.4 The App Sampling Problem

There has been a surge of work recently in the field of App Store Mining and Analysis, much of which is focussed on app reviews. 9 out of 13 past studies identified on App Store Review Mining and Analysis use *Pa* sets of data, drawn from the most recent reviews of the most popular apps at the time of sampling. A summary of the work to date on App Review Mining and Analysis is presented in Table 5.2, and a further discussion of related App Store research is presented in Section 5.11.

Following from the data accessibility summarised in Table 5.1 in the four App Stores, sets are *available* under the following conditions at the time of writing:

Apple App Store: F set available for apps with one release, or where only the most recent release has been reviewed; *Pa* set available for apps with multiple reviewed releases.

Blackberry World App Store: F set available for all apps.

Google Play: Data available for a maximum of 480 reviews under each star rating via web page (2,400 total) per app; hence F set available for apps with fewer than 481 reviews under each star rating, *Pa* set available otherwise.

Windows Phone Store: Data available for a maximum of 36 reviews per app; hence F set available for apps with less than 37 reviews, *Pa* set available otherwise.

Historically, greater access appears to have been available. Further data access is also available through 3rd party data collections, subject to their individual pricing plans, terms and conditions; such collections are out of the scope of this study as they are not freely available, and have not been used in the literature.

Table 5.2: **Summary of related work datasets** [183]
a: Apple App Store
b: Blackberry
g: Google Play or other Android stores
Pa: Partially complete review set
F: Fully complete review set
A: combined set of Pa, F and Zero-rated apps

Authors [Ref], Year	**Store**	**Apps**	**Reviews**	**Type**
Hoon et al. [120] 2012	A	17,330	8,701,198	*F*
Vasa et al. [274] 2012	A	17,330	8,701,198	*F*
Hoon et al. [119] 2013	A	17,330	8,701,198	*F*
Iacob et al. [124] 2013	G	161	3,279	*Pa*
Galvis Carreño et al. [83] 2013	G	3	327	*Pa*
Khalid [138] 2013	A	20	6,390	*Pa*
Fu et al. [82] 2013	G	171,493	13,286,706	*F*
Pagano et al. [214] 2013	A	1,100	1,100,000	*Pa*
Iacob et al. [126] 2013	G	161	3,279	*Pa*
Khalid [140] 2014	A	20	6,390	*Pa*
Chen et al. [50] 2014	G	4	241,656	*Pa*
Guzman et al. [104] 2014	A,G	7	32,210	*Pa*
Khalid et al. [139] 2014	G	99	144,689	*Pa*
This study	B	15,095	2,729,103	*A*

Availability of app store data is likely to become a pressing problem for research if we cannot find ways to mitigate the effects of “enforced” sampling bias: sampling in which sample size and content are put outside of experimental control. This may be for reasons unavoidable (app store data availability), pragmatic (memory or framework limitations on loading large dynamic web pages), or otherwise. We call this the *App Sampling Problem*.

App Sampling Problem: The problem of *enforced* sampling bias, as a result of incomplete dataset availability, or other mining limitations.

In this study we investigate the effects of the app sampling problem on an app and review dataset mined from the Blackberry app store.

5.5 Data

The data mining process described in Section 3.3 was used to mine app and review data. `spynner` was used to continually load more reviews from Blackberry pages for review mining. This has the same memory limitation as (Section 3.3) Phase 1, enabling us to capture between 4k and 4.5k reviews per app, before the framework runs out of memory. We therefore limit the number of captured reviews to 4k per app, and it is this limitation that causes a *Pa* dataset; the set of apps for which we have mined only a subset of the available reviews.

Spam is common in app reviews, and can occur for reasons of competition, i.e. to boost one’s own app, or to negate a competitor’s app. Some app stores have methods in place to help prevent spam, but these methods are not publicly available, and they can never be 100% effective. Jindal and Liu [130] found that online reviews that are identical to another review are almost certainly spam, and are also the largest portion of spam by type. We therefore filter the reviews for duplicates, including only one copy of each unique review. A unique review is defined by the rating, review body and author name. We do not use more sophisticated spam detection and filtering methods in this study because there is no prescribed standard, and spam detection is not the aim of this study.

Errors in app pages such as negative ratings, negative number of ratings and empty app IDs, led to the exclusion of a number of apps, since we judged that the

Table 5.3: **Blackberry dataset used**[183]

Set	Apps	Reviews
Pa	5,422	1,034,151
F	6,919	1,694,952
Z	2,754	0
A	15,095	2,729,103

anomalies *would* affect the results of our experiments. The final number of apps used in the study is therefore 15,095[1].

This full set of apps is the *A* set. As detailed in Table 5.3, it is split into *Pa*, *F* and *Z* subsets according to the definitions in Section 5.3. In the following experiments, *Pa*, *F* and *Z* contain distinct sets of apps: $Pa \neq F \neq Z$.

5.6 Research Questions

In this section we discuss the research questions that are asked in order to explore the app sampling problem.

RQ1: How are trends in apps affected by varying dataset completeness?

We explore how app-level trends differ, based on the completeness of their review sets. Since review sets are limited to 4,000 reviews as discussed in Section 3.3, their completeness (and therefore which set they appear in out of *Pa*, *F* and *Z*) in this case reflects how many reviews they have received at the time of mining.

RQ1.1: Do the subsets Pa, F and Z differ? The mined dataset is separated into the subsets of *Pa*, *F* and *Z*, as illustrated in Figure 5.1.

We compare the datasets using the metrics (P)rice, (R)ating, (D)ownload rank, (L)ength of description and (N)umber of ratings with a 2-tailed, unpaired Wilcoxon test [295]. The Wilcoxon test is appropriate because the metrics we are investigating are ordinal data, and therefore a non-parametric statistical test is needed, that makes few assumptions about the underlying data distribution. We make no assumption about which dataset median will be higher and so the test is 2-tailed; the test is unpaired because the datasets contain entirely different sets of apps. We

[1]Data from this study is available at `http://www0.cs.ucl.ac.uk/staff/W.Martin/projects/app_sampling_problem/`.

test against the Null hypothesis that the datasets in question are sampled from distributions of the same median, and a p-value of less than 0.05 indicates that there is sufficient evidence to reject this Null hypothesis with a maximum of 5% chance of a Type 1 error. Because multiple tests are performed, we apply Benjamini-Hochberg correction [24] to ensure that we retain only a 5% chance of a Type 1 error.

We investigate the effect size difference of the datasets using the Vargha-Delaney $\hat{A}_{12}$ metric [273]. Like the Wilcoxon test, the Vargha-Delaney $\hat{A}_{12}$ test makes few assumptions, and is suited to ordinal data such as this. It is also intuitive: for a given app metric m (from P, R, D, L or N) and the app sets (A, B), $\hat{A}_{12}$(A, B) is an estimate of the probability that the value of a randomly chosen value of m from A will be higher than that of a randomly chosen value of m from B.

RQ1.2: Are there trends within subsets Pa, F and Z? We compare the trends within datasets *Pa, F, Z* and *A* using Spearman's rank correlation [256]. This methods will be run on each pair of the metrics P, R, D, L and N to check for correlations in subsets of the data, which we can compare with the overall trends from the *A* set.

RQ2: What proportion of reviews in each dataset contains a request?

Once a set of app reviews has been obtained, it can be useful to extract user requests from them. For example, this is useful for development prioritisation, bug finding, inspiration and otherwise requirements engineering [50, 82, 83, 104, 124, 126, 138]. We explore the subset of reviews that contain a user request, as this is the most studied in the literature.

RQ2.1: What proportion of user reviews does the Iacob & Harrison process identify? We use the process proposed by Iacob and Harrison [124] to identify requests in the set of mined reviews. Reviews are treated to the text preprocessing algorithm detailed in Section 3.5. The process detects requests on a per-sentence basis by matching key words in a set of linguistic rules; in this case the sentences are the entire review bodies as we remove punctuation as part of the text pre-processing (detailed in Section 3.5). This does not affect the matching except to reduce the number of possible matches per review to 1 (but we care about reviews that contain a request in this case, not how many requests they contain).

RQ2.2: What is the Precision and Recall of the extraction process? We assess the Precision and Recall of the request extraction algorithm, and compare its performance against a random guessing approach. We sample 1,000 random reviews from each of 4 sets: *Pa* (all reviews; requests), *F* (all reviews; requests). The sample of 1,000 is representative at the 99% confidence level with a 4% confidence interval [62]. Under the criterion that "*the user asks for ('requests') some action to be taken*", each sampled review is manually assessed as either a valid request or an invalid request.

RQ2.3: How do valid and invalid requests differ? We compare the set of correctly identified requests, the true positives (TP) with the set of falsely identified requests, the false positives (FP), in order to establish whether manual verification is necessary before using the extracted request data.

Using topic modelling (discussed in Section 3.6), allows for intersections of app descriptions and user requests that share content to be computed through use of a threshold probability value. A topic's 'strong contribution' to an app description or user request indicates that the topic falls above the topic likelihood threshold for that particular document. Experiments in this section use a topic likelihood threshold of $t = 0.05$, as is the standard in the literature.

Trained topics from each set, as well as derived topic metrics, are compared to establish how different the TP and FP sets are. The metrics P, R, D, L and N are used as before, except that at the topic level they are computed as the median of all apps to which a topic strongly contributes. Hence, we compare the metrics for each topic in the 2 sets, using the Wilcoxon test as well as the $\hat{A}_{12}$ effect size comparison.

RQ3: How are trends in requests differ in the sets Pa, F and A?

RQ1 seeks to establish an experimental approach to handling partial data, which we now apply to an instance, specifically request analysis. We compare the trends in each group of requests using correlation analysis. For comparison, we define 3 metrics at the topic level, using the topics from RQ2.3:

Ta: App prevalence: the number of app descriptions to which the topic strongly contributes.

Tr: Request prevalence: the number of requests to which the topic strongly contributes.

Td: Download rank: the median download rank of the apps to which the topic strongly contributes.

Table 5.4: **Manual topic classification results**[183]

Class	Description	Count
Valid	Defines one or more functional or non-functional properties of an app.	80
Review	Words highly coupled to review text such as `[error,load,sometimes]`.	8
App	Words highly coupled to app description text such as `[please,review,feedback]`.	6
Spam	Highlights spam in app descriptions and reviews used for advertising.	1
Other	Does not fit into another category but is not valid.	5
Total	All topics	100

For RQ3 we lower the topic likelihood threshold to $t = 0.02$ (0.05 was used in RQ2.3) to ensure a large enough set of topics for the statistical set comparison tests Wilcoxon and Vargha-Delaney's $\hat{A}_{12}$ measure. Using the set of topics from RQ2.3, we compare requests between *Pa*, *F* and *A* datasets using both the topics themselves and the three topic metrics. Because RQ3 concerns request properties and trends, we exclude the *Z* set for this question, since it cannot contain requests.

RQ3.1: Do the request sets from Pa, F and A differ? To answer this question we run the same 2-tailed, unpaired Wilcoxon test and $\hat{A}_{12}$ metric used in RQ1.1. We apply the statistical tests to the Ta, Tr and Td distributions between *Pa*, *F* and *A* datasets. This shows the difference between the dataset distributions, and indicates how the effect size differs. We apply Benjamini-Hochberg [24] correction for multiple tests, to ensure that we retain only a 5% chance of a Type 1 error.

RQ3.2: Are there trends within the request sets of Pa, F and A? We compare the trends of requests within each set by applying correlation methods to each pair of the topic metrics app prevalence, request prevalence and download rank, defined above. We run Spearman's correlation as used in RQ1.2, on the topic metrics Ta, Tr and Td. This enables us to identify how requests are affected by varying dataset completeness, and to establish whether we can learn anything from incomplete datasets.

Table 5.5: **RQ1.1: Vargha and Delaney's $\hat{A}_{12}$ effect size comparison results**. Large effect sizes (≥ 0.8) are marked in bold[183]

Datasets	P	R	D	L	N
Pa - F	0.533	0.529	0.564	0.554	0.630
Pa - Z	0.514	**0.965**	**0.834**	0.718	**1.000**
F - Z	0.547	**0.966**	**0.811**	0.763	**1.000**

5.7 Topic Modelling

This study uses the topic modelling process discussed in detail in Section 3.6.

Topic modelling with LDA leads to topics that reflect the source corpus, and so inevitably if spam exists in any of the user requests or app descriptions, it will also appear in some topics. For this reason we manually verified the 100 trained topics and classified them as valid or invalid, with the criteria that "the top 5 terms should refer to some functional or non-functional app property or properties". Classification results are presented in Table 5.4. An example valid topic includes the top terms: {`gps, speed, location, track, distance...`}, and an example invalid topic includes the topic includes the top: terms {`great, much, awesome, worth, well...`}. Henceforth only the 80 valid topics are used for comparison.

Topic settings are discussed in Section 3.6. We experimented with settings of $K = 100$, 200, 500 and 1000 topics, and found that the settings of $K = 500$ and $K = 1000$ led to over-specialised topics which reflected the descriptions of individual apps. The settings of $K = 100$ and $K = 200$ enabled more generalisation throughout the topics, and we selected $K = 100$ as we judged that it was the lowest setting without any app-specific topics.

5.8 Results

This section answers the research questions defined in Section 5.6.

RQ1: How are trends in apps affected by varying dataset completeness?

RQ1.1: Do the subsets Pa, F and Z differ? The Wilcoxon test results for metrics P, R, D, N and N between the sets *Pa*, *F* and *Z* were all < 0.001, and remained below 0.05 after we applied Benjamini-Hochberg correction for multiple tests. This shows

Table 5.6: **RQ1.2: Spearman's rank correlation results**. Non significant results ($p > 0.05$) are omitted; results in bold have strong correlation coefficients[183]
P: Price
R: Rating
D: Download rank
N: Number of ratings
L: Length of description

Set	PR	PD	RD	NP	NR	ND	LP	LR	LD	LN
Pa	0.16	0.12	-0.06	0.11	0.04	**-0.64**	0.23	0.09	-0.21	0.39
F	0.25	0.39	-0.03	0.12	0.06	-0.35	0.25	0.15	-0.06	0.32
Z	-0.06	-	-0.23	-	-	-	0.26	-0.20	-	-
A	0.20	0.22	-0.31	0.14	0.45	**-0.57**	0.27	0.27	-0.22	0.46

that there is a significant difference between sets *Pa*, *F* and *Z*. The Vargha-Delaney's $\hat{A}_{12}$ effect size comparison results in Table 5.5 show that the *Pa* and *F* sets have a small effect size difference: hence, *Pa* and *F* are unlikely to yield different results for the metrics tested.

This result is a contrast to the comparisons between sets *Pa* and *Z*, and *F* and Z. The *Z* set contains only apps that have not been reviewed and therefore have no rating, and hence have a large effect size difference with the N (number of ratings) metric. However, there is also a large effect size difference between the rating, download rank and description length between *Z* and the other sets. This shows that the *Z* set is very different from the other sets and indicates that it should be treated separately when analysing app store data.

All sets have a small effect size difference when comparing P (price), which might be expected because 90% of the apps in the set *A* are free.

RQ1.2: Are there trends within subsets Pa, F and Z? The results in Table 5.6 serve to further distinguish the need to separate the *Z* set when performing app studies, as it possesses almost none of the trends of the other sets. Indeed, almost all correlation results from the *Z* set were not statistically significant (marked with '-' in the table). Conversely, most results from the other sets were statistically significant, and the following (marked in bold), were strong:

Inverse RD correlation This indicates that as the (**R**)ating of apps increases, the (**D**)ownload rank decreases (a desirable effect since 0 is the top download rank with the most exposure). The RD correlation results were not particularly strong, and were present in the *A* and *Z* sets only. The correlation is caused by large numbers of non-rated apps in the *Z* dataset: zero rated apps have high download ranks and zero ratings, which skews Spearman's rank test.

Positive NR correlation This indicates that as the (**N**)umber of ratings increases, the (**R**)ating of apps increases. The trend was present in the *A* set only, and non-existent in the *Pa*, *F* or *Z* sets. Similar to the RD correlation result, this result is caused by the combination of non-rated and rated apps.

Inverse ND correlation This indicates that as the (**N**)umber of ratings increases, the (**D**)ownload rank decreases, which means that the app is more popular.

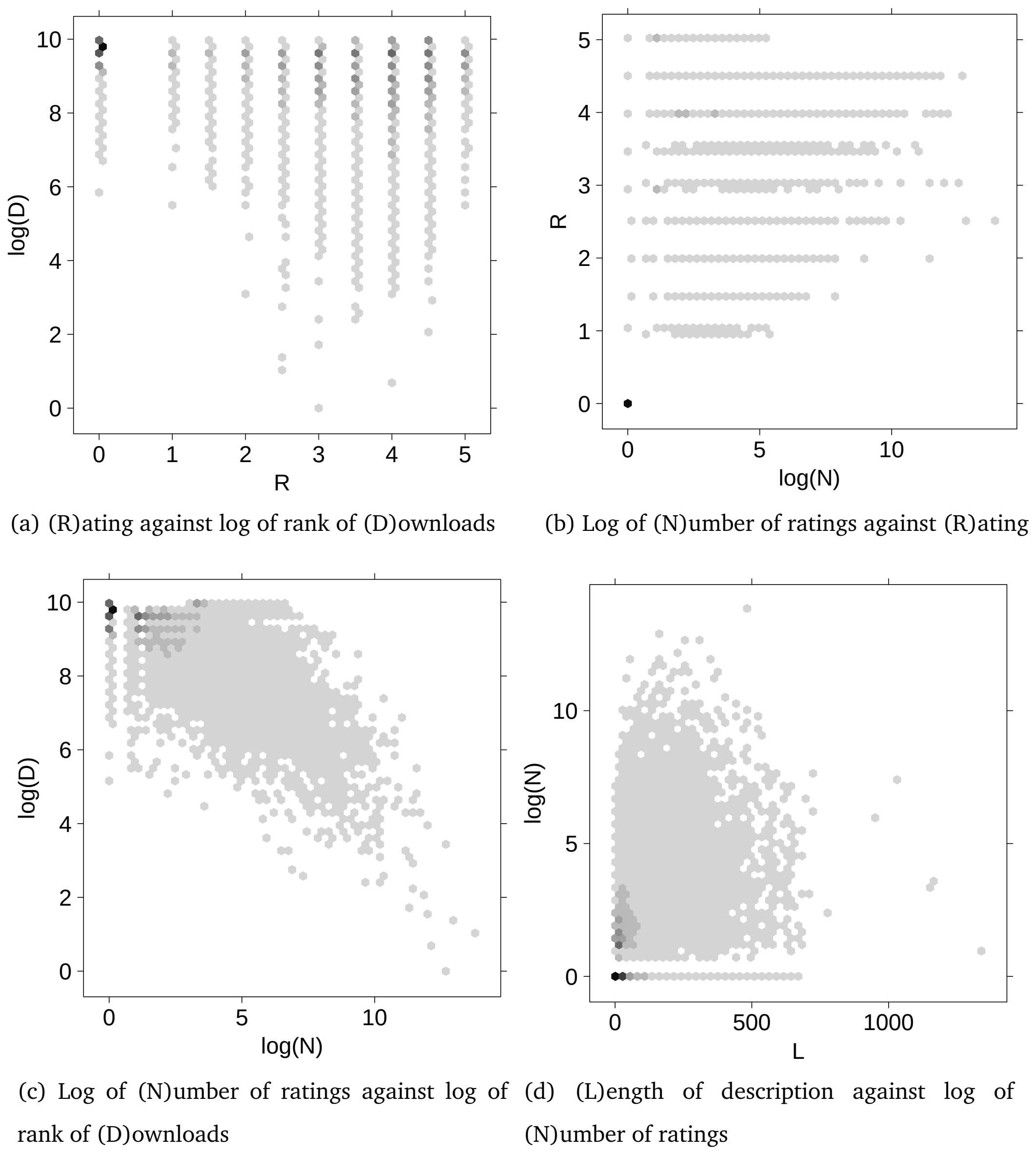
Positive LN correlation This indicates that as the (**L**)ength of descriptions increases, the (**N**)umber of ratings for the app increases. This result was consistent across both *Pa* and *F* datasets, and had an even stronger coefficient in the *A* set.

Figure 5.2 shows density scatter plots for the *A* set, for between RD, NR, ND and LN. Logarithmic scale is used where appropriate. Darker cells indicate greater app density than lighter cells. The plots indicate that the RD and NR correlations exist only because of a large number of non-rated, and therefore zero-rated, apps from the *Z* dataset. Results such as these emphasise the need to consider *Pa*, *F* and *Z* datasets separately.

RQ1: How are trends in apps affected by varying dataset completeness?
There is a significant difference between Pa, F and Z datasets, in some cases with a large effect size; the trends observed differ between datasets, especially when Z is included.

RQ2: What proportion of reviews in each dataset contains a request?

RQ2.1: What proportion of user reviews does the Iacob & Harrison process identify? Details of the extracted requests can be found in Table 5.7. The proportion of extracted requests is lower than that reported by the previous study of Iacob & Harrison [124], which can be attributed to two major differences in this study: **1) Blackberry World App Store is used in this study instead of Google Play.** It may be that user behaviour in reviews is different, manifesting in fewer requests. It may



(a) (R)ating against log of rank of (D)ownloads

(b) Log of (N)umber of ratings against (R)ating

(c) Log of (N)umber of ratings against log of rank of (D)ownloads

(d) (L)ength of description against log of (N)umber of ratings

Figure 5.2: **RQ1.2: Scatter density plots for (*A*)ll dataset** discussed in Section 5.8[183]

R: Rating
D: Download rank
N: Number of ratings
L: Length of description

also be that users use different words or phrases when making requests, and so the request extraction algorithm's recall is diminished. **2) Our dataset contains 19 times more reviews and 56 times more apps.** It therefore seems likely, due to the 'most popular first' nature of app data availability, that the dataset used in this study

Table 5.7: **RQ2.1: Request datasets**[183]
Pa: Apps their reviews where review sets are incomplete
F: apps and reviews where review sets are complete
Z: apps with no reviews
A: $Pa \cup F \cup Z$

Set	Apps	Reviews	Requests	Proportion
Pa	5,422	1,034,151	32,453	3.14%
F	6,919	1,694,952	74,438	4.39%
Z	2,754	0	0	0%
A	15,095	2,729,103	106,891	3.92%

contains proportionally more low ranked apps. It is possible that users make fewer requests in their reviews of such apps.

RQ2.2: What is the Precision and Recall of the extraction process? A sample of 1000 random reviews was taken from each of 4 sets, and each review was manually classified as containing a request for action to be taken or not. The sets used were the reviews and requests from each of the (*Pa*)rtially complete dataset and the (*F*)ully complete dataset. The manual assessment of request and review samples was completed by the author of this thesis in two days. Results from the assessment can be found in Table 5.8, and the computed Precision, Recall and F-Measure are found in Table 5.9. The resulting Precision is low, yet Recall is very high; the Iacob & Harrison algorithm [124] performs over 11 times better than random guessing. An example TP (true positive) request and FP (false positive) request can be read below.

TP request: "Would be nice to be able to access your account and rate the movies. Please include for next update."

FP request: "Amazing app every thing u need to know about real madrid is in this app and it deserve more than five stars"

A general observation from the sampling process was that the reviews that were identified as requests were not short, and always contained 'request-like' words. However, there was a large proportion of (FP) reviews which contained words like 'need', but did not ask for any action to be taken by the developers. Handling such cases is a challenge for content analysis of user reviews such as the Iacob & Harrison algorithm [124]. In order to deal with these cases more research is needed.

Table 5.8: **RQ2.2: Manual assessment of reviews**[183]
Pa: Apps their reviews where review sets are incomplete
F: apps and reviews where review sets are complete

Set	Type	Population	Sample	TP	FP	Precision
Pa	Request	32,453	1000	279	721	0.279
Pa	Review	1,034,151	1000	14	986	0.014
F	Request	74,438	1000	347	653	0.347
F	Review	1,694,952	1000	29	971	0.029

Table 5.9: **RQ2.2: Assessment of request algorithm**[183]
Pa: Apps their reviews where review sets are incomplete
F: apps and reviews where review sets are complete
TP: request identified was true positive
FP: request identified was false positive
FN: request identified was false negative

Set	TP	FP	FN	Precision	Recall	F-Measure
Pa	279	721	10	0.279	0.965	0.433
F	347	653	9	0.347	0.975	0.512

The set of FN (false negative) reviews, that were not identified as requests but were manually assessed to contain a request, asked for work to be done in different ways. Linguistic rules could be added to encompass the FN cases, but this would run the risk of further lowering the Precision. An example FN request is as follows: **FN request**: "Sharp, but layout can be worked on!"

RQ2.3: How do valid and invalid requests differ? Results comparing topics from different request groups are presented in Table 5.10. It can be seen from the results in Table 5.10 that the content of valid requests is broad, using 31 of the 80 topics; and that the set of 2,000 identified requests used 56 of 80 topics. The two sets of topics do not differ significantly, as shown in the right hand side of Table 5.10, where we use the Wilcoxon test to compare topic metrics between the two sets.

The observation that the topical content of identified requests is indistinguishable between sets of TP and FP requests shows that they do not differ greatly, and therefore FP requests have little impact on the topical content, but may affect its distribution. The high Recall result shows that it is possible to reduce the set of all reviews to a much smaller set that is more likely to contain requests, and this

Table 5.10: **RQ2.3: Comparison of TP and FP request topics**[183]
P: Price
R: Rating
D: Download rank
N: Number of ratings
L: Length of description
TP: request identified was true positive
FP: request identified was false positive

Set	**Topics**	**Metric**	**Wilcoxon p**	**Median difference**
TP	31	P	0.36	0.00
FP	47	R	0.39	0.00
Intersection	22	D	0.14	1322
Total	56	N	0.32	3
Total Available	80	L	0.39	4

process excludes very few requests falsely. We suggest that without a proven better performing method for data reduction of user reviews, it is sufficient to use the set of all requests identified by the Iacob & Harrison algorithm [124] for analysis.

RQ2: What proportion of reviews in each dataset contains a request? *Less than 3.92% of A, 3.14% of Pa and 4.39% of F reviews are requests in the Blackberry dataset, and we found the algorithm to have a recall* > 0.96 *and precision* < 0.350*. Manual inspection identified FN requests using different phrasing and keywords than those identified by the Iacob & Harrison algorithm.*

RQ3: How are trends in requests differ in the sets Pa, F and A?

RQ3.1: Do the request sets from Pa, F and A differ? The Wilcoxon test results for the metrics Ta (topic app prevalence), Tr (topic request prevalence) and Td (topic median download rank) between the sets *Pa*, *F* and *A* show that there is a significant difference between sets *Pa*, *F* and *A*. The $\hat{A}_{12}$ test results in Table 5.11 also show that there is a large effect size difference for Ta, Tr and Td between the three sets.

It can be seen from the results for RQ1.1 in Table 5.5 that there is a significant distribution difference, but a small effect size difference between the description lengths in *Pa* and *F*. This means that *Pa* and *F* are unlikely to yield different description lengths. It can seem surprising, therefore, that there is a large effect size difference between the app prevalence for topics between the *Pa* and *F* sets.

Table 5.11: **RQ3.1: Comparison results between request sets**[183]
Pa: apps their reviews where review sets are incomplete
F: apps and reviews where review sets are complete
Z: apps with no reviews
A: $Pa \cup F \cup Z$
Ta: number of apps topic strongly contributes to
Tr: number of requests topic strongly contributes to
Td: median download rank of apps topic strongly contributes to

Datasets	**Ta**	**Tr**	**Td**
Pa - F	0.794	0.974	0.999
A - F	0.831	0.839	0.874
A - Pa	0.914	0.998	0.971

Recall that app prevalence for a topic means the number of app descriptions to which the topic strongly contributes. This result is explained by the size difference of the sets: it stands to reason that in a set with more apps, a topic would be featured in more app descriptions than in the smaller set. The effect can be seen in greater magnitude with the larger request prevalence effect size difference. The *Pa* set has a large effect size difference in Tr from both the *A* set and the *F* set. These results again show that topics are used more frequently in the larger of the sets, an expected result.

The results for Td (topic download rank) are more surprising. Table 5.11 shows that the distributions of topic download rank in the three sets are very different, and have a large effect size difference. Bearing in mind that the *A* set is the combination of apps from *Pa*, *F* and *Z*, the *A - Pa* result suggests that the *Pa* set's median download rank is different from the overall median, while the *F* set's median is closer. However, the *Pa - F* results are surprising as they mean that for the same topics, one set leads to apps with greater median download ranks than the other; yet the results for RQ1.1 in Table 5.5 showed a small effect size difference in download rank for *Pa - F*.

One possible explanation is that the topic download ranks are exaggerating the difference in distribution medians. Because of the way the topic model is trained, each app description must have at least one assigned topic; likely more than one as the Dirichlet prior α is set to 50 [32]. Therefore, the entire set of app descriptions must be represented by only the 100 topics (80 of which are used here as explained in Section 3.6). It is not hard to imagine that the median download rank of apps in

Table 5.12: **RQ3.2: Request Spearman's rank correlation coefficient results**[183]
Pa: Apps their reviews where review sets are incomplete
F: apps and reviews where review sets are complete
Z: apps with no reviews
A: $Pa \cup F \cup Z$
Ta: number of apps topic strongly contributes to
Tr: number of requests topic strongly contributes to
Td: median download rank of apps topic strongly contributes to
-: no significant correlation coefficient ($p < 0.05$)

Set	**Ta - Tr**	**Ta - Td**	**Tr - Td**
Pa	-0.341	-0.230	-
F	**-0.596**	-	-
A	**-0.554**	-	-

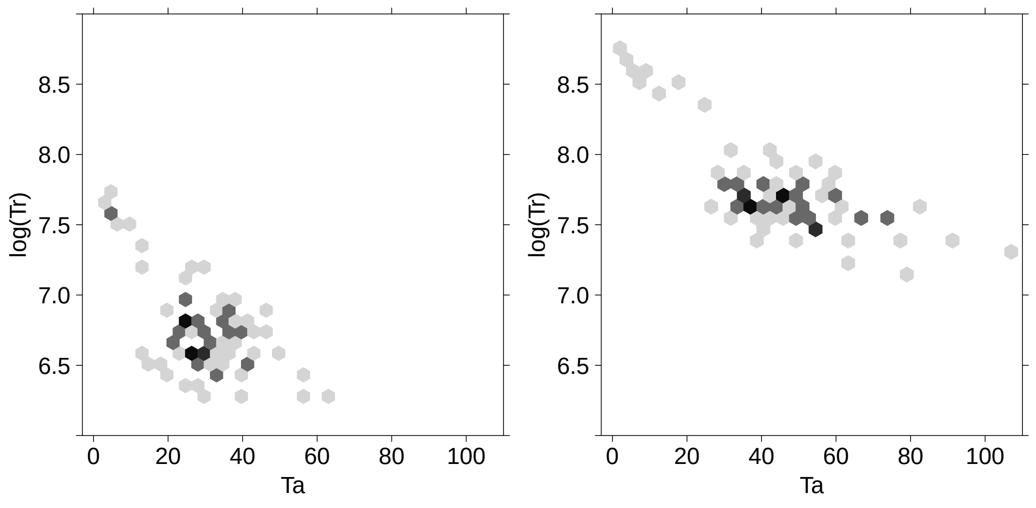
these 80 topics could be very different for *Pa*, *F* and *A*, even if the distributions of apps have small effect size differences in D.

RQ3.2: Are there trends within the request sets of Pa, F and A? Figure 5.3 shows that the *F* set has a higher median app prevalence and request prevalence for topics. That is, topics contribute in a greater number of apps and requests on average in the *F* set than the *Pa* set.

The results presented in Table 5.12 show that there is a strong negative correlation between the prevalence of topics in apps and requests. This is evidence that users tend to request things to a greater extent when they are present (or absent) in a smaller proportion of apps. The graphs in Figure 5.3 show that the scatter plot of Ta-Tr from the *F* set resembles the *Pa* set, but appears more stretched out and higher due to the greater number of apps and requests in the *F* set.

One might expect there to be a strong correlation between popularity and request prevalence, indicating that users request things present in more popular apps (or conversely, a negative correlation indicating users request more from low rated apps that may have less features), but this is not the case, as there was no significant correlation between the two metrics. A (speculative) interpretation of this result is that while users do not discriminate in terms of popularity, they do value diversity, hence the results of the negative prevalence correlation.

An important finding from these results is that the correlation between app and request prevalence exists not only in *Pa* and *F* datasets, but is actually stronger over-



(a) (*Pa*)rtial dataset (b) (*F*)ull dataset

Figure 5.3: **RQ3.2: Scatter plots for Ta against Tr**[183]
Pa: Apps their reviews where review sets are incomplete
F: apps and reviews where review sets are complete
Ta: number of apps topic strongly contributes to
Tr: number of requests topic strongly contributes to

all in the *A* dataset. This demonstrates that although the *Pa* dataset is incomplete and has different properties, the trends are consistent showing that it is still possible to learn things from the data.

RQ3: How are trends in requests differ in the sets Pa, F and A? *Trends in requests appear more robust to app sample bias than trends in apps; we found a strong inverse linear correlation between topic prevalence in apps and requests, that is consistent in Pa, F and A datasets.*

5.9 Actionable Findings for Future App Store Analysis

In this study, we have presented empirical evidence that indicates that the partial nature of data available on App Stores can pose an important threat to the validity of findings of App Store Analysis. We show that inferential statistical tests yield different results when conducted on samples from partial datasets compared to samples from full data sets; even if one were to exhaustively study the entire partial data set available, one would be studying a potentially biased sample.

The findings reported in this chapter suggest that this will be a potent threat to validity. Naturally, where researchers have full data sets available, they should sample (randomly) from these or base their results on the exhaustive study of the entire dataset. However, this raises the uncomfortable question: “What should we do when only a partial dataset is available?”. This question is uncomfortable, because partial datasets are so prevalent in App Store Analysis, as the survey in this study indicates.

Clearly, researchers need to augment their research findings with an argument to convince the reader that any enforced sampling bias is unlikely to affect their research conclusions and findings. Or, where possible, researchers can provide an assessment about the effect of the bias on their findings. This may be possible in some cases, since the sampling bias is often at least known, and sometimes quantifiable. For example there may be simply a bias on either recency or on popularity. Alternatively, researchers may choose to constrain study scope, making claims only about populations for which the sample is unbiased (e.g., all recent or popular apps on the store(s) studied).

Ideally, one would like to see further studies of other App Stores to confirm these results. Unfortunately, many App Stores do not make full information available, as the Blackberry World App Store did at the time we studied it. Nevertheless, where full information is available, further replication studies are desirable: correlation analysis of App Stores provides a potential tool to understand the relationship between the technical software engineering properties of apps, and the users’ perceptions of these apps. Understanding such relationships is one of the motivations for the excitement and rapid uptake of App Store Analysis.

5.10 Threats to Validity

This study is primarily concerned with addressing threats to validity, yet it also may have some of its own.

Internal validity: Our internal validity could be affected by issues with the data used in our experiments, which was gathered from the Blackberry World App Store. We therefore rely on the maintainers of the store for the reliability and availability of raw data. Due to the large-scale and automated nature of data collection, there may be some degree of inaccuracies and imprecisions in the data. However, we have made general observations on large sets of data, rather than detailed observations from individual apps, which provides some robustness to the presence of inaccuracies in the data. We also make the assumption when processing the (N)umber of ratings for an app, that reviews are not removed from the store. This would cause an app that is reviewed to appear non-rated.

The mined review data is parsed in the same way as app data, to capture each review's rating, body and author attributes; although the author information is only used to ensure that no duplicates are recorded as a means of reducing spam.

Were an app with many reviews to have some removed, it is unlikely that this would impact the overall findings, as the scale of the number of ratings provides some robustness to small changes. To the best of our knowledge, however, reviews are neither removed nor changed.

External validity: With regard to external threats, we return once again to the dataset. We mined a large collection of app and review data from the Blackberry World App Store, but we cannot claim that these results generalise to other stores such as those owned by Google or Apple. Rather, this study discusses the issues that can arise from using biased subsets of reviews, such as the kind available from App Stores.

Construct validity: Our construct validity is affected by our selection of experiments to measure the effect of the app sampling problem. We chose to compare empirical app metric distributions and trends, as well as topic trends, but this selection has surely missed other potential experiments which may be affected by the app sampling problem. We hope that the experiments chosen enlighten the reader as to the potential pitfalls of the app sampling problem, in order that they are able to avoid or mitigate its effects.

Conclusion validity: Our conclusion validity could be affected by the human assessment of topics and requests in answer to RQ2. However, our definitions and methodology can be applied more widely to assess the effect of the app sampling problem on any dataset.

5.11 Related Work

This section outlines related work in the field of app review analysis. A detailed survey of app store analysis literature can be found in Chapter 2.

User reviews are a rich source of information concerning features users want to see, as well as bug and issue reports. They can serve as a communication channel between users and developers. Several studies have utilised (*F*)ully complete user review datasets, and these analyse many more user reviews than those using (*Pa*)artially complete user review datasets: a mean of 9.8 million reviews are used by studies on *F* datasets, which starkly contrasts with a mean of 0.2 million reviews used by studies on *Pa* datasets. A summary of recent work on App Review Mining and Analysis can be found in Table 5.2.

In 2012 Hoon et al. [120] and Vasa et al. [274] collected an *F* dataset containing 8.7 million reviews from the Apple App Store and analysed the reviews and vocabulary used. Hoon et al. [119] then further analysed the reviews in 2013, finding that the majority of mobile apps reviews are short in length, and that rating and category influences the length of reviews. Another *F* sample was used in the 2013 study by Fu et al. [82], which analysed over 13 million Google Play reviews for summarisation.

Studies on *Pa* datasets use smaller sets of reviews, yet have produced useful and actionable findings. In 2013 Iacob and Harrison [124] presented an automated system for extracting and analysing app reviews in order to identify feature requests. The system is particularly useful because it offers a simple and intuitive approach to identifying requests. Iacob et al. [126] then studied how the price and rating of an app influence the type and amount of user feedback it receives through reviews. Khalid [138, 140] used a small sample of reviews in order to identify the main user complaint types in iOS apps. Pagano and Maalej [214] gathered a sample of 1.1 million reviews from the Apple App Store in order to provide an empirical summary of user reviewing behaviour, and Khalid et al. [139] studied the devices used to submit app reviews, in order to determine the optimal devices for testing.

Several authors have incorporated sentiment in their study of reviews. Galvis Carreño and Winbladh [83] extracted user requirements from comments using the ASUM model [133], a sentiment-aware topic model. In 2014 Chen et al. [50] produced a system for extracting the most informative reviews, placing weight on negative sentiment reviews. Guzman and Maalej [104] studied user sentiments towards app features from a small multi-store sample, which also distinguished differences of user sentiments in Google Play from Apple App Store.

Research on app reviews is recent at the time of writing, but much work has been done on online reviews [111, 121, 130, 131], all built upon by the papers mentioned. Morstatter et al. [202] carried out a similar study to this on Twitter data, finding that the subset of tweets available through the Twitter Streaming API is variable in its representation of the full set, available via the Twitter Firehose dataset.

5.12 Conclusions

Sampling bias presents a problem for research generalisability, and has the potential to affect results. We partition app and review data into three sets of varying completeness. The results from a Wilcoxon test between observable metrics such as price, rating and download rank between these partitions show that the sets differ significantly. We show that by appropriate data reduction of user reviews to a subset of user requests, we can learn important results through correlation analysis. For example, we find a strong inverse linear correlation between the prevalence of topics in apps and user requests. We build on the methods used by Iacob & Harrison [124] to extract requests from app reviews,in addition to using topic modelling to identify prevalent themes in apps and requests as a basis for analysis.

Chapter 6

Causal Impact Analysis for App Stores

Rapid release strategies can offer significant benefits to both developers and end users [144], but high code churn in releases can correlate with decreased ratings [100]. Releases occur for a number of reasons [2], such as updating advertisement libraries [101], or stimulating user downloads and ratings [55], in addition to more traditional bug fixes, feature additions and improvements. Developers may even find certain days of the week can stimulate user activity more than others [67, 113], although this may not be related to user preference [205]. McIlroy et al. [191] studied update frequencies in Google Play, finding that 14% of their studied apps were updated in a two-week period. Nayebi et al. [205] found that half of surveyed developers had a clear release strategy, and experienced developers believed it affects user feedback.

In this chapter, we study time-series information about apps, and identify how release frequency can affect an app's performance as measured by rating, popularity and number of user reviews. We introduce the method of Causal Impact Analysis, and describe our tool, `CIRA`, that performs this analysis for app store data. Using this tool, we identify the set of 'impactful' releases, that statistical evidence suggests caused a significant change in one of their app's performance metrics, compared with a baseline set of non-releasing apps. We then analyse the characteristics of these highly impactful app releases, in order to understand the properties under developer control which could lead to releases impacting performance.

6.1 Introduction

Causal inference is a method used to determine the causal significance of an event or events, by evaluating the post-event changes using observational data. A traditional approach to causal inference is differences-in-differences analysis [28], which compares the differences between the prior and posterior vectors of a group that receives a given intervention, against a group that does not receive this intervention. Conversely, the Causal Impact Analysis method [37], based on state-space models, treats each vector separately, in each case using the full control set to provide global variance.

In the context of app stores, these methods have the potential to assess the causal significance of releases upon an app's success in the store, in terms of user rating behaviour or download rank. Specifically, the prior and posterior vectors are observations of a metric such as rating, number of ratings or download rank, recorded at regular time intervals (i.e. weekly). The given interventions are new app version releases, as denoted by a change in the app's version identifier in the store. The group that does not receive the intervention (release) is the set of apps that do not release new versions in the studied time period.

Multiple apps typically do not receive the same intervention (release) at the same time, which makes differences-in-differences analysis difficult to apply in this case. Causal Impact Analysis, on the other hand, treats each app vector that receives a release separately, thereby making it better suited to this analysis of app releases. This motivates the use of Causal Impact Analysis. In this chapter we describe Causal Impact Analysis, and our tool, `CIRA`, that implements it.

Release: We determine a 'release' to have occurred if and only if the version identifier changes.

6.1.1 Process

Causal impact analysis trains a Bayesian structural time-series model [116, 249] on the data vector for each target release, using a set of unaffected data vectors known as the control set (defined in Section 7.4.2). This enables the model to make a prediction of the data vector in the posterior time period accounting for local and global variations. Each metric (R, N, NW) and each release, requires an individual experiment, as the method works each time on a single data vector.

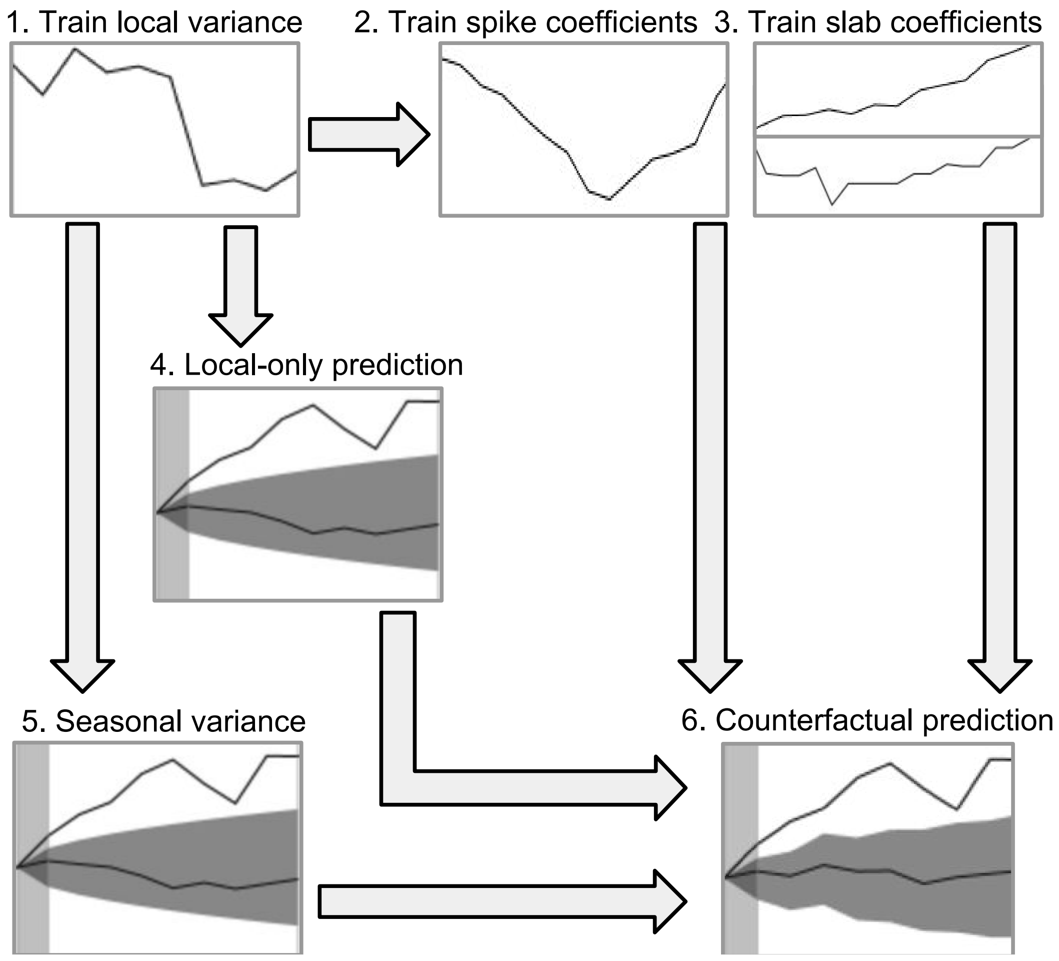
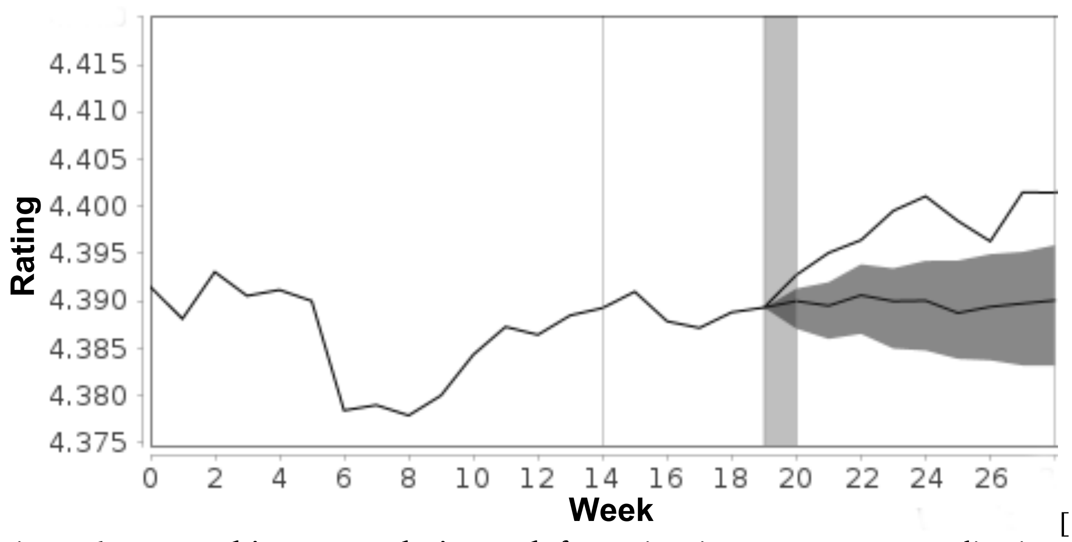

1. Train local variance
2. Train spike coefficients
3. Train slab coefficients
4. Local-only prediction
5. Seasonal variance
6. Counterfactual prediction

Figure 6.1: **Causal Impact Analysis workflow**



[185]

Figure 6.2: **Causal impact analysis graph** for an invoice management application in Google Play
Shaded vertical bar: target release
Solid line plotted throughout: observed vector
Shaded region: confidence interval of 0.95
Solid line inside shaded region: counter-factual prediction

Figure 6.1 shows the overall Causal Impact Analysis workflow:

1. Train the local trend parameters using the deviation of the observed vector in the prior time period. This is used to sample changes and compute the confidence interval.

2. Compute the 'spike' [249] for the observed vector from the set of controls, and assign coefficients for them, in order to make an accurate prediction.

3. Use the rest of the control set as the 'slab' [249], assigning equal coefficients for them, in order to account for global variations in the dataset.

4. Make a set of predictions using the local trend trained earlier, sampling for changes and noise.

5. Compute seasonal variance (optional). This is not used as app store ratings do not demonstrate cyclic behaviour over the maximum studied time (52 weeks).

6. Combine local-only prediction with control changes multiplied by their (static or dynamic) coefficients. This is the counterfactual prediction [37].

We then compute the cumulative pointwise difference between the observed vector and the prediction, normalised to the length of time at which each deviation occurs, and compute the cumulative probability from the local trend parameters.

A low value ($p \leq 0.01$) indicates that the success metric changed significantly; see for example Figure 6.2, which shows that the rating of an invoice management application deviates significantly from the predicted vector. By setting the threshold to be 0.01, there is a 0.01 probability of claiming a significant change where one does not exist, and therefore expect roughly 1% false positive rate.

We use CIRA to identify the set of releases, for which there is evidence of a statistically significant effect on the metrics we collect. We refer to such releases as **impactful releases** in this thesis.

6.2 Causal Impact Release Analysis Tool (CIRA)

This section describes the tool, `CIRA`, that we have implemented in order to facilitate further study on app store data using causal impact analysis.

6.2.1 Design

`CIRA` is written in `Java` version 1.8 [213], making full use of lambda expressions for increased efficiency and potential for runtime parallelisation. `Maven` version 2 [268] is used to control the build, allowing for modularisation and testing. Unit tests are implemented in `JUnit` [136], and the following additional libraries are used: `apache.commons.math3` [267] is used for statistics and regression; `guava` [93] is used for general utility functions; `JFreeChart` [211] is used for plotting graphs; as is `JMathPlot` [301] which was used for an earlier version of the prediction graphs.

The module structure used is shown in Figure 6.4. Three main modules are used for command line functionality: `cira`, `graph` and `runner`. The `runner` module serves to run the command line tool that provides batch computation for any of the different modules that are implemented, as well as graph drawing on an individual basis.

Supplementary modules were implemented that provide easier usability, for demonstration purposes. The `tool` module provides an executable `jar` binary that can be run cross-platform on desktop operating systems (provided they have a compatible version of `Java` installed). Similarly, the `server` module provides a servlet that runs on a `Tomcat` [266] server, providing access from any web browser.

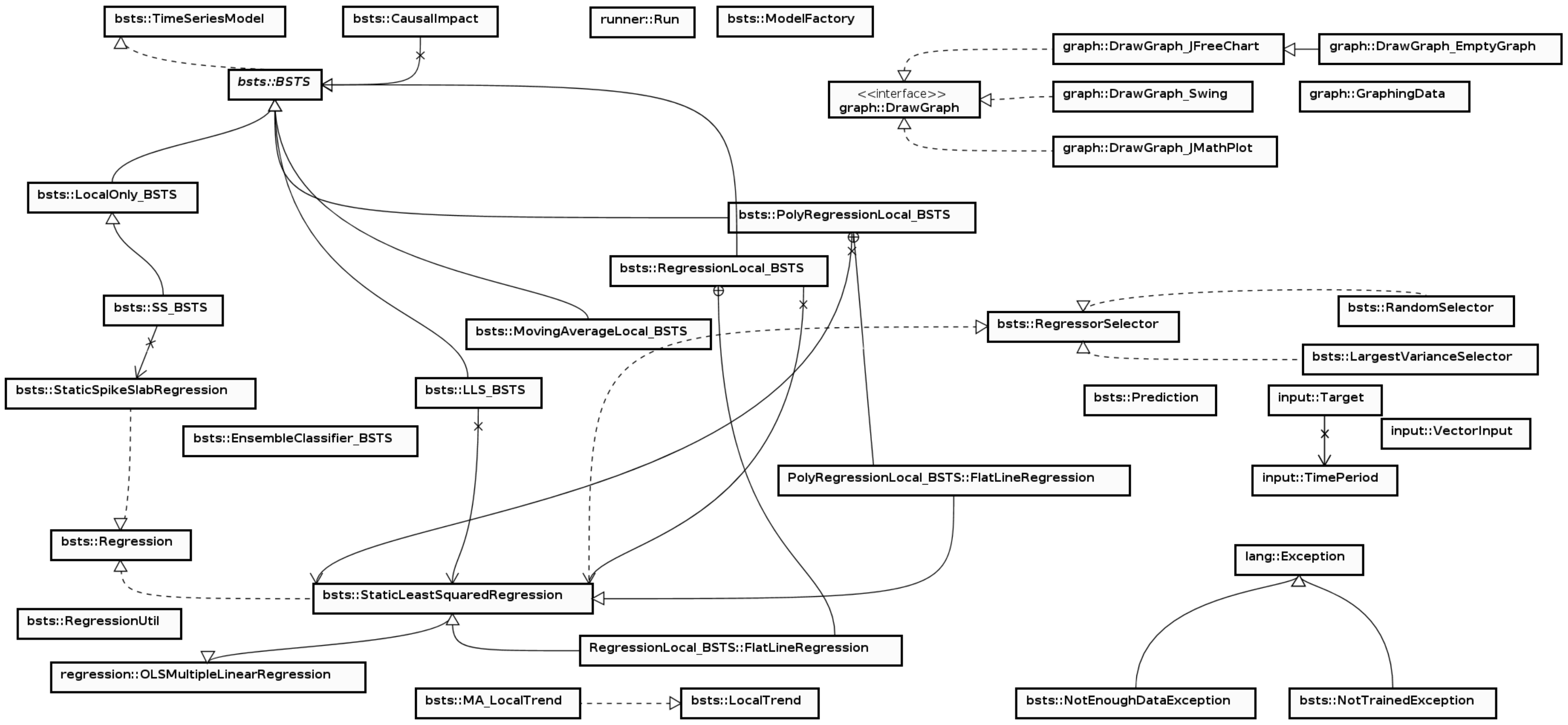

bsts::TimeSeriesModel
bsts::CausalImpact
runner::Run
bsts::ModelFactory
graph::DrawGraph_JFreeChart
graph::DrawGraph_EmptyGraph
bsts::BSTS
<<interface>>
graph::DrawGraph
graph::DrawGraph_Swing
graph::GraphingData
graph::DrawGraph_JMathPlot
bsts::LocalOnly_BSTS
bsts::PolyRegressionLocal_BSTS
bsts::RegressionLocal_BSTS
bsts::SS_BSTS
bsts::RandomSelector
bsts::MovingAverageLocal_BSTS
bsts::RegressorSelector
bsts::LargestVarianceSelector
bsts::StaticSpikeSlabRegression
bsts::LLS_BSTS
bsts::Prediction
input::Target
input::VectorInput
bsts::EnsembleClassifier_BSTS
PolyRegressionLocal_BSTS::FlatLineRegression
input::TimePeriod
bsts::Regression
lang::Exception
bsts::StaticLeastSquaredRegression
bsts::RegressionUtil
RegressionLocal_BSTS::FlatLineRegression
regression::OLSMultipleLinearRegression
bsts::MA_LocalTrend
bsts::LocalTrend
bsts::NotEnoughDataException
bsts::NotTrainedException

Figure 6.3: **CIRA class diagram** Dotted line: inheritance Solid line: subclass

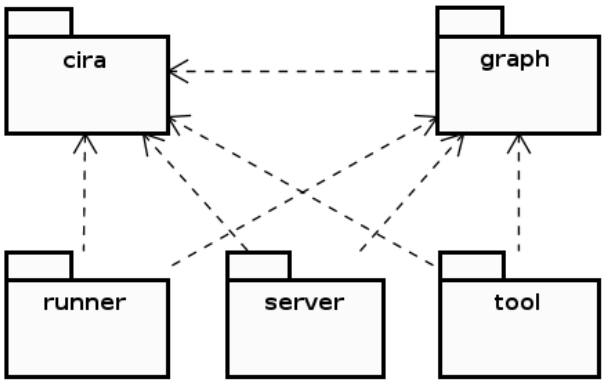


Figure 6.4: **CIRA module diagram** Dotted line: dependency

The `CIRA` class diagram for the three main modules is shown in Section 6.2.1, and a module diagram is shown in Figure 6.4.

6.2.2 Program Flow

Figure 6.5 shows the workflow of `CIRA`, which is the algorithm as described in Section 6.1.1, with the exclusion of the seasonal variance computation (originally step 5.). The seasonal variance is excluded from `CIRA` since, as noted in Section 7.3, the app store data studied does not exhibit signs of cyclic variance.

`CIRA` takes as input two files: `controls` and `targets`. These files must be in the same format as those for `CausalImpact`. The format for `CIRA` input is specified in the usage manual in Appendix D.

The local trend is computed by evaluating the standard deviation in the prior time period, σ, and incorporating two variance modulators based on this, v_v and v_d. These modulators serve to add general variance, and delta variance, respectively. For simplicity they are assumed to have the same distribution:

$$v_v \sim \mathcal{N}(0,\sigma^2) \qquad v_d \sim \mathcal{N}(0,\sigma^2) \tag{6.1}$$

The counterfactual local trend is evaluated by sampling each of v_v and v_d for each counterfactual value, and adding their modulations to the last counterfactual value (starting with the final observed value in the prior time period):

$$counterfactual_i = counterfactual_{i-1} + v_v + v_d \qquad \text{for } 1 < i < |posterior| \tag{6.2}$$

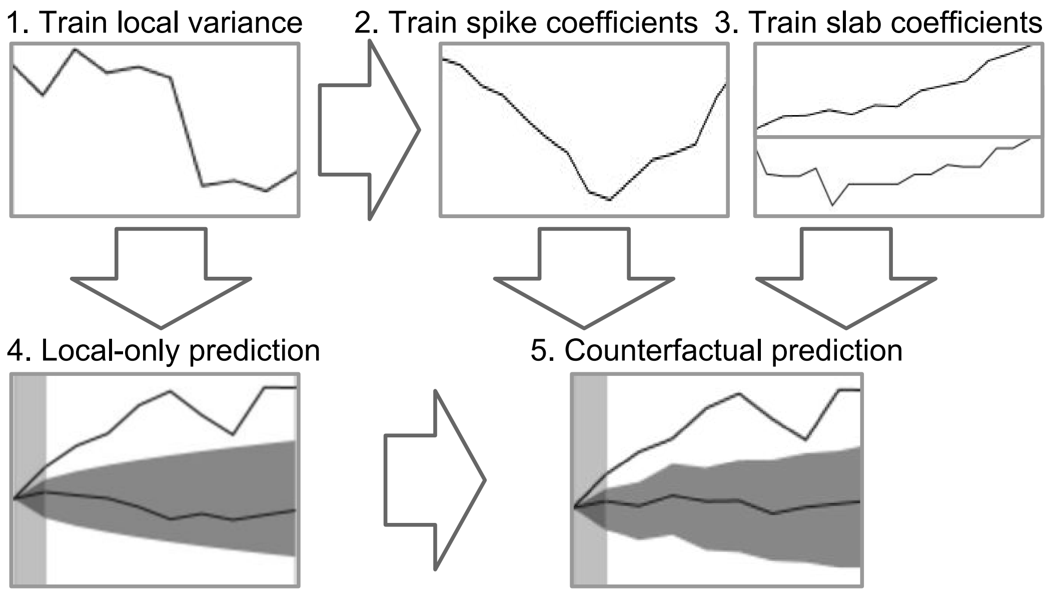

1. Train local variance
2. Train spike coefficients
3. Train slab coefficients
4. Local-only prediction
5. Counterfactual prediction

Figure 6.5: **CIRA workflow** explained in Section 6.2.2.

where:

$$counterfactual_0 = prior_{|prior|} \tag{6.3}$$

The global trend is computed by assigning spike coefficients to a predetermined number of controls (the default is 3), and slab coefficients to the remaining controls. `CIRA` assigns spike coefficients using regression: partitions are assigned, of size equal to the prior time length - 1, and regression is performed using each partition. Half of the partition is eliminated (the half with the lowest coefficients), and this process is repeated until the desired number of spike regressors is reached. In this way, the controls that most accurately predicted the observed vector are selected. The coefficents themselves are chosen by performing regression with the chosen vectors, and then multiplying the coefficients by a spike weight, w_{spike}, divided by the number of spike coefficients. Spike regressors could be assigned using alternative methods, but good agreement performance (with `CausalImpact`) was achieved using this method.

Slab coefficients are assigned as:

$$c_{slab} = \frac{w_{slab}}{|slab|} \tag{6.4}$$

where c_{slab} is a coefficient for a slab control, w_{slab} is the weight assigned the slab portion of coefficients, and $|slab|$ denotes the number of slab controls.

The default spike and slab weights used in `CIRA` are:

$$w_{slab} = 0.5 \qquad w_{spike} = 0.5 \tag{6.5}$$

Different values can be used, but we found good performance giving equal weight to spike and slab coefficients.

6.2.3 Agreement with CausalImpact

When the number of control vectors in increased, the execution time of `CausalImpact` increases exponentially, whereas `CIRA` is affected in order $O(n)$ (in terms of additional regression runs) where n is the number of control vectors, and the complexity order of each regression run is unaffected. This is an advantage of `CIRA`, and a specialisation for app store data, where the number of control vectors available may be in the thousands. In contrast, the use case of web site ad campaign data is more likely to be in the tens.

In order to assess the accuracy of `CIRA`, We compute the agreement:

$$agreement = \frac{YY + NN}{YY + NN + YN + NY} \tag{6.6}$$

and Cohen's Kappa [54], that takes into account the chance to randomly agree, and is bounded above by *agreement*.

We used the dataset that will be used in Section 7.3, consisting of 442 targets and 100 controls, and manually assessed each target graph to indicate 'human agreement' with the impacts. This, of course, represents a threat to validity, for the human's understanding and assessment of causal analysis, but helps to provide context for the agreement assessment.

Section 6.2.3 shows the results of the agreement test between human, `CausalImpact` and `CIRA`. The results in Section 6.2.3 show that `CausalImpact` and `CIRA` show very similar agreement and Cohen's Kappa with the human assessment, of approximately 75% agreement and 0.35 Cohen's Kappa. This indicates that the approaches both agree with the manual assessment 75% of the time, of which about half of the agreement may be random. `CausalImpact` and `CIRA` are shown to have 85% agreement with each other, of which again about half of the agreement may be random. The approaches exhibit greater agreement with each other than with the manual assessment, indicating that they are providing similar

Table 6.1: **Causal impact analysis agreement**
Ag: agreement
CK: Cohen's Kappa

	CausalImpact		**CIRA**			**CausalImpact**	
	Ag	**CK**	**Ag**	**CK**		**Ag**	**CK**
Human	0.74	0.36	0.76	0.34	**CIRA**	0.85	0.45

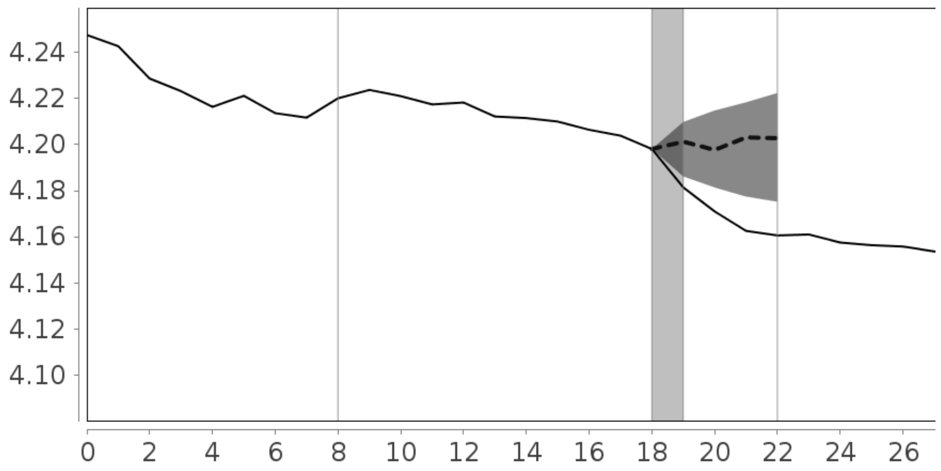


Figure 6.6: **Real-world prediction issue** flatlining despite an apparent prior trend
Shaded vertical bar: target release
Solid line plotted throughout: observed vector
Shaded region: confidence interval
Dotted line inside the shaded region: counter-factual prediction

results. The runtime of `CIRA` was approximately 428 times faster for this dataset, with a median runtime of 1.916 seconds over 5 runs, compared with a median of 820.340 for `CausalImpact`.

6.2.4 Web Access

`CIRA` is available for use via the integrated `Server` module, at `http://www0.cs.ucl.ac.uk/staff/W.Martin/cira`.

6.3 Extension 1 – Ensemble Classifier

The default mode in `CIRA` (known internally as SpikeSlab) relies on the control set spike partition to help predict the counter-factual trend. However, sometimes this prediction falls short of what a human, using their own judgement, would call obvious. This is noticeable when an observed vector is increasing at a steady rate in the prior time period, yet the counterfactual prediction plot flat-lines. This effect can be seen in Figure 6.6 and Figure 6.7a.

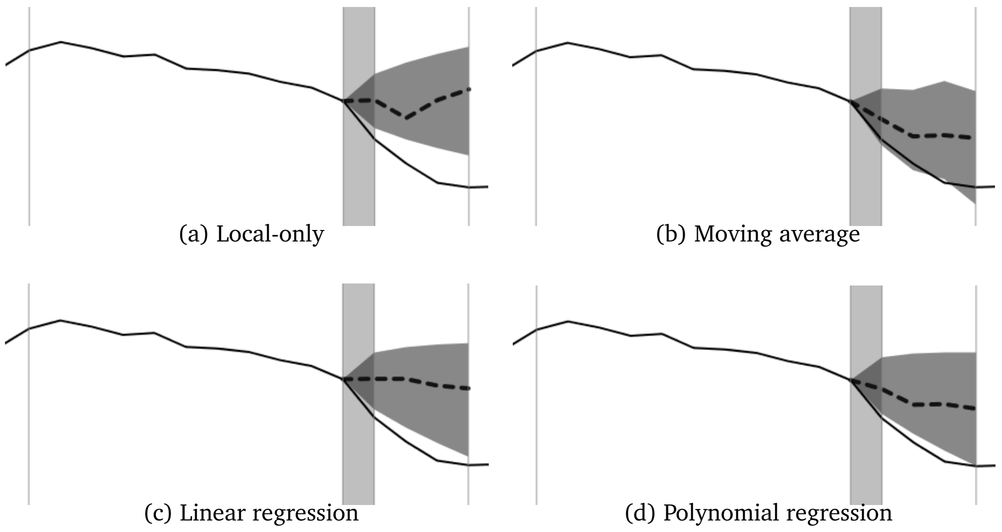


Figure 6.7: **Comparison of CIRA baseline methods** for correcting flatlining
Shaded vertical bar: target release
Solid line plotted throughout: observed vector
Shaded region: confidence interval
Dotted line inside the shaded region: counter-factual prediction

To combat this issue, we have implemented several alternative means to 'direct' the counterfactual prediction:

Moving average: The moving average from the last 3 weeks of the prior time period observed vector is computed, and the counterfactual prediction is adjusted prior to the local trend variance samples, in order to 'continue' the trend. This technique can be seen in Figure 6.7b.

Linear regression: Regression is trained on the prior time period, using an arbitrary increasing predictor variable equal to the week number, and using the observed vector each week as the response. The prediction is then carried out each week in the posterior time period, before the local trend variance samples are added. This technique can be seen in Figure 6.7c.

Polynomial regression: This works similarly to linear regression, except it uses multiple powers of the predictor variable (week) for training and prediction. The method is implemented to use as many powers as possible: i.e. 2 powers for a prior time period of 3. This technique can be seen in Figure 6.7d.

We computed the agreement between each of the different prediction methods, and present results in Table 6.2. Each of the methods performs similarly, with the

Table 6.2: **CIRA models agreement** on sample of 442 targets using 100 controls
SAg: agreement
CK: Cohen's Kappa
Local: local-only 'standard' prediction
MA: moving-average prediction
LR: linear regression prediction
PR: polynomial regression prediction

	Local		**MA**		**LR**		**PR**	
	Ag	**CK**	**Ag**	**CK**	**Ag**	**CK**	**Ag**	**CK**
Human	0.75	0.33	0.72	0.31	0.74	0.33	0.63	0.22
CausalImpact	0.85	0.47	0.72	0.22	0.79	0.32	0.62	0.15
CIRA (SpikeSlab)	0.95	0.77	0.82	0.41	0.88	0.54	0.66	0.21

exception of polynomial regression, which appears to over-train on the prior time period (shown in Figure 6.7d).

`CIRA`'s SpikeSlab method achieves higher overall agreement with the human assessment, and with `CausalImpact`, than the other methods, and therefore it remains the default model. Each of the models is available through `CIRA`'s command line execution, and furthermore we have implemented an "ensemble classifier" that runs each model in turn, and presents the user with results from each model, as well as the overall confidence in its assessment. All of these methods evaluate the prediction in the same as way as `CIRA`, by computing the inverse cumulative probability of the difference, after normalisation.

6.4 Extension 2 – Feature Regression

It would be useful to tie features directly to impacts, and to use them to make counterfactual predictions with greater accuracy. This information could then be used to predict which features could be added in order to produce an impactful release for a specific app. To facilitate this, we have implemented a feature regression component in `CIRA`, that works in a similar way to the spike partition of the spike-slab prior method [37] described in Section 6.1.1.

This extended method uses repeat runs of regression in order to select the features that most strongly predict the observed vector in the prior time period, and assigns coefficients to them which add to the overall SpikeSlab prediction. The model takes as input the document vector for each release, and the feature vector for each document.

This model assumes that the feature vectors are fluctuating vectors that can be used to help predict the response vector. Unfortunately, these assumptions do not hold true for any of the collected apps: descriptive text is changed too rarely for this method to work properly. Hence, we have conducted an alternative study into features impacting releases in Section 7.5. However, this module remains available in the `CIRA` codebase as it may prove useful to integrate additional information into the model.

6.5 Threats to Validity

Internal validity: Our internal validity may be affected by the comparison of `CIRA` to `CausalImpact`, when `CIRA` is in fact based upon `CausalImpact`. We mitigate this threat by comparing the classification results of both tools with a human assessment of releases.

External validity: Our external validity may be affected by the dataset we used, and the scale of the metric changes inherent in it. This threat is mitigated by the adaptive nature of causal impact analysis, in that the scale (and thus significance) of a change is relative to its standard deviation prior to the intervention.

Construct validity: Our construct validity may be affected by our agreement tests, where we tested the classification results of each tool and algorithm only, without testing the effect size of the significant change detected. However, in our studies we apply causal impact analysis as a classifier only, and so this seems an appropriate agreement test in our case. Future work may compare the effect size of significant changes detected.

Conclusion validity: Our conclusion validity may be affected by our understanding of causal impact analysis. We mitigate this threat by comparing classification results between the tools directly, and by comparing the results with a human assessment.

6.6 Conclusions

`CIRA` is a `Java` based implementation of causal impact analysis [37], that enables the use of large scale control sets, such as found in app stores when using non-releasing apps as the control set. `CIRA` differs from `CausalImpact` in its approximation of the local delta variance parameter v_d, and its choice and parameters of the control set spike partition. Results have shown strong agreement between `CausalImpact` and `CIRA`, and a human evaluation of a sample of releases indicates that both methods perform similarly. A number of modules are built into `CIRA`, in order to facilitate future studies: a desktop tool, a web servlet tool, an ensemble classifier using alternative prediction techniques, and a feature regression predictor.

Chapter 7

What are the Properties of Impactful Releases?

Causal Impact Analysis [37] (discussed in more detail in discussed in Chapter 6) is a method for identifying significant events which may have impacted time-series vectors. In this chapter we apply causal impact analysis to identify significant 'impactful' app releases. This is done in the first instance using the tool written by Brodersen et al., `CausalImpact` [36], and subsequently using our own tool, `CIRA` (described in more detail in Section 6.2). We follow up on the causal impact analysis with more traditionally familiar (frequentist) inferential statistical analysis to further investigate the probabilistic evidence for potential causes, characteristics and effects.

7.1 Introduction

Causal inference[175] is primarily used in economic forecasting, for measuring or predicting the effect of an event on time-series data, but it has also seen recent use for software defect prediction [60, 61, 306]. We apply causal impact analysis to app releases in order to identify "impactful releases", that significantly impacted subsequent user rating behaviour or download rank. The subsequent inferential statistical analysis in Sections 7.3 to 7.5 complements this analysis, pointing to the set of potential properties which may play a role in this causal significance. Of course, the degree to which one can assume causality relies on the strength of their causal assumptions, that the control set is unaffected by the release, and the relationship of the control to the released app is unchanged.

We compare distributions using a two-tailed unpaired non-parametric Wilcoxon test [295], that tests against the Null-hypothesis that the result sets are sampled from the same distribution. We also compare the result sets using Vargha and Delaney's $\hat{A}_{12}$ effect size comparison test [273], which results in a value between 0 and

1, that tells us the likelihood that one measure will yield a greater value than the other. In this case, we apply these inferential statistical tests to examine differences in properties between the sets of releases that have demonstrated a causal impact (using causal impact analysis) and those which have not, as well as between the sets of releases that positively impacted rating and those that negatively impacted rating.

To analyse the text of impactful releases, we perform information retrieval analysis on app descriptions and "what's new" text, using Topic Modelling and, in the case of Sections 7.3 and 7.4, also TF.IDF (both techniques are described in Chapter 3). We use these two different techniques to increase the confidence with which we can identify the top terms that occur in impactful 'release text'. Text is processed according to the steps detailed in Section 3.5.

In any and all causal impact analyses, it is of course impossible to identify *all* external properties (such as advertising campaigns) that might have played a role in the changes observed. In the case of app stores, our current analysis cannot, for example, identify advertising campaigns, timed to coincide with the new release. However, if such an external factor does have a significant effect, then causal impact analysis may detect it. We ask developers in Section 7.4 if they are aware of external factors which may have caused the observed changes, in order to determine how often this may be the case.

7.2 Developer Controller Metrics

The developer-controlled metrics we use in experiments in this chapter are given in Table 7.1. We also use (P)rice, defined in Table 3.3. For Google, we measure release text as the combination of description and 'what's new' text (description for Windows). Here we do not insist that release text must have changed from a previous week, since this is reflected by $\text{RT}_{change} = 0$.

Table 7.1: **Developer controlled metrics**

Metric	**Description**
Day	Day of release (i.e. Monday, Tuesday)
RT_{size}	The size of RT (release text) in words
RT_{change}	The change in RT_{size} on the week of release

7.3 What are the Characteristics of Impactful App Releases?

This work was published in ICSE 2016 Companion Proceeding [182]. The first author's contribution to this paper was to formulate the idea, implement and execute the experimentation and collect the results and analyse them; other authors of the paper contributed to research question formulation, result analysis and narrative write up.

App developers would like to know the characteristics of app releases that achieve high impact. To address this, we mined the most consistently popular Google Play and Windows Phone apps, once per week, over a period of 12 months. In total we collected 3,187 releases, from which we identified 1,547 for which there was adequate prior and posterior time series data to facilitate causal impact assessment, analysing the properties that distinguish impactful and non-impactful releases. We find that 40% of target releases impacted performance in the Google store and 55% of target releases impacted performance in the Windows store. We find evidence that higher prices, day of release and fewer mentions of bug fixing can increase the chance for a release to be impactful, and evidence that higher prices, more descriptive release text and new features rather than bug fixes can increase the chance for a release to improve rating.

7.3.1 Findings

Looking ahead to the results of this section, our findings are as follows:

i) Higher priced releases have a greater chance of impacting performance in the Google store, but the opposite is true in the Windows store: impactful release prices have a mean of £0.99 compared with £0.65 for non-impactful release prices in the

Google store, and £0.47 compared with £0.54 in the Windows store.

ii) Of all impactful releases, higher priced releases have a greater chance of positively impacting rating: releases that positively impacted rating have a mean price of £1.03 compared with £0.78 for those that negatively impacted rating in the Google store, and £0.57 compared with £0.38 in the Windows store.

iii) A higher probability of impact for releases between Saturday and Tuesday, with up to 46% proving impactful on Tuesday, as low as 36% on Wednesday in the Google store, and up to 63% on Saturday (with as low as 48% on Thursday) in the Windows store.

iv) Impactful releases had proportionally fewer mentions of (bug, fix) in their release text than non-impactful releases: 33% to 38% in the Google store and 44% to 48% in the Windows store.

v) Releases that positively impacted rating had proportionally fewer mentions of (bug, fix) than those that negatively impacted rating: 29% to 33% in the Google store and 38% to 43% in Windows store, and more mentions of (new, feature): 30% to 25% in Google store and 55% to 39% in Windows store.

vi) Of all impactful releases, those with longer release text have a greater chance of positively impacting rating: those that positively impacted rating have a median filtered word count of 25 (to a median of 19 for those that negatively impacted rating) in Google store; 135 to 119 in Windows store.

vii) Our results indicate that Causal Impact Analysis is useful for identifying impactful releases for further analysis.

7.3.2 Data

We mined app data from Google Play and Windows Phone Store between July 2014 and July 2015, as detailed in Section 3.2.

The download rank is unavailable for apps outside the 'top' free or paid lists for both Windows Phone and Google Play app stores. For Google Play, this information is available for only the most popular 540 free and 540 paid apps, while for Windows Phone, download information is available for only the most popular 1000 free and 1000 paid apps.

We therefore mined apps from the Google Play and Windows Phone stores (on a weekly basis), recording the most popular 540 free and 540 paid apps from Google, and top 1000 free and 1000 paid apps from Windows. The time period that we considered was from July 2014 to July 2015, giving 52 snapshots for each of Windows Phone and Google Play. In order to focus solely on the consistently popular apps, we computed the intersection of 52 snapshots. This resulted in 307 consistently popular (i.e. always within the top 540) Google apps, and 726 consistently popular (i.e. always within the top 1000) Windows apps. These 1,033 apps are those studied in this Section.

Our conclusions thus concern the effect of releases on the most consistently popular apps over a period of the year (July 2014 to July 2015), and care will be required before the results could be extended to apps and their releases more generally, due to the App Sampling Problem [183]. Nevertheless, we believe that conclusions about the characteristics of impactful releases for the most consistently popular apps will yield interesting and actionable findings for developers, because consistently popular apps are an inherently attractive and interesting subset of app stores.

We extract app metrics from each of the 1,033 apps including price, rating, download rank, number of ratings, description, what's new and version. In the case of Google Play, the app store reports rounded app ratings (rounded to 1 decimal place), thereby creating a potential source of imprecision, which we would like to overcome. To improve comparability between the two app stores, we would also like to ensure that ratings are computed to the same precision. Therefore, we recalculate the (more precise) Google Play average ratings, using the extracted numbers of ratings in each of the five star ratings (from 1-5).

We use the following data to answer RQ3 and RQ4:

Control set: apps that have no releases in the 12 month time period: for Google this set is 97 apps, and for Windows this is 397 apps. We compare different control set sizes in RQ3.2 in order to establish whether the choice of control set affects the results.

Target releases: releases that occurred at least 3 weeks after the previous release of the same app (or start of the time-series), and that occur at least 3 weeks before the next release (or end of the time-series). This ensures sufficient data availability to accurately train causal impact analysis.

Impactful release: a release for which one of the performance metrics (R, D, N and NW) significantly deviated from the counterfactual prediction. We consider a significant deviation to be one in which the probability of predicting the observed is ≤ 0.01. This is a cautious choice of probability (corresponding to the 99% confidence interval) to reduce the likelihood of raising false alarms, which would subsequently turn out not to be truly impactful after all. By setting the threshold to be 0.01, there is a 0.01 probability of claiming an impact where one does not exist, and we therefore expect a false positive rate of roughly 1%.

7.3.3 Research Questions

This subsection explains the questions posed in our study, and how we approach answering them.

RQ1: Do app metrics change over time?

Before performing detailed analysis of the changes over time, we first set a baseline by establishing whether app performance metrics change between snapshots. Using the metrics R, D, N and NW as defined in Section 3.4, we compute their standard deviations over 52 weeks for each app, and draw them on box plots. This enables us to establish whether metrics change over time, and to what extent this occurs. If the metrics do change over time, it motivates further analysis of events that could cause changes over time, specifically releases.

RQ2: Do release statistics have a correlation with app performance?

We measure whether app performance is affected by the number of app releases, by measuring the correlation between performance metrics and the number of releases in 52 weeks, as well as the change in metrics over 52 weeks.

RQ2.1: Does the number of releases have a high correlation with app performance? We perform correlation analysis between the number of releases of each app and their current value for the metrics R, D and N. We do not use NW because this number is set on a per-week basis, and instead use the change in R, D and N from the first snapshot to the last, denoted ΔR, ΔD and ΔN respectively.

RQ2.2: Does the median time interval between releases have a correlation with app performance? We perform correlation analysis between the median interval between multiple releases of each app and the metrics used in RQ2.1. For this, only apps that have releases in the time period are used.

RQ3: Do releases impact app performance?

There are two limitations to correlation analysis. Firstly, correlation analysis seeks an overall statistical effect in which a large number of apps and their releases participate. However, it may also be interesting, from the developers' point of view, to identify specific releases that have an atypically high impact, compared to the 'background' behaviour and characteristics of the app store as a whole; general correlation analysis is not well-suited to this more specific question. Secondly, of course, as is well-known any correlation observed does not necessarily imply the presence of a cause (correlation is not causation). Therefore, even if we were to find strong correlations, this would not, of itself, help to identify causes. This motivates the use of causal impact analysis. We apply the causal impact analysis on each target release to see if it caused a significant change in any of the app-level performance metrics R, D, N or NW defined in Section 3.4. Causal impact analysis is described in Section 6.2.

RQ3.1: What proportion of releases impact app performance? We compute the proportion of apps whose releases have affected performance, and the proportion of overall releases.

RQ3.2: How does the causal control set size affect results? Causal impact analysis requires a control set (in this case, a set of apps that have zero releases in the period studied). As the set of potential members of the control set is different in size between Google and Windows, we carry out experiments to assess how much control set differences could influence the results. Our approach is similar to the experiment using different control sets in the study by Broderson et al. [37]. We compute the causal impact analysis results for each metric for a sample of 100 target releases in the Windows dataset, using control sets of size 100, 200 and 397, the smaller of which are randomly sampled from the maximum possible set of 397 non-releasing apps.

RQ4: What characterises impactful releases?

We use the causal impact analysis results from RQ3 to analyse impactful and non-impactful releases.

RQ4.1: What are the most prevalent terms in releases? We pre-process the release text from all releases in a given store as described in Section 3.5, then identify the 'top terms' (most prevalent) for each set of releases using two methods: **TF.IDF** [179] and **Topic Modelling** [32].

We train both methods on the release text corpus, treating each instance of release text as a document. Each store is treated separately for training and evaluation, to prevent combining store-specific vocabularies. For both methods, we sum the resultant scores (probabilities) for terms (topics) over each set of releases.

The topic model is trained using 20 topics in each case. We chose 20 topics for two reasons: i) the number of documents in each corpus is between 1000 and 2000, each typically consisting of tens to hundreds of words, which is by no means a *large* corpus; ii) we wish to generalise, to avoid training topics that are relevant to certain apps or releases. The choice of 20 topics allows for generalisation, and for manual inspection each of the trained topics, without much risk of training a topic that is overly specific to an app or release.

RQ4.2: How often do top terms and topics occur in each set of releases? We compute the counts in each set of releases that contain top terms from TF.IDF and topic modelling, as identified from RQ4.1. We apply a bag-of-words model, which ignores the ordering of the words in each document. This eliminates the need to check for multiple forms of text that is discussing the same thing.

RQ4.3: What is the relative probability of each of the candidate causes? We select the sets of impactful and non-impactful releases, as well as the sets of impactful releases that positively impacted rating and those that negatively impacted rating, and compare the distributions of several developer-controlled metrics. This helps to establish probable causes that may have led to the releases being impactful, or positively affecting one of the most important performance metrics for developers: the rating.

We use the developer metrics P, Day, RT_{size} and RT_{change} in this experiment, as described in Section 7.2. We perform a Wilcoxon test [295] and Vargha and Delaney's $\hat{A}_{12}$ effect size comparison [273] between these distributions, described in more detail in Section 3.1. We also compare the distributions using box plots, which show the median and interquartile range of distributions in order that we can see if one sits higher or lower than another. We compare the distributions of release

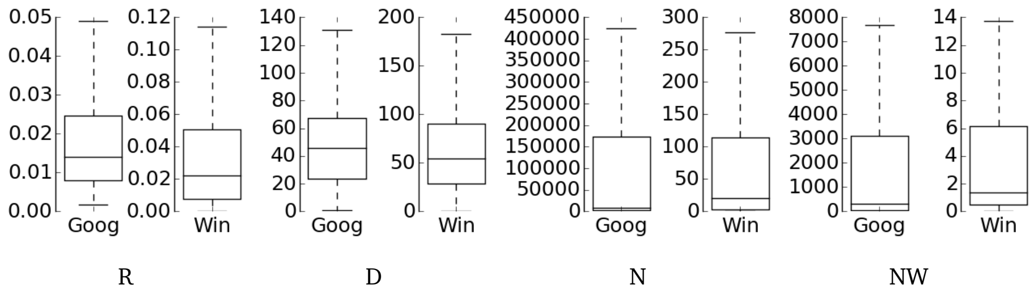


Figure 7.1: **RQ1: Standard deviation box-and-whisker plots**[182]
R: rating
D: download rank
N: number of ratings
NW: number of ratings per week
Goog: Google Play dataset
Win: Windows Phone Store dataset

days using histograms, which will enable us to see if any particular weekday or set of weekdays leads to better or worse likelihood of impactful releases.

7.3.4 Application of CausalImpact

In this section, we apply the Causal Impact Analysis method [37] using the Google `CausalImpact` framework [36].

7.3.5 Results

This subsection answers the questions posed in Section 7.3.3.

RQ1: Do app metrics change over time? The box plots in Figure 7.1 show that the metrics (R)ating, (D)ownloads, (N)umber of reviews and (NW) number of reviews per week do, indeed, change over time, because their median standard deviation is always positive.

However, Figure 7.1 reveals that not all metrics vary so greatly: the (R)ating metric exhibits the least variation (median standard deviation < 0.05 for both Google and Windows). This is a potentially useful baseline finding, because it means that a high-impact release (that does affect rating), has a chance to ‘stand out against the crowd’.

Developers are likely to care about such releases, since ratings are important, and there is some evidence that they impact upon popularity, and thereby revenue [109].

Table 7.2: **RQ2.1 and RQ2.2: Correlation results for releases**[182]
R: rating
D: download rank
N: number of ratings
Δ: change in metric from week 1 to week 52
-: correlation not significant ($p > 0.05$)

Store	**R**	**ΔR**	**D**	**ΔD**	**N**	**ΔN**
Google	-	-	-	-	-	0.13
Windows	0.20	-	-0.17	-	0.32	0.42

RQ2.1: Number of releases

Store	**R**	**ΔR**	**D**	**ΔD**	**N**	**ΔN**
Google	-	-	-	-	-0.15	-0.19
Windows	-	-	0.12	0.13	-	-

RQ2.2: Release interval

Answer to RQ1: Do app metrics change over time? The metrics (D)ownload rank, (N)umber of ratings and (NW) number of ratings per week show a high standard deviation for apps over the 52 week time period, but (R)ating shows only a small deviation.

RQ2: Do release statistics have a correlation with app performance? Having established a baseline, we now measure the correlations between the number and interval of releases in 52 weeks, and performance metrics.

RQ2.1: Does the number of releases have a high correlation with app performance? Table 7.2 presents the results of correlation analysis between release frequency and app metrics for Google and Windows app stores. We only report correlation coefficients (the rho values) that are deemed significant ($p \leq 0.05$), i.e., where there is sufficient evidence that rho $\neq 0$.

The results from our Google dataset in Table 7.2 are sparse, indicating only one significant correlation between the number of releases and ΔN, the change in the number of reviews. Even this correlation, although significant, is still weak (rho $= 0.13$) so we conclude that there is little evidence for correlation between release frequency and app metrics in the Google app store. A greater number of significant correlations was observed for the Windows app store (Table 7.2). However, the corresponding correlation coefficients (rho values) remain low, providing evidence only for a mild correlation between release frequency and the change in the number of reviews per week (ΔN).

We therefore conclude that there is no strong overall correlation between release frequency and the app metrics we collect for either app store, but there is evidence for a mild correlation between release frequency and number of reviews in 52 weeks in the Windows store.

RQ2.2: Does the median time interval between releases have a correlation with app performance? Table 7.2 presents the results of correlation analysis between release interval and app metrics for the Google and Windows app stores. As these results revealed, there is little evidence for any strong correlation between the median inter-release time period and the app metrics we collect. These findings corroborate and extend the recent findings by McIlroy et al. [191], who reported the rating was unaffected by release frequency in the Google app store. This is interesting because there is evidence that app developers release more frequently when an app is performing poorly [55]; our results indicate that this, perhaps rather desperate behaviour, is unproductive.

Answer to RQ2: Do release statistics have a correlation with app performance? Neither higher numbers of releases nor shorter release intervals correlate with changes in performance; developers who increase release frequency with the aim of improving performance may be wasting their effort.

RQ3: Do releases impact app performance? The results from RQ1 and RQ2 have established that app performance metrics do vary over releases, but that the number of releases and time intervals between releases are not important factors in determining these performance changes. This makes causal impact analysis potentially attractive to developers. With it, a developer can seek to identify the set of specific releases that had a higher performance impact, using evidence for significant changes in post-release performance compared with the set of non-releasing apps. This is the analysis to which we now turn in RQ3.

RQ3.1: What proportion of releases impact app performance? Table 7.3

Table 7.3: **RQ3.1: Causal impact analysis results**[182]
R: rating
D: download rank
N: number of ratings
NW: number of ratings per week

Details of target releases (percentages reported over 754 target releases)

Type	**Total (of target)**	**Apps**	307
Non-impactful	453 (60.1%)	**Total releases**	1,570
Impactful	301 (39.9%)	**Target releases**	754

Details of impactful releases (percentages reported over 301 impactful releases)

Metric	**Total (of impactful)**	**+ve**	**-ve**
R	152 (50.74%)	67 (22.3%)	85 (28.2%)
D	130 (43.2%)	48 (15.9%)	82 (27.2%)
N	54 (17.9%)	54 (17.9%)	0
NW	84 (29.9%)	52 (17.3%)	32 (10.6%)
R ∩ D	32 (10.6%)	9 (3.0%)	12 (4.0%)
R ∩ D ∩ NW	7 (2.3%)	2 (0.7%)	1 (0.3%)

Google

Details of target releases (percentages reported over 793 target releases)

Type	**Total (of target)**	**Apps**	726
Non-impactful	356 (44.9%)	**Total releases**	1,617
Impactful	437 (55.1%)	**Target releases**	793

Details of impactful releases (percentages reported over 437 impactful releases)

Metric	**Total (of impactful)**	**+ve**	**-ve**
R	228 (52.2%)	90 (20.6%)	138 (31.6%)
D	207 (47.4%)	93 (21.3%)	114 (26.1%)
N	267 (61.1%)	267 (61.1%)	0
NW	48 (11.0%)	27 (6.2%)	21 (4.8%)
R ∩ D	90 (20.6%)	15 (3.4%)	36 (8.2%)
R ∩ D ∩ NW	11 (2.5%)	2 (0.74%)	5 (1.1%)

Windows

Table 7.4: **RQ3.2: Impactful releases using different control sets**[182]
R: rating
D: download rank
N: number of ratings
NW: number of ratings per week

	Control Set		
Result	100	200	397
R results	45	43	39
D results	40	42	41
N results	36	40	40
NW results	13	13	14

presents overall summary statistics for the results of causal impact analysis. The row labelled 'Apps' indicates the number of apps summarised in each of the two sub tables (307 Google apps and 726 apps for Windows). This is the total number of apps which remain consistently popular over all 52 weeks studied. The total number of releases reports the number of app releases over the 52 weeks studied, while the 'target releases' denotes the subset of releases for which there is sufficient prior and posterior information available to support causal impact analysis, in terms of counterfactual posterior predictions, based on prior observations. Those releases that occur near the beginning or end of the time period will therefore not have sufficient information available, and so the causal impact cannot be studied; hence we select a subset of releases that must also belong in the range of weeks [4, 49], out of a possible [1, 52]. Of these target releases, some are impactful and some are not according to causal impact analysis. As Table 7.3 reveals, 39.9% of the target releases in the Google store and 55.1% in the Windows store are impactful.

The remainder of Table 7.3 reports the observed change in performance metrics for impactful releases, thereby identifying candidate causes of these impacts. For each performance metric change, we report the total number of releases that exhibited an impactful change in the associated metric and the percentage (of all app releases) that exhibited the change. We further subdivide this total into those which are considered positive and those which are considered negative from the developers' perspective.

From the 39.9% of impactful Google releases, approximately a third (31.9%) impacted more than one performance metric. The releases of most potential interest to developers are those that impact rating and download rank (since these are most closely coupled to revenue), of which there were 32 impactful releases.

In the Windows dataset there were a higher proportion of impactful releases: 55.1% were impactful in some performance metric, and of these impactful releases, approximately half (49.7%) had an impact on multiple metrics. There were 90 releases in the Windows dataset that impacted rating and download rank, and 11 that impacted rating, download rank and number of ratings per week.

These results support the hypothesis that there is a subset of releases that cause significant changes to their app's performance in the store.

RQ3.2: How does the causal control set size affect results? Table 7.4 reports the effect of choosing different control set sizes, from among those apps which did not undergo any releases during the time period studied. The Table 7.4 results show that very similar findings are observed for impactful releases (with respect to each performance metric), irrespective of the choice of control set.

Answer to RQ3: Do releases impact app performance? There is strong evidence ($p \leq 0.01$) that approximately 40% of the target releases in the Google Play store significantly affected a performance metric, and approximately 55% of target releases in the Windows store significantly affected a performance metric, perhaps indicating greater maturity in the Google Play store (and thus greater difficulty in having an "impact").

RQ4: What characterises impactful releases? The finding from RQ3 indicates that there are impactful releases in both app stores, but it cannot identify the causes, merely that there has been an impact in post-release performance. We now turn to analyse candidate causes, and investigate the relative probability that each has played a significant role in causing the impacts observed.

Table 7.5: **RQ4.1: Top release text terms** TF.IDF terms are on the left and Topic Modelling topics are on the right[182]
R: rating
D: download rank
N: number of ratings
NW: number of ratings per week

Type	**Overall**		**Apps**	307		
Non-impactful	new fix bug	fix bug fixed	**Target releases**	754		
Impactful	fix bug new	fix bug fixed	**Release text**	641		
Metric	**All**		**+ve**		**-ve**	
R	fix bug new	sky device channel	sky fix bug	sky device channel	fix bug new	app account use
D	fix bug new	new game experience	new game security	character new power	fix bug new	new game experience
N	new fix bug	sky device channel	new fix bug	sky device channel		
NW	new fix improvement	photo filter add	new performance improvement	app music live	fix bug improvement	photo filter add
R ∩ D	fix bug sky	app account use	sky fruit carp	sky device channel	bug fix pay	app account use
R ∩ D ∩ NW	various klingon improvement	android song facebook	security traveler dim	app purchase flight	song piano smule	android song facebook

Google

Type	**Overall**		**Apps**	726		
Non-impactful	video photo new	message chat free	**Target releases**	793		
Impactful	video app new	file medium server	**Release text**	546		
Metric	**All**		**+ve**		**-ve**	
R	video app phone	added app sound	video music phone	message chat free	app video photo	added app sound
D	video app phone	app live new	video new app	app live new	video phone app	message chat free
N	video app phone	file medium server	video app phone	file medium server		
NW	app video dating	app apps music	dating video app	added app sound	weather shazam app	app apps music
R ∩ D	video app phone	message chat free	video music task	video fix youtube	channel app phone	app also account
R ∩ D ∩ NW	hike dating event	message chat free	dating pof free	message chat free	hike learn learning	message chat free

Windows

Table 7.6: **RQ4.2: Occurrences of terms**[182]
R: rating
D: download rank
N: number of ratings
NW: number of ratings per week

Type	**Overall**	**Apps**	307
Non-impactful	145 / 379 (38.3%)	**Target releases**	754
Impactful	87 / 262 (33.2%)	**Release text**	641
Metric	**Total**	**+ve**	**-ve**
R	41 / 133 (30.8%)	16 / 56 (28.6%)	25 / 77 (32.5%)
D	42 / 110 (38.2%)	10 / 36 (27.8%)	32 / 74 (43.2%)
N	17 / 46 (37.0%)	17 / 46 (37.0%)	0 / 0
NW	22 / 77 (28.6%)	13 / 48 (27.1%)	9 / 29 (31.0%)
R ∩ D	12 / 29 (41.4%)	2 / 7 (28.6%)	4 / 12 (33.3%)
R ∩ D ∩ NW	3 / 7 (42.9%)	1 / 2 (50.0%)	0 / 1

Google (bug, fix)

Type	**Overall**	**Apps**	726
Non-impactful	118 / 248 (47.6%)	**Target releases**	793
Impactful	130 / 298 (43.6%)	**Release text**	546
Metric	**Total**	**+ve**	**-ve**
R	62 / 152 (40.8%)	22 / 58 (37.9%)	40 / 94 (42.6%)
D	50 / 137 (36.5%)	22 / 71 (31.0%)	28 / 66 (42.4%)
N	77 / 183 (42.1%)	77 / 183 (42.1%)	0 / 0
NW	14 / 33 (42.4%)	8 / 18 (44.4%)	6 / 15 (40.0%)
R ∩ D	18 / 57 (31.6%)	2 / 9 (22.2%)	9 / 18 (50.0%)
R ∩ D ∩ NW	1 / 8 (12.5%)	0 / 1	1 / 3 (33.3%)

Windows (bug, fix)

Type	**Overall**	**Apps**	307
Non-impactful	92 / 379 (24.3%)	**Target releases**	754
Impactful	72 / 262 (27.5%)	**Release text**	641
Metric	**Total**	**+ve**	**-ve**
R	36 / 133 (27.1%)	17 / 56 (30.4%)	19 / 77 (24.7%)
D	34 / 110 (30.9%)	13 / 36 (36.1%)	21 / 74 (28.4%)
N	12 / 46 (26.1%)	12 / 46 (26.1%)	0 / 0
NW	21 / 77 (27.3%)	14 / 48 (29.2%)	7 / 29 (24.1%)
R ∩ D	10 / 29 (34.5%)	4 / 7 (57.1%)	2 / 12 (16.7%)
R ∩ D ∩ NW	3 / 7 (42.9%)	1 / 2 (50.0%)	0 / 1

Google (new, feature)

Type	**Overall**	**Apps**	726
Non-impactful	133 / 248 (53.6%)	**Target releases**	793
Impactful	145 / 298 (48.7%)	**Release text**	546
Metric	**Total**	**+ve**	**-ve**
R	69 / 152 (45.4%)	32 / 58 (55.2%)	37 / 94 (39.4%)
D	68 / 137 (49.6%)	39 / 71 (54.9%)	29 / 66 (43.9%)
N	95 / 183 (51.9%)	95 / 183 (51.9%)	0 / 0
NW	16 / 33 (48.5%)	8 / 18 (44.4%)	8 / 15 (53.3%)
R ∩ D	27 / 57 (47.4%)	6 / 9 (66.7%)	7 / 18 (38.9%)
R ∩ D ∩ NW	4 / 8 (50.0%)	1 / 1 (100.0%)	2 / 3 (66.7%)

Windows (new, feature)

RQ4.1: What are the most prevalent terms in releases? Table 7.5 reports the results of information retrieval, using TF.IDF and the topic modelling on the release text of impactful releases. In this table, we consider only those apps for which release text is available. For Google, of the 754 target releases (those with sufficient evidence for causal impact analysis), 641 have release text available. For Windows, 546 of the 793 target releases have available release text. The remainder of the table consists of four overall columns, giving the metric for which a release is found to be impactful (leftmost column), followed by the most prevalent terms (for TF.IDF) and topics (for topic modelling), followed by a subdivision of these prevalent terms into those whose impacts are positive and negative from the developers' perspective. We restrict topics to only the top three terms in the table, in order to give an overview of each topic in a concise manner.

Table 7.5 reveals that terms and topics themed around bug fixes occur frequently in the Google dataset, while in the Windows dataset, the topics appear to be more closely associated with features (message chat free, new search feature). The Windows app store is comparatively more recent than the Google app store, and it is tempting to speculate that, at this comparatively immature stage, perhaps users are more concerned with new features than bug fixes. Further research would be required to investigate this possibility. Nevertheless, these observations motivate our analysis in RQ4.2 for the tuples (bug, fix) and (new, feature).

RQ4.2: How often do top terms and topics occur in each set of releases? Table 7.6 shows the number of occurrences, within impactful and non-impactful releases, of the tuples (bug, fix) and (new, feature) highlighted by information retrieval in RQ4.1. The results in Table 7.6 show that the terms (bug, fix) are more common in the non-impactful releases in both stores. However, it is more striking that for the metrics (R)ating and (D)ownload, the terms occur more often in the releases that negatively impact rating, and negatively impact popularity (having a positive impact on (D)ownloads column). We conclude, therefore, that release text mentioning bug fixes occurs more frequently in releases that negatively impact metrics such as rating and download rank.

The results show that there are proportionally more mentions of (new, feature) in releases that positively impact (R)ating and popularity for both Google and Win-

Table 7.7: **RQ4.3: Probabilistic analysis of candidate contributions**[182]
P: price
Day: Day of release (i.e. Monday, Tuesday)
RT$_{size}$: size of release text in words
RT$_{change}$: change in RT$_{size}$ on the week of release
+ve R: releases increase rating
-ve R: releases decreased rating

Impactful / Non-impactful			**+ve R / -ve R**		**Impactful / Non-impactful**			**+ve R / -ve R**	
Metric	**Wilcoxon**	$\hat{A}_{12}$	**Wilcoxon**	$\hat{A}_{12}$	**Metric**	**Wilcoxon**	$\hat{A}_{12}$	**Wilcoxon**	$\hat{A}_{12}$
P	0.006	0.546	0.045	0.570	**P**	0.121	0.480	0.073	0.546
Day	0.132	0.476	0.444	0.493	**Day**	0.158	0.520	0.439	0.506
RT$_{size}$	0.295	0.488	0.054	0.576	**RT**$_{size}$	0.359	0.493	0.279	0.523
RT$_{change}$	0.224	0.484	0.155	0.548	**RT**$_{change}$	0.241	0.486	0.249	0.526
Google					Windows				

dows. There were (proportionally) more impactful releases in the Google store that mentioned (new, feature), but fewer for Windows.

These results lead to the conclusions that releases are more likely to positively impact rating or popularity if they claim to introduce new features, and more likely to negatively impact rating or popularity if they claim to fix bugs. While the former finding is to be expected, it seems a little unfair on developers that bug fix claims might reduce performance. Future work might further investigate this effect to see whether bug fix claims are unsubstantiated, thereby providing a potential explanation.

RQ4.3: What is the relative probability of each of the candidate causes? To better understand the differences between candidate causes of impacts, we use non-parametric inferential statistics to investigate the differences between the metrics we collect for releases, depending on whether they are identified as impactful or not, and to assess the relative probability that each of these candidate causes plays a role in the impacts observed. We measure aspects of the release that lie within the control of the developer: price, day of release, the size of release text and the change in size of release text on release week.

The results in Table 7.7 show that the most probable candidate causes in each case are price and the size of the release text. Figure 7.2 presents box plots showing the distribution of price and release text size, comparing impactful releases against non-impactful releases, and releases that positively impact rating against those that

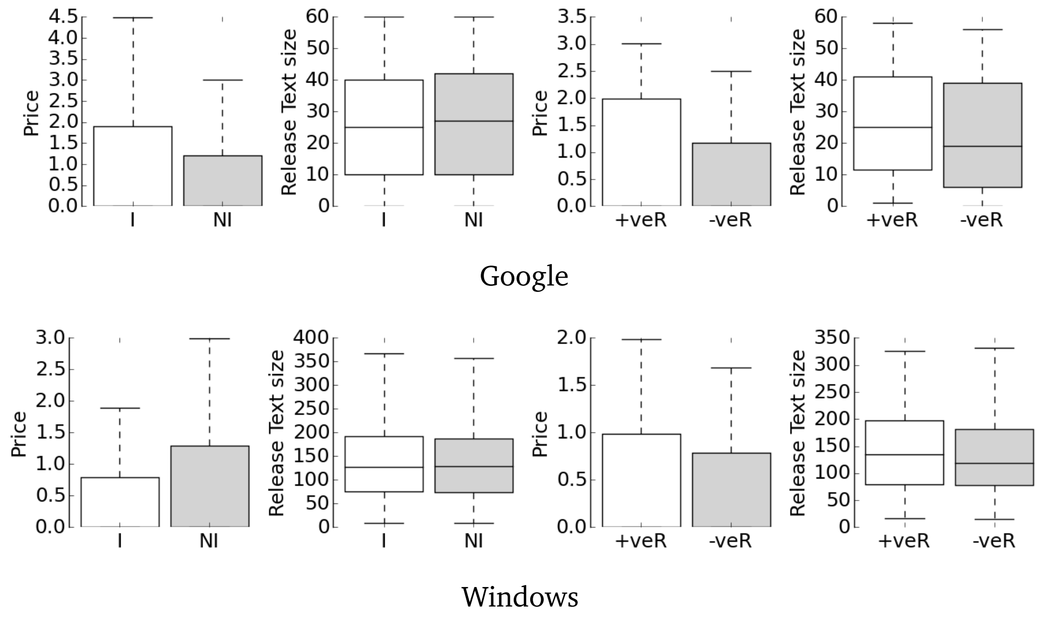
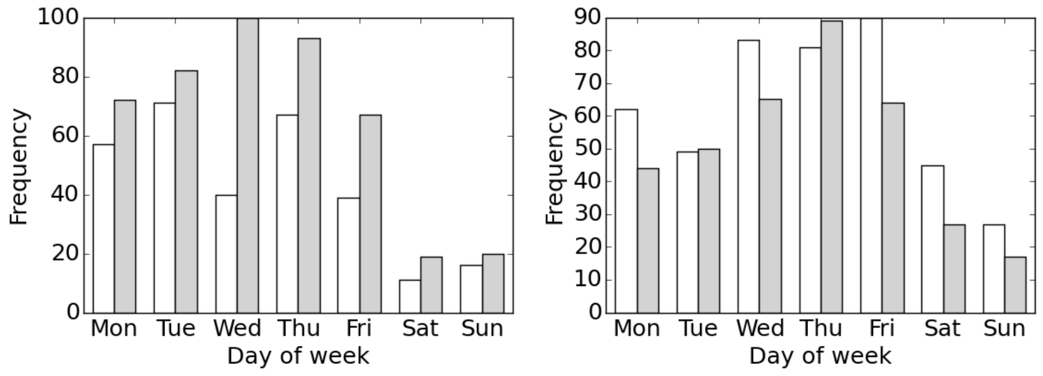


Figure 7.2: **RQ4.3: Box plots of Price and Release Text**[182]
I: impactful releases
NI: non-impactful releases
+ve R: releases increase rating
-ve R: releases decreased rating

negatively impact rating. Since half of the apps are free, the median prices are 0 and so the box plots only show the upper quartile; we therefore computed the mean prices of impactful releases and non-impactful releases: impactful releases had a mean price of £0.99 and non-impactful releases had a mean of £0.65 in the Google store; £0.47 compared with £0.54 in the Windows store. The results show that higher priced releases are more likely to be impactful in the Google store, but conversely, lower priced Windows releases are more likely to be impactful, highlighting interesting differences between the stores.

Unsurprisingly, these results confirm the intuition that users can be expected to be price-sensitive (in both stores). What is surprising and potentially interesting for developers is that the releases of higher-priced apps are more likely to have positive impacts on rating. We computed the mean prices for impactful releases that positively or negatively affected the rating: £1.03 and £0.78, respectively, for Google and £0.57 and £0.38, respectively, for Windows.

Of course, the price difference of £0.25 in Google Play appears relatively trivial. However, the difference in revenue that accrues can be substantial. We conservatively calculate that for the app, OfficeSuite Pro, over its (minimum) 1 million



Google Windows

Figure 7.3: **RQ4.3: Histograms showing the days of release**[182]
Clear: impactful releases
Shaded: non-impactful releases

installs [94], the price difference of £0.25 extrapolates to (minimum) £250,000. Our results revealed that this app had a Google Play release on 9th Sept 2014, for which we observed a significant impact on the subsequent rating. This calculation is particularly conservative, since at the time of this release, OfficeSuite Pro was priced at £8.90. Therefore, this suggests that developers need not fear a price 'race to the bottom' with competitors.

The results in Table 7.7 and Figure 7.3 also show that releases with larger amounts of release text are more likely to positively impact the rating of an app. The median number of words in positively impactful release text is 25, compared to only 19 for negatively impactful release text in Google Play, and 135 compared to 119 in Windows Phone. This provides evidence that users do read and respond to release text. The finding also suggests that developers might wish to spend time carefully choosing their release text to maximise positive influences on the users.

The reader may be surprised to learn that app developers need to choose very carefully the day of the week on which they choose to release their app [113]. This is one of the (perhaps) surprising observations that underscores the peculiarities of software engineering for app stores compared to traditional software development release practices. The potentially high release frequency, and immediacy of the app store ecosystem, migrates software engineers into a new world with quite different concerns from those with which they may be familiar.

Figure 7.3 presents histograms showing the frequency distribution of impactful and non-impactful releases over the day of release, for Google and Windows. These results extend (to Google Play and Windows Phone) the previous finding of Henze and Boll [113], that release activity is lowest on a Sunday in the Apple App Store. However, we find that, while Windows developers may benefit the most from weekend releases, Google Play developers are more likely to have impact early in the week on Monday and Tuesday. Therefore, should a developer seek to coordinate releases in multiple app stores, the evidence suggests that it would be advisable to release between Saturday and Tuesday.

Answer to RQ4: What characterises impactful releases? There is evidence that releases with higher prices and more descriptive release text could be more likely to positively impact rating if they are impactful, and evidence that releases from Saturday to Tuesday are more likely to be impactful. There is also evidence that, for both stores, the release text of releases that positively impact performance make fewer mentions of bug fixes and focus more on new features.

7.3.6 Threats to Validity

Internal validity: Our dataset is subject to the App Sampling Problem [183]. Performance data is available only for the most popular apps in each of the app stores studied. We restrict claims about findings to those that apply specifically to the most consistently popular apps over the 52 week period studied, and thereby do not suffer from the App Sampling Problem in our findings. However, any attempt to extend and generalise the findings to other apps, would be vulnerable to the App Sampling Problem, and so great care is required due to this potential threat to validity.

External validity: Naturally, care is required when extending these findings to other app stores. Indeed, this study on the Google and Windows stores shows differences between the two, as higher priced releases in Google are more likely to be impactful, whilst the same is true of lower priced releases in Windows. There are many common findings, but this difference highlights the fact that our results may not apply to other app stores. Nevertheless, the methods we used to analyse causal impacts can be applied to other app stores.

Construct validity: Our construct validity may be affected by our selection of price, day of week and release content as potential causes for impact releases. We test for significant differences between impactful and non-impactful releases for these metrics, but we may be missing other potential causes. It may be that impactful releases have other properties that we have missed, which we leave as as future work to explore. Additionally, we ask developers if they know of external causes in Section 7.4.

It may also be that developers update an app without updating its documentation, and thus any feature changes are not detected in the release text of the app. This threat is mitigated by comparing only apps with changed text.

Conclusion validity: Our conclusion validity could be affected by the qualitative human assessment of 'top terms' and topics for sets of releases in RQ4.1. We mitigate against this threat to validity by asking a quantitative question of the number of times (bug, fix) and (new, feature) occur in each set of releases in RQ4.2.

7.3.7 Conclusions

Our analysis of the Google Play and Windows Phone app stores, over a period of 52 weeks from July 2014 to July 2015 has found that overall release frequency is not correlated with subsequent app performance, but that there is evidence that price, release text size and content and day of release all play a role in whether a release is impactful and the type of impact it has. Higher priced releases are more likely to be impactful and, perhaps surprisingly, to *positively* impact rating; releases launched between Saturday and Tuesday (and therefore not mid-week) are more likely to be impactful; releases with text mentioning new features instead of bug fixes are more likely to be impactful and to positively impact rating, and releases with longer (more descriptive) text are also more likely to positively impact rating.

7.4 Causal Impact Analysis for App Releases in Google Play

This work was published in FSE 2016 Proceeding [185]. The first author's contribution to this paper was to formulate the idea, implement and execute the experimentation and collect the results and analyse them; other authors of the paper contributed to research question formulation, result analysis and narrative write up.

App developers would like to understand the impact of their own and their competitors' software releases. In Section 7.3, we found that causal impact analysis is useful for identifying impactful releases for further analysis, using a dataset of the most popular 1,033 apps mined from Google Play and Windows Phone Store. In order to investigate whether findings extend to the wider app store and less popular apps, we mined 38,858 popular Google Play apps, over a period of 12 months. The extension to this dataset raises computation time exponentially, due to the larger control set size. To address this, we use our Causal Impact Release Analysis tool, CIRA, that implements causal impact analysis for app stores. For these apps, we identified 26,339 releases for which there was adequate prior and posterior time series data to facilitate causal impact analysis. We found that 33% of these releases caused a statistically significant change in user ratings. We use the approach to reveal important characteristics that distinguish causal significance in Google Play. To explore the actionability of causal impact analysis, we elicited the opinions of app developers: 52 companies responded, 78% concurred with the causal assessment, of which 33% claimed that their company would consider changing its app release strategy as a result of our findings.

7.4.1 Findings

The findings from this study are as follows:

1. The causal impact analysis tool, `CIRA`[1], is useful for performing causal impact analysis on app store data, where dataset size may raise computation time exponentially for other tools.

2. We contacted developers of impactful releases: 52 developers responded, 78% of whom agree with `CIRA`'s assessment, of which 33% claimed that their company

[1]Available at `http://www0.cs.ucl.ac.uk/staff/w.martin/cira`

would consider changing their release strategy.

3. We study Google Play app releases using `CIRA`, finding:

3.1. Paid app releases have a greater chance of affecting subsequent user ratings (a chance of 40% for paid apps, compared with 31% for free apps).

3.2. Paid apps with releases that had impactful positive effects have higher prices.

3.3. Free apps with impactful releases have a greater chance for their effects to be positive (37% for paid apps compared with 59% for free apps).

3.4. Releases that positively affected user ratings had more mentions of bug fixes and new features.

3.5. Releases that affected subsequent user ratings were more descriptive of changes.

7.4.2 Data

We mined app data from Google Play between February 2015 and February 2016, as detailed in Section 3.2.

Full set: All apps mined in the time period. Some apps drop out of the store (for unknown reasons), but they are included in the full set for the duration in which they appear in the store. This full set consists of 38,858 apps.

Control set: The set of apps that have no new releases over the studied time period. We refer to this as the control set, as it is the benchmark by which one can measure changes in the releasing apps. Apps that drop out of the store are not included in the control set because consistency is required for a reliable control set. This control set consists of 680 apps.

Target set: The set of app releases that occur in the studied time period and occur at least 3 weeks after the previous releases, and at least 3 weeks before the next release. The target releases have some longevity, which suggests that they are more than 'hotfixes' for recently introduced bugs. They also have a non-trivial window of data on either side, which makes it possible to observe any effect the release may have had on the app's performance. This ensures sufficient data availability to perform causal impact analysis. Apps that drop out of the store are included in the target set if they include adequate prior and posterior information as defined above. This target set consists of 14,592 apps and 26,339 releases.

7.4.3 Research Questions

This section explains the questions posed in this study and how we approach answering them.

RQ1: Do app metrics change over time?

We establish a baseline by establishing whether app success metrics change over time, by computing the standard deviation over 52 weeks, of the metrics R, N, NW and L as defined in Section 3.4. These distributions are drawn using box plots. We determine whether app success changes more or less for those apps that have releases, by comparing plots for those apps that have no releases (the control set), with releasing apps (the target set). If the metrics for the target set change over time more than those in the control set, this motivates further analysis of releases that could *cause* the observed changes. It is expected that releasing apps would show greater deviation in description length, describing changes and features.

RQ2: Do release statistics have a correlation with app performance?

We build on this analysis by measuring whether app success is correlated with the number of app releases in the time period studied, and whether it is correlated with the time interval between releases. This will show whether a large (or conversely, small) number of releases might lead to increased success, and likewise for release interval.

For both of these experiments, only apps in the target dataset are used. We perform correlation analysis between the number of releases of each app and their value for the metrics R and N at the end of the time period. The metric NW is not used because this number is set on a per-week basis, but instead use the change in R and N from the first snapshot to the last, denoted ΔR and ΔN respectively. Additionally, we do not use L because this does not represent app success.

RQ2.1: Does the number of releases have a high correlation with app performance? We perform correlation analysis between the number of releases in the studied time period and the metrics R, ΔR, N and ΔN at the end of the time period.

RQ2.2: Does the median time interval between releases have a correlation with app performance? We perform correlation analysis between the median interval between releases of each app and the metrics used in RQ2.1.

RQ3: Do releases impact app performance?

Using CIRA to perform causal impact analysis, we identify the impactful releases as defined in Section 6.1.1, and experiment with different control set sizes to assess the effect this may have on results.

RQ3.1: What proportion of releases impact app performance? We compute the proportion of apps whose releases have affected subsequent success. We group results under the metrics affected: R, N, NW and the intersection of R and NW, overall and for positive or negative changes.

RQ3.2: How does the causal control set size affect results? Causal impact analysis uses a control set: a set of unaffected data vectors, which in this case is the set of apps that have zero releases in the period studied. As the set is used in two different ways ('spike' and 'slab' [249], as explained in Section 6.1.1), we test whether the set size makes a difference that could influence results. Our approach is similar to the experiment using different control sets in the study by Brodersen et al. [37]. We compute the causal impact analysis results for each metric with the full target dataset, using repeatedly halved control sets of size 340, 170, 85, 42, 21, 10, 5, and 3 which are each randomly sampled from the maximal set of 680 non-releasing apps.

We compute the agreement between each set of results and the results used to answer RQ3.1. Agreement is defined as $\left(\frac{YY+NN}{total}\right)$, where YY indicates a significant change as detected on both datasets, NN indicates no significant change detected on both datasets, and total is a count of all results (including disagreements). We also compute the Cohen's Kappa [54], which takes into account the chance for random agreement and is bounded above by the agreement. We compute a second set of results using 680 control vectors, which will show the expected difference between consecutive runs with the same control set, since there is a random component in the predictive model.

RQ4: What characterises impactful releases?

We use the causal impact analysis results from RQ3 to identify the differences between impactful and non-impactful releases.

RQ4.1: What are the most prevalent terms in releases? We identify the most prevalent terms for each set of releases using **TF.IDF** [179] and **Topic Mod-**

elling [32]. Both methods are trained on the release text corpus, treating each instance of release text as a document. For both methods, we sum the resultant scores (probabilities) for terms (topics) over each set of releases.

The topic model is trained using 100 topics. We chose 100 topics for three reasons: i) the number of documents in the corpus (updated release text from target releases), each consisting of tens to hundreds of words, which is a large sized corpus; ii) we do not wish to over-generalise, nor over-fit; 100 topics will allow diversity without assigning the same topic to every document with a release; iii) 100 topics is a common selection for non-trivial datasets, serving as the default setting in `GibbsLDA++`[224] and `JGibbLDA`[225]. Intuitively, the choice of 100 topics allows for diversity in the trained topics, without unduly elevating the risk of training a topic that is overly specific to an app or release.

RQ4.2: How often do top terms and topics occur in each set of releases? We compute the counts in each set of releases that contain top terms that emerge from TF.IDF and topic modelling, as performed in RQ4.1. We apply a bag-of-words model, which ignores the ordering of the words in each document. This eliminates the need to check for multiple forms of text that discuss the same topic.

RQ4.3: What are the effects of each of the candidate causes? We select the sets of statistically impactful and non-impactful releases, as well as the sets of impactful releases that increased and decreased rating, and compare the distributions of several developer-controlled properties. This will help to establish potential causes that may have led to the releases being impactful, or positively affecting an important success metric for developers: the rating.

We consider properties of the release that lie within the control of the developer: (P)rice, (RT_{size}) the size of release text (defined in Section 7.2) in words, and (RT_{change}) the change in RT_{size} on the week of release. For each of these properties, we use the Wilcoxon test and Vargha and Delaney's $\hat{A}_{12}$ effect size comparison [273], to identify statistically impactful differences between causally impactful releases and non-impactful releases. We use the untransformed Vargha and Delaney's comparison [208] because we are only interested in the raw probability value it produces. In case a statistical difference is found for a given aspect, we use box plots to understand if this property may play a role in the changes observed (i.e. it may be a candidate cause).

At this point, we will have identified a subset of releases that exhibit statistically significant success effects, and we will have identified further differences between sets of impactful and non-impactful releases, as well as the differences between positive and negative impactful releases. To further explore the actionability potential of the tool and our findings, we ask the following research question.

RQ5: How useful is causal impact analysis to developers?

Because there is no ground truth, only different implications of any discovered statistical significance, we can only answer this question semi-quantitatively. To determine whether our results are useful we simply ask the developers of causally impactful releases, as determined by our tool, `CIRA`, whether they agree with the classification. We email developers via the email addresses contained on their app store pages, informing them of the impactful release and proposing to send a report detailing the tool's findings. We expect a large proportion of these emails may fail to reach a human, and can only confirm contact is established if we hear back from a developer. Once contact is established with the app's developers, we ask them the following questions[2]:

Agree with detected significance: We ask if the developers of the app agree with `CIRA`'s assessment that the release was impactful.

External cause of changes: We ask if the company is aware of an external event which may have caused the detected significant change, such as advertising campaigns.

Would change strategy: We ask if the company would consider changing their release strategy based on findings.

Receiving further reports: We ask if the company is interested in receiving further reports for their app releases.

Learning contributing factors: We ask if the company is interested in learning more about the characteristics of impactful releases, from the results of our study.

[2]The full questionnaire is included in Appendix E

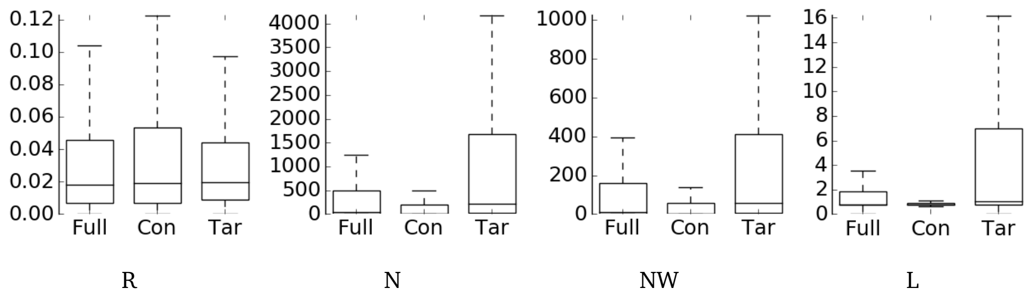


Figure 7.4: **RQ1: Standard deviation box plots**[185]
R: rating
N: number of ratings
NW: number of ratings in the last week
L: length of description in words
Full: complete set of apps and releases
Con: just control set of non-releasing apps
Tar: target set of apps and releases

7.4.4 Application of CIRA

`CIRA` is used to apply causal impact analysis in this study. The high number of apps in the control set resulted in an apparent exponentially long execution time when running `CausalImpact`. This is likely due to the selection of control vectors to use as spike regressors. However, the execution time `CIRA` is affected in O(n) time.

7.4.5 Results

This section answers the questions posed in Section 7.3.3.

RQ1: Do app metrics change over time? The box plots in Figure 7.4 show that the metrics (R)ating, (N)umber of ratings, (NW) number of ratings per week and (L)ength of description do change over time, because their median standard deviation is always positive. The deviation in number of ratings and number of ratings per week is high, but very low for rating. This is because, for apps with many ratings, even a small change corresponds to thousands of users rating higher or lower than the established mean.

The deviation is approximately even between the control and target datasets for rating deviation; this is a surprising result, and indicates that either a) ratings are unstable (because they change) even for stable, established, non-releasing apps, or b) app releases have little effect over all, globally detectable, ratings. The finding of a low deviation in the target set means that a causally significant release (that affects rating), has a good chance to 'stand out from the crowd'.

Table 7.8: **RQ2: Correlations between release statistics**[185]
R: rating
N: number of ratings
Δ: change in metric from week 1 to week 52

Release statistic	**R**	**ΔR**	**N**	**ΔN**
Quantity	0.13	0.09	0.27	0.30
Median interval	-0.12	-0.06	-0.19	-0.21

The box plots show that standard deviations in number of ratings and rating frequency length are significantly higher for the target releasing dataset, than for all apps in the dataset and for the control set. This is expected, and shows some utility to app releases: to increase user activity in downloading and rating the apps, and perhaps to increase the user base.

The deviation in description length is higher for the target dataset as expected, suggesting that descriptions are updated to provide information about releases. This finding supports the intuition that descriptions may be used as a channel of communication between developers and users.

Answer to RQ1: Do app metrics change over time? The metrics (N)umber of ratings and (NW) number of ratings per week show a high standard deviation for apps between February 2015 and February 2016, but (R)ating shows only a small deviation. The deviation in user rating frequency is higher for the target releasing dataset, suggesting that app releases lead to user activity.

RQ2: Do release statistics have a correlation with app performance? We now measure the correlations between the number and interval of releases and success metrics.

RQ2.1: Does the number of releases have a high correlation with app performance? Table 7.8 presents the results of correlation analysis between release quantity and median interval, and app metrics for the target dataset. We report correlation coefficients (rho values) that are deemed significant ($p \leq 0.05$).

The results in Table 7.8, indicate only weak significant correlations between the success metrics and their change over the time period studied. The strongest correlation, for ΔN where rho = 0.30, is still too weak to definitely suggest a strong relationship. We therefore conclude that there is no strong overall correlation between release frequency and the app metrics we collect, but there is evidence for a weak correlation between number of releases and number of reviews accrued over a year.

RQ2.2: Does the median time interval between releases have a correlation with app performance? Table 7.8 presents the results of correlation analysis between release interval and app metrics. As these results reveal, there is little evidence for any strong correlation between the median inter-release time period and the app metrics we collect. These findings corroborate and extend the recent findings by McIlroy et al. [191], who reported the rating was unaffected by release frequency in the Google app store. This is interesting, because there is evidence that app developers release more frequently when an app is performing poorly [55]; our results indicate that this, perhaps rather desperate behaviour, is unproductive.

Answer to RQ2: Do release statistics have a correlation with app performance? Neither higher numbers of releases nor shorter release intervals correlate strongly with changes in success.

RQ3: Do releases impact app performance? The results from RQ1 and RQ2 have established that app rating metrics do vary over releases, but that the number of releases and time intervals between releases are not important factors in determining these changes. This makes causal impact analysis potentially attractive to developers. With it, developers can seek to identify the set of specific releases that had a higher effect on success, using evidence for significant changes in post-release success compared with the set of non-releasing apps. This is the analysis to which we now turn in RQ3.

RQ3.1: What proportion of releases impact app performance? Table 7.9 presents overall summary statistics for the results of causal impact analysis. The 'Apps' row indicates the number of apps summarised as part of the target dataset, and the 'Target releases' row indicates the number of releases these apps underwent in the studied time period that were outside of a 3-week window of other releases.

Table 7.9: **RQ3.1: Causal impact analysis results**[185]
R: rating
N: number of ratings
NW: number of ratings per week
+ve: releases that increased rating
-ve: releases that decreased rating

Details of target releases (percentages reported over 26,339 target releases)

Type	**Total (of target)**	**Apps**	14,592
Non-impactful	17,639 (67.0%)	**Target releases**	26,339
Impactful	8,700 (33.0%)	**Control apps**	680

Details of impactful releases (percentages reported over 8,700 impactful releases)

Metric	**Total (of impactful)**	**+ve**	**-ve**
R	4,781 (55.0%)	2,563 (29.5%)	2,218 (25.5%)
N	4,747 (54.6%)	4,747 (54.6%)	0
NW	2,226 (25.6%)	862 (9.9%)	1,364 (15.7%)
R ∩ NW	701 (8.1%)	220 (2.5%)	199 (2.3%)

Those releases that occur near the beginning or end of the time period will therefore not have sufficient information available, and so causal impact analysis cannot be applied. Hence we select a subset of releases that must also belong in the range of weeks [4, 49], out of a possible [1, 52]. Of these target releases, some are impactful and some are not according to causal impact analysis. As Table 7.9 reveals, we found that 33.0% of releases were impactful.

The remainder of Table 7.9 reports the observed change in success metrics for impactful releases, thereby identifying candidate causes of these effects. For each success metric change, we report the total number of releases that exhibited a significant change in the associated metric and the percentage (of all app releases) that exhibited the change. We further subdivide this total into those that are considered 'positive' and 'negative' (from a developer's perspective).

From the 33.0% of impactful app releases in Google Play, approximately a third (30.0%) affected more than one success metric. The releases of most potential interest to developers are likely to be those that affect both rating and rating frequency (due to their potential for increasing user base and revenue). Of these, there were 701 impactful releases.

Table 7.10: **RQ3.2: Agreement between full control set and different set sizes**[185]
R: rating
N: number of ratings
NW: number of ratings per week

		Control Set								
	Metric	3	5	10	21	42	85	170	340	680
Agreement	R	0.93	0.93	0.93	0.91	0.91	0.91	0.93	0.93	0.94
	N	0.91	0.91	0.91	0.91	0.91	0.90	0.91	0.91	0.92
	NW	0.97	0.97	0.97	0.97	0.97	0.95	0.97	0.97	0.98
	All	0.94	0.94	0.94	0.93	0.93	0.92	0.93	0.94	0.94
Cohen's Kappa	R	0.78	0.76	0.76	0.72	0.72	0.73	0.76	0.77	0.79
	N	0.69	0.70	0.70	0.70	0.69	0.66	0.69	0.69	0.73
	NW	0.78	0.78	0.80	0.83	0.81	0.71	0.81	0.83	0.85
	All	0.75	0.74	0.75	0.74	0.73	0.70	0.74	0.76	0.78

These results support the hypothesis that there is a subset of releases that cause significant changes to their app's success in the store.

RQ3.2: How does the causal control set size affect results? Table 7.10 reports the effect of choosing different control set sizes, from among those apps that did not undergo any release during the time period studied.

The results in Table 7.10 reveal strong agreement for each control set size, indicating that the model is stable in this case (using non-releasing apps). One can see from the two runs (using the full 680 apps in the control set), that there is between 0.92 and 0.98 agreement due to the stochastic element of Causal Impact Analysis. Our results show that restricting the choice of control set does not have advantages, and so we opt to use the full 680 apps for the control set for subsequent experiments.

Answer to RQ3: Do releases impact app performance? There is strong evidence ($p \leq 0.01$) that 33% of the target releases in the Google Play store significantly affected a success metric, and approximately 11% significantly affected more than one success metric.

Table 7.11: **RQ4.1: Top release text terms** TF.IDF terms on the left and Topic Modelling topics on the right[185]
R: rating
N: number of ratings
NW: number of ratings per week

Type	**Overall**		**Apps**	14,592		
Non-impactful	feature fix time	mode challenge friend	**Target releases**	26,339		
Impactful	feature fix device	hero monster battle	**Release text**	20,014		
Metric	**All**		**+ve**		**-ve**	
R	feature word fix	hero monster battle	feature word time	wallpaper live christmas	bible fix support	account mobile card
N	feature word time	wallpaper live christmas	fix feature word	wallpaper live christmas		
NW	map feature word	tip local city	card map feature	hero monster battle	map feature time	tip local city
R ∩ NW	feature video map	wallpaper live christmas	account card feature	account mobile card	video time feature	hero monster battle

Table 7.12: **RQ4.2: Occurrences of the terms in release text**[185]

Type	**Overall**	**Apps**	14,592
Non-impactful	4,690 / 13,200 (35.5%)	**Target releases**	26,339
Impactful	2,432 / 6,809 (35.7%)	**Release text**	20,014
Metric	**Total**	**+ve**	**-ve**
R	1,336 / 3,794 (35.2%)	745 / 2013 (37.0%)	591 / 1781 (33.2%)
N	1,300 / 3,693 (35.2%)	1,300 / 3,693 (35.2%)	0 / 0
NW	626 / 1,783 (35.1%)	260 / 685 (38.0%)	366 / 1098 (33.3%)
R ∩ NW	190 / 578 (32.9%)	63 / 179 (35.2%)	48 / 170 (28.2%)

Occurrences of the terms ‘bug’ and ‘fix’ in release text.

Type	**Overall**	**Apps**	14,592
Non-impactful	2,473 / 13,200 (18.7%)	**Target releases**	26,339
Impactful	1,355 / 6,809 (19.9%)	**Release text**	20,014
Metric	**Total**	**+ve**	**-ve**
R	795 / 3,794 (21.0%)	437 / 2013 (21.7%)	358 / 1781 (20.1%)
N	687 / 3,693 (18.6%)	687 / 3,693 (18.6%)	0 / 0
NW	393 / 1,783 (22.0%)	151 / 685 (22.0%)	242 / 1098 (22.0%)
R ∩ NW	149 / 578 (25.8%)	52 / 179 (29.1%)	37 / 170 (21.8%)

Occurrences of the terms ‘new’ and ‘feature’ in release text.

RQ4: What characterises impactful releases? The finding from RQ3 tells us that there are impactful releases, but it cannot identify the causes, merely that there has been a significant change in post-release success. We now turn to identify any candidate causes, which may have played a significant role in the changes observed, and analyse their effects.

RQ4.1: What are the most prevalent terms in releases? Table 7.11 reports the results of information retrieval, using TF.IDF and the topic modelling on the release text of impactful releases. In this table, we consider only those apps for which release text is available. The results show only the top TF.IDF terms, and terms from the top topic, respectively; thereby indicating the most prevalent terms and topic.

Of the 26,339 target releases (those with sufficient evidence for causal impact analysis), 20,014 have release text available. The remainder of the table consists of four overall columns, giving the metric for which a release is found to be impactful (leftmost column), followed by the most prevalent terms (for TF.IDF) and topics (for topic modelling), followed by a subdivision of these prevalent terms into those whose effects are positive and negative (from a developer's perspective).

Table 7.11 reveals that terms and topics themed around bug fixes and features occur frequently overall in the text of impactful releases. The text of releases that positively or negatively affected rating is slightly more specific to certain sets of apps, i.e. "card map feature", "wallpaper live christmas", yet still appear to refer to features. These observations motivate our subsequent analysis in RQ4.2 for the terms 'bug' and 'fix' and 'new' and 'feature'.

RQ4.2: How often do top terms and topics occur in each set of releases? Table 7.12 shows the number of occurrences, within impactful and non-impactful releases, of the terms 'bug' and 'fix', and 'new' and 'feature' which emerged as the top terms from our information retrieval in RQ4.1.

The results in Table 7.12 show that the terms 'bug' and 'fix' are far more common than 'new' and 'feature' in both impactful and non-impactful releases. This is unsurprising because bug fixing occupies a large proportion of development effort [304]. However, it is noteworthy that, in both cases, the terms are more common in the releases that positively affected metrics, as opposed to those that negatively affected metrics.

Table 7.13: **RQ4.3: Probabilistic analysis of candidate contributions**[185]
P: price
RT_{size}: size of release text in words
RT_{change}: change in RT_{size} on the week of release
+ve R: releases increase rating
-ve R: releases decreased rating

	Impactful / Non-impactful		+ve R / -ve R	
Metric	**Wilcoxon**	$\hat{A}_{12}$	**Wilcoxon**	$\hat{A}_{12}$
P	0.000	0.539	0.000	0.419
$\mathbf{RT}_{size}$	0.000	0.532	0.006	0.479
$\mathbf{RT}_{change}$	0.110	0.505	0.434	0.499

RQ4.3: What are the effects of each of the candidate causes? Based on the low p-values reported in Table 7.13 we further analyse price and the size of the release text as candidate causes. Figure 7.5 presents box plots showing the distribution of (paid app) price and release text size, comparing impactful releases against non-impactful releases, and releases that positively affect rating against those that negatively affect rating. Since 61% of the apps are free, we plot paid apps in the price boxplot, and compute the proportion of free and paid apps in each set, as well as the mean and median prices.

The results for price are interesting, somewhat surprising, and nuanced. We found that impactful releases (positive and negative) have higher prices than non-impactful releases. The mean price of all impactful releases (including free) was £0.79, and the mean price of all non-impactful releases was £0.59. Paid releases were more likely to be impactful: 40.2% of paid app releases were impactful, compared with 30.7% of free app releases. A somewhat surprising finding is that higher priced (paid) app releases are more likely to have a positive effect: a mean of £3.25 and median of £1.72 compared with £2.45 and £1.58 for those with negative effects, respectively. However, a greater proportion of impactful paid app releases *negatively* affected rating: 62.6% compared with 41.0% for free apps.

Overall, a larger proportion of impactful releases are paid than non-impactful releases (29.6% compared with 21.7%, respectively). As a result, the mean price of impactful releases is higher by £0.20. Of course, a price difference of £0.20 in Google Play appears relatively trivial. However, the difference in revenue that accrues can be substantial.

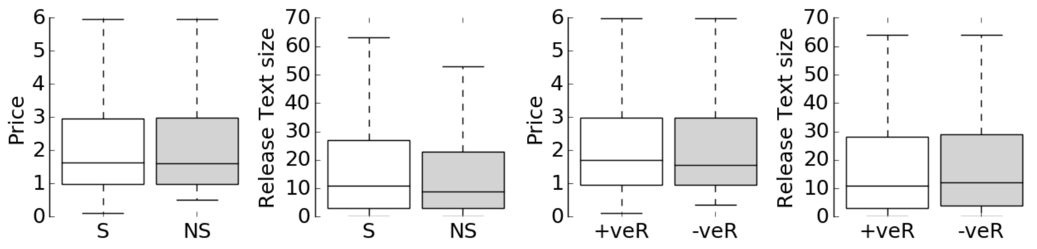


Figure 7.5: **RQ4.3: Box plots of Price and Release Text size**[185]
S: impactful releases
NS: non-impactful releases
+veR: releases that increased rating
-veR: releases that decreased rating

We conservatively calculate that for the app, OfficeSuite Pro + PDF (which had a release on 23rd December 2015, for which we observed a significant effect on the subsequent rating), over its (minimum) 50,000 installs [94], the price difference of £0.20 extrapolates to (minimum) £10,000 in accrued revenue. Our revenue calculation is particularly conservative, since at the time of this release, OfficeSuite Pro was priced at £11.66. This finding suggests that developers need not fear a 'race to the bottom' with competitors over pricing. Unsurprisingly, these results confirm the intuition that users can be expected to be price-sensitive.

The results in Table 7.13 also show that there are significant differences between the distributions of price and release text size, comparing impactful with non-impactful releases and releases that increase rating to those that decreased rating. The median number of changed, stopword-filtered words in impactful release text is 11, compared to only 9 for non-impactful release text. This provides evidence that users may be influenced by release text, but the effect size is relatively small. Nevertheless, developers might wish to spend time carefully choosing their release text to maximise positive influences on the users.

Answer to RQ4: What characterises impactful releases? There is evidence that releases that significantly affect subsequent app success have higher prices and more descriptive release text. Releases that positively affect rating are more common in free apps, and in paid apps with high prices. We also note that the release text of releases that positively affect success make more prevalent mentions of bug fixes and new features.

Table 7.14: **RQ5: Developer responses** to questionnaire[185]

	Receiving further reports	Agree with detected significance	External cause of changes	Learning contributing factors	Would change strategy
Yes	27	35	20	37	17
No	14	11	22	4	23
Total	41	46	42	41	40

RQ5: How useful is causal impact analysis to developers? We sent 4,302 emails to the email addresses available in the Google Play app store pages of companies with impactful releases, as detected by our tool, CIRA. Of course, this is something of a 'cold call', and we suspect that many emails never even reached a human. We can report that 90 immediately failed due to invalid email addresses used on app store pages, and 127 were immediately assigned to a support ticket. Those that received a response are the only cases in which we can verify that contact was established with developers, of which there were 138.

Of these 138 developers, 52 (distributed across 25 of 41 app store categories) replied to express their opinion in response to follow up questions. All respondents' apps were established in the store; the smallest had 31 reviews and the largest had 78,948 at the time of our experiments. We summarise developers' opinions in Table 7.14, in each case indicating the instances where developers expressed an opinion: the most answered question concerned agreement with CIRA's assessment, of which there were 46 respondents.

We observe that 35 out of 46 development teams who expressed an opinion, agreed with CIRA's assessment that their app release was causally impactful. For example, the developers of a dictionary application said: "in our case it was obvious for ourselves because it was a totally new release and with lots of new features", resonating with our earlier finding that new features increase the chance for impactful releases (see RQ4.3). Only 11 developers disagreed with CIRA's assessment. However, some of them were still able to identify a cause for the significant change detected by CIRA (but did not think that the release itself could be the cause). For example, the developers of a security caller app said: "We did not release anything. We just upload builds for our beta version which uses few users :-) So your tools are

WRONG." In this case, although the developers did not consider a beta version as a full release, the app's version identifier changed on its app store page resulting in a 'release', by our definition in Section 6.1. This detected release, combined with increased user activity, resulted in the causal significance detected by `CIRA`.

About one half of those who expressed an opinion (20 out of 42) indicated that they knew of an external reason for the changes, and several teams elaborated further. One might expect the developers who know of an external cause to be a subset of those who believe there to be a significant causal effect. However, 5 of 20 developers who claimed to know of external causes also disagreed with `CIRA` that the corresponding release was impactful. One set of developers described the release as: "a minor 'bugfix-only' release. Therefore I doubt that this release was the main reason for this change.". However, as shown in the RQ4 results, mentions of bug fixes in release text are more prevalent in impactful releases that positively affected rating. Indeed, this particular release did mention bug fixes in its release text, and significantly and positively affected rating.

Over half of the developers who expressed an opinion (27 of 41) were interested in receiving further reports. The majority of developers (37 of 41) indicated that they would like to learn more about the characteristics of impactful releases, and 17 of 40 indicated they would consider changing their release strategy based on our findings. Only 10 of the 17 developers who would change their strategy knew of an external cause of the changes, thus suggesting that this consideration will be based on our general findings. These results provide initial evidence that causal impact analysis can be useful to app developers.

Answer to RQ5: How useful is causal impact analysis to developers? Three quarters of developers who expressed an opinion (35 of 46) agreed with `CIRA`. Most developers (37 of 41) were keen to learn more about the characteristics of impactful releases, and 17 of 40 said that they would consider changing their release strategy based on `CIRA`'s findings. This provides initial evidence that causal impact analysis is useful to developers.

7.4.6 Threats to Validity

Internal validity: Like the study in Section 7.3, our dataset is subject to the App Sampling Problem [183]. We restrict claims about findings to those that apply specifically to the popular Google Play apps, and thereby do not suffer from the App Sampling Problem in our findings.

External validity: Care is required when extending these findings to other app samples and app stores. Nevertheless, the methods used to analyse causal effects can be applied to other app stores and datasets. Conclusions about the characteristics of impactful releases over this sample will yield interesting and actionable findings for developers, and we contact app developers for their opinions on the usefulness of our technique in RQ5.

We can only report the views of those developers with whom contact could be established (see Section 7.4.5), and care is required when interpreting their responses. Since we were able to effectively reach 52 developers, we cannot claim that our sample is representative of the entire population. However, this is still a fairly large sample with respect to those used in other studies involving app developers (e.g., [166, 206]).

Construct validity: Our construct validity may be affected by our study of empirical app metrics and release text topics, as properties of impactful releases. To mitigate this threat, we ask developers if they know of external causes of the observed impacts in RQ5.

As with Section 7.3, it may be that developers update an app without updating its documentation, and thus feature changes are not detected in the release text of the app. This threat is mitigated by comparing only apps with changed text for experiments comparing changed features.

Conclusion validity: The qualitative assessment of 'top terms' and topics in RQ4.1 could affect conclusion validity of this study. This is mitigated in RQ4.2 by asking a quantitative question of the number of times 'bug' and 'fix' and 'new' and 'feature' occur in each release set.

7.4.7 Conclusions

In this work we propose the use of Casual Impact Analysis to identify causally significant releases in app stores. In particular, we introduce our tool `CIRA` to perform causal impact analysis on 26,339 Google Play app releases between February 2015 and February 2016, for all apps that appeared at least once in the top rated apps in the year prior to February 2015. For these apps, we found that overall release frequency is not correlated with subsequent app success, but that there is evidence that price and release text size and content all play a role in whether a release is impactful and the type of effect it has on success. Higher priced releases are more likely to be impactful and, perhaps surprisingly, to *positively* affect rating; there were more prevalent mentions of new features and bug fixes in releases that positively affected rating, and impactful releases also had longer (likely more descriptive) release text. We have shown that causal analysis can be a useful tool for app developers by eliciting their opinions: most of those who responded were interested in the findings and agreed with CIRA's assessment, and some said they would consider changing their releasing strategy based on our findings.

7.5 What are the Features that are Added to impactful Releases?

Knowing the releases that are significantly impactful is only part of the story for developers; they want to know the features that can be added to their apps to achieve impact. Using topic modelling, we identify the features that have changed in app release text at the time of releases, comparing changes between impactful releases and non-impactful releases. We identify the features that are added in impactful releases and not added in non-impactful releases, and therefore might serve as candidate features to implement in order to stimulate user download or rating activity. Conversely, we also identify the features added in non-impactful releases that are not added in impactful releases, and so may not lead to impact. We conduct this comparison between positive and negative impactful releases, in order to identify the features that developers could add to their apps in order to stimulate a positive user reaction. Using our identified set of 6,776 changed, impactful releases, of which 3,750 are positive and 3,026 are negative, and our set of 17,639 changed, non-impactful releases, we identify the features that developers need to know about, both overall and at category granularity.

7.5.1 Introduction

As is well known, correlation does not necessarily imply causation. Therefore, when an app happens to have a very impactful release that also adds a new feature, it does not necessarily follow that the topic has caused this impact. A multitude of other factors, many outside of the developers' control, may have contributed to this impact, outside of the feature. Consider, however, a feature that is very frequently observed to have been included in app releases that were very impactful: the resourceful developer will begin to wonder if they should include this illustrious feature in their next release too. This is because correlation provides evidence that suggests a relationship, even if it cannot prove it.

In this study, we seek to find the set of feature changes that are most prevalent amongst the most impactful app releases studied in Section 7.4. We analyse the changes in features in app release text at the time of release, using the topic modelling technique to identify topics that are analogous to features (explained in

Chapter 4). We break down the difference in proportion of topic changes between impactful and non-impactful releases, and positive and negative releases, in order to identify the topic changes that are most different between the sets. The results of this study provide evidence that suggests a relationship between topics and release impact.

7.5.2 Contributions

The contributions of this study are as follows:

i) We have identified a set of features added to impactful releases that aren't added to non-impactful releases, and vice-versa: such as "start stop data gps directly" and "run smoother make faster content".

ii) We have identified a set of features added to positive releases that aren't added to negative releases, and vice-versa: such as "better performance faster optimisation enhancement" and "sound thanks rate quality really".

iii) We have identified specific changed features at a category granularity: providing meaningful results for developers in each specific category.

7.5.3 Data

We use the dataset mined from Google Play, that consists of 14,592 apps and 26,339 releases, as described in Section 7.4.2.

7.5.4 Research Questions

The following questions are answered using the topic modelling method first described in Chapter 4, that treats topics analogously to features.

RQ1: What are the features that are added or removed from releases that are impactful, that differ from releases that are non-impactful?

We find the features that are most commonly added to impactful app releases, that aren't added (or are removed) from non-impactful releases. Additionally, we find the features that are most commonly removed from impactful app releases, that may be added to non-impactful releases.

RQ2: What are the features that are added or removed from releases that are positive, that differ from releases that are negative?

We find the features that are most commonly added to impactful app releases that positively affected their app performance, that aren't added (or are removed) from negative releases. Additionally, we find the features that are most commonly removed from positive app releases, that may be added to negative releases.

7.5.5 Methodology

Process: We use the following process to identify the how different features appear in changed release text.

1. Separate impactful from non-impactful releases: We separate the releases that had a significant change in post-release user rating performance, as identified by `CIRA` in Section 7.4.

2. Separate positive from negative impactful releases: We further separate the impactful releases by analysing the subsequent change as a positive change or a negative change, i.e. an increase or decrease in rating or rating frequency.

3. Filter app releases for those with changed release text: In order that we can analyse and compare the changes which may contribute to impactful releases, we consider only those releases with changed text. It may seem surprising that some releases, that have significantly impacted the app's performance, did not change their release text in any way (yet did update the app). However, this reinforces the notion that external factors also contribute to an app's performance. This filtering process may remove apps that do not have any releases with changed release text.

4. Compute the differences in changes: We consider the topics where one set reduces the count, and the other increases it, or otherwise the topics where the impactful count is at least twice that of the non-impactful count. We compute the proportion that changed from all releases in each particular set, and rank the results according to the magnitude of the difference between sets.

Topic Modelling: We use topic modelling (described in Section 3.6) in order to compute the features present in the release text, using 1000 topics in order to produce a high granularity of specialised topics. We use a topic threshold value of 0.05 in the reported results, which is sufficiently permissive to enable changes of 2-3 terms to register as an addition or removal of a topic.

Dataset: We use the persistent google dataset, that is also used in Section 7.4. This dataset consists of 14,592 apps and 26,339 releases.

Table 7.15: **RQ1 + RQ2: Topics that changed at release time**
NI: non-impact releases
I: impactful releases
N: releases that decreased rating
P: releases that increased rating
Top: proportion of topics that were added in I/P releases, and taken away or not added to NI/N releases

Bottom: proportion of topics that were removed or not changed in I/P releases, but were added in NI/N releases

NI	I	Topic	N	P	Topic
-0.01	0.30	start stop data gps directly	-1.19	-0.05	better performance faster optimisation enhancement
-0.03	0.12	reward arena class skill season	-0.93	-0.05	facebook follow like twitter news
-0.02	0.12	flash add widget version language	-0.93	-0.11	wallpaper live muzei beautiful picker
-0.01	0.12	completely redesigned design reworked companion	-0.40	0.16	map location gps data zoom
-0.03	0.09	expense report transaction income bill	-0.46	0.00	sticker testing line bot chat
-0.01	0.10	updated library internal party component	-0.43	0.00	android app time improved present
0.06	-0.06	power level ups unlock time	0.30	0.03	permission read used storage external
0.16	0.04	run smoother make faster content	0.13	-0.16	notification push receive alert receiving
0.05	-0.07	challenge reward win daily earn	0.53	0.00	trip travel destination time guide
0.03	-0.10	orientation landscape portrait tablet mode	0.59	0.05	start stop data gps directly
0.04	-0.21	sticker testing line bot chat	0.66	0.05	app inside update mean just
0.13	-0.47	wallpaper live muzei beautiful picker	0.96	0.13	sound thanks rate quality really

7.5.6 Results

We run the algorithm described in Section 7.5.5 to compute the list of features that differ between impactful releases with changed text, and non-impactful releases with changed text. Table 7.15 reports the results for RQ1 and RQ2 for topics.

Several of the top features overall are specific to certain categories of apps, and thus are not discussed in detail below. Additional category-specific results can be found in Appendix C, which may be of interest to developers in each of these specific categories.

RQ1: What are the features that are added or removed from releases that are impactful, that differ from releases that are non-impactful?

The top topic that is added to releases that become impactful is “start stop data gps directly”, which seems intuitive in its application as a feature, and does seem like a useful one for apps that use data. The results in Table 7.15 show that terms from "reward arena class skill season" are added to releases that become impactful, and not added to any releases that are non-impactful. This, however, is ambiguous in its interpretation as a feature. The second topic, "puzzle pack update piece daily", is more clear in its interpretation as a feature. Where applicable, developers might add daily packs of pieces or puzzles to their apps.

The topic "expense report transaction income bill" seems specific to utility management applications, but could potentially be applied more widely to apps that handle or contain assets of value. Several topics are specific for game apps, such as "tournament player online win ranking" and "race track mode car win". However, the topic "completely redesigned design reworked companion" is very general, but would require a lot of development effort to implement. Another more general topic is "mode ambient color weather hand", which could apply and be added to many apps.

RQ2: What are the features that are added or removed from releases that are positive, that differ from releases that are negative?

The results in Table 7.15 show that the top impactful topic, "start stop data gps directly" correlates with negative releases and not positive releases, thus indicating it is likely not a good feature to add to apps. Similarly, the topic "sound thanks rate quality really" that appears to elicit user feedback correlates with negative releases. It may also lead to negative impacts to add the feature "notification push receive alert receiving", which seems intuitively to relate to push notifications, as this feature is removed from positive releases and added to negative releases.

Conversely, the topic "facebook follow like twitter news" is removed (or downplayed such that it is no longer a significant proportion of release text) from a much higher proportion of negative releases than positive. This result suggests that Facebook and Twitter integration is not a negative feature to have, and that removing it could lead to a negative impact on success. Similarly, "better performance faster optimisation enhancement" was removed from a higher proportion of negative releases than positive, perhaps indicating that it was merely downplayed in release text: no developer would deliberately 'remove better performance'. A higher proportion of positive releases added "map location gps data zoom", and a proportion of negative releases removed it, thus indicating that map and gps integration may be a desirable feature to have.

7.5.7 Threats to Validity

In this section we discuss threats to the construct, conclusion and external validity of our findings.

Construct Validity: The same threats to validity apply from the studies detailed in Section 7.3 and Section 7.4. We build on the results of Section 7.4, in order to identify the features discussed as changed in the release text of releases.

Conclusion Validity: The conclusion validity of this study could be affected by the qualitative human assessment of features and topics in RQ1 and RQ2.

External Validity: Care is required when extending these findings to other app samples and app stores. We report the proportion of each dataset that added or removed a topic, and in each case use the maximum set of viable releases; thus mitigating a potential threat to validity in the representation of each dataset.

7.5.8 Conclusions

We have found that certain features are added exclusively to apps in impactful releases, and that other features are added exclusively to non-impactful releases. By identifying these features, developers can gain insights into things they can implement in order to achieve high impact in their releases, and features that might not result in a high subsequent impact. The feature that most correlates with positive impact is “map location gps data zoom”, that suggests map and gps integration may be a feature to add to achieve high positive impact, where applicable. The feature that most correlates with high (negative) impact is “start stop data gps directly”, indicating that apps with data usage might implement a direct control over their usage, in order to control bandwidth and potential costly data usage. We identified several other features that may apply to a variety of different app types, and that developers might try adding to their apps.

7.5.9 Threats to Validity

The threats discussed in this Section apply to the studies undertaken and detailed in Sections 7.3 to 7.5.

Construct Validity: The gap between data analysis and causality is large, forcing any user to make very strong assumptions if they hope to effectively imply causality.

Causal analysis can help to reduce this gap, but no such analysis could hope to fully close it; there will always be unknown factors which may nevertheless have affected the data. In the case of app stores, there will always be potential external influences for which no data is available to capture them.

We apply causal impact analysis in our experiments, but there are other forms of causal analysis such as differences-in-differences. Nonetheless, we believe this method is the most suitable due to its independent consideration of each app release, and the ability to use all non-releasing apps as the control set in every experiment, thus reducing the risk of control set choice influencing results.

Since we are computing multiple p values, the reader might expect some kind of correction, such as a Bonferroni or Benjamini-Hochberg [24] correction for multiple statistical testing at the (traditionally popular) 0.05 probability level (corresponding to the 95% confidence interval). However, since we are *not* using p values to test for significance; should a p value lie above this (corrected) threshold, then this *does not* necessarily indicate that the property does not contribute to the observed causal impact. Quite the contrary; since we have already observed that there exists a causal impact, then the property that exhibits the lowest p value remains that with the highest probability of having *some* influence on the causal impact, amongst those properties assessed using inferential statistics.

7.6 Related Work

In this section, we discuss previous work on software releases and causal analysis in software engineering.

There has been a large amount of recent work linking software quality with user perceived quality. Ruiz et al. [238] studied how ad library usage affected user ratings. Bavota et al. [22] investigated how the changes and faults present in APIs used affected apps' ratings. Panichella et al. [218] classified user reviews for software maintenance. Palomba et al. [215] studied how developers responding to user feedback can increase the rating. Moran et al. [200] presented an Android bug reporting tool that can increase the engagement between users and software quality.

It therefore stands to reason that software releases affect quality and consequently may affect user rating performance. In 2011 Henze and Boll [113] analysed release times and user activity in the Apple App Store, and found that Sunday evening is the best time for deploying games. In 2013 Datta and Kajanan [67] found that apps receive more reviews after deploying updates on Thursday or late in the week. In 2015 Gui et al. found that 23% of releases from frequently-releasing apps contained ad-related changes [101]. Comino et al. [55] studied the top apps in Apple and Google stores, finding that releases can boost user downloads.

McIlroy et al. [191] studied update frequencies in the Google Play store, finding that only 1% of studied apps received more than one update per week. These findings support our weekly data collection schedule, as very few releases can be 'missed' by collecting data weekly; additionally the target releases we use (defined in Section 7.4.2), mandate that very frequently updated apps are excluded due to lack of sufficient prior and posterior time series data. McIlroy et al. [191] also found that rating was not affected by update frequency, however the findings by Guerrouj et al. [100] indicate that high code churn in releases correlates with lower ratings. Nayebi et al. [205] surveyed developers and users, finding that half of developers had clear releasing strategies, and many experienced developers thought that releasing strategy affects user feedback. Users were not more likely to install apps based on release date or frequency, but preferred to install apps which have been infrequently, but recently, updated.

All of these previous findings on app releases tantalisingly point to the possibility that certain releases may have higher causal significance than others. However, no previous study has specifically addressed the question of how to identify the set of releases that are significant. Furthermore, no previous work attempted to identify the characteristics of highly significant app releases: the primary technical and scientific contributions of the present study.

To the best of our knowledge, this is the first study to apply causal impact analysis to app store analysis. However, causal inference has been discussed in empirical science papers [147] and it has been previously used in software engineering. For example the work using the Granger causality test by Couto et al. [60, 61] and Zheng et al. [306] as a means of software defect prediction, and the work by Ceccarelli et al. [42] on identifying software artefacts affected by a change. Since `CIRA` enables the application causal impact analysis on any time series vector, future work will use it to analyse other metrics from app store data, and other time series datasets.

Chapter 8

Conclusions and Future Work

App store analysis is a new research field that concerns mining empirical data about apps from app stores. From 2012 to 2016, the field has continued to grow and expand in the subfields of research that it encompasses, as discussed by the survey in Chapter 2. This thesis presents novel contributions to the field of app store analysis, helping to define the field, identify associated issues that researchers should perhaps be aware of, and helping to facilitate future research through the inclusion of a checklist of data inclusion guidelines. These guidelines are adhered to where relevant in this thesis, but including data on the app stores, quantity and popularity range, reviews, and any empirical metrics used. It is the hope that future app store analysis work will bear this checklist in mind when publishing work, and continue to build on it and increase the information given about research carried out, to support future synthesis in surveys.

Our survey identified the key sub-fields of app store analysis to date: API analysis, feature analysis, release engineering, review analysis, security analysis, store ecosystem comparison, and analysis of size and effort prediction. A key outcome of this study was the definition of apps and app store in the context of app store analysis research, that may help to define which research is part of the field and which research is not. This helps to facilitate future literature surveys, and to point authors to the relevant literature in their particular subfield in order to help prevent replication and direct research to new, novel ventures.

The survey included as part of this thesis is not the end point for app store analysis; rather, it is the starting point of a well defined body of literature, that will encourage future app store analysis authors and surveyors to advance the field. We identify areas in which the field can grow, for example the extension to smaller stores such as Windows Phone store, and an increase in the use of time series data; but our analysis is by no means complete, and we encourage readers to identify potential research options we missed. We also encourage future surveyors to ask research questions, which can lead to more specific SLRs, now that an initial body of literature has been identified.

This thesis tries to answer the question of "what makes successful apps successful?", which it tackles from several different approaches. This was aimed not only at researchers in the field, but at the developers of apps, hoping to increase knowledge and awareness from their perspective. The first approach was to develop an approach to feature extraction from textual descriptions, that uses topic modelling to train topics, in a way analogous to feature extraction. The goal was to uncouple feature extraction from individual stores, as the n-gram method was. The approach was validated by replicating results previously run using the n-gram feature extraction algorithm.

Our topic extraction method was useful in performing the studies in Chapters 5 and 7, but it by no means the only solution which may have been applied. We encourage researchers to develop alternative methods for feature extraction, and to reuse the alternatives discussed in Chapter 2. We also encourage comparison of results, between different methods and sources of feature extraction. This is particularly relevant to the studies in Chapter 7, where we may miss many potential properties of impactful releases.

After mining app store data to tackle the problem of what makes apps successful, we encounted the problem that only a subset of app store data is available, starting with the most popular. Further inspection showed that this is a methodological problem afflicting app store analysis research more widely. We dubbed the "app sampling problem", and investigated its effects by comparing metrics and trends between sets of varying completeness of review data. Our investigation was limited, therefore, to our defined sets as ranked by review data, yet the problem applied more widely. Overall, this study was aimed at raising awareness of the problem, whilst identifying properties and trends that differ between sets of varying completeness, and trends that are consistent. We encourage researchers to take the app sampling problem into account when performing studies, and to extend our work to investigate how results are affected.

We next tackled the question of "what makes successful apps successful?" using a dataset accumulated over time, which featured app releases from Google Play and Windows Phone Store. Causal inference provides a method with which to assess the subsequent impact of releases on performance, and causal impact analysis is particularly suited to app store analysis data sets due to its independent consideration of each release. For this reason we used causal impact analysis for our experiments, but other causal inference methods are available, such as differences-in-differences analysis. We encourage future researchers to explore use of these tools for app store analysis, and to develop new approaches for identification of events or properties of significance.

The Google tool `CausalImpact` was used for our initial study into the releases which had a significant subsequent impact on app performance ("impactful" releases), and enabled the identification of properties such as price and release text, in impactful releases from Google and Windows stores. For the extension to the large app store datasets, such as the dataset of 36,000 Google Play apps recorded weekly over a period of 12 months, it was necessary to implement `CIRA`, a Java implementation of causal impact analysis which uses alternative methods to process and assign the global variance component: the control set.

`CIRA`'s implementation enables the processing of large control sets, and made the analysis of our data set of 36,000 apps possible. This subsequent study proved that the previous findings held true, such as the finding that higher priced (paid) app releases are more likely to have a positive effect, and also enabled access to a larger selection of app developers. As part of the study, we contacted 52 developers of apps with impactful releases, regardless of whether their particular impact was positive or negative, in order to ask a number of questions regarding the results of the study. 78% of developers agreed with the observed impacts, of which 33% said that they would consider changing their releasing strategy as a result of the findings.

As an extension to this work, we investigated the features that correlate with highly impactful releases, and that correlate with highly impactful positive releases. The findings show several general features that correlate with positive impactful releases, such "start stop data gps directly", and others that correlate with negative impactful releases, such as "facebook follow like twitter news". Many of the features that differed the most between impactful and non-impactful (or positive and negative impactful) datasets were domain-specific. We therefore include the category specific results in Appendix C, which provide insights for developers in each of these domains, and show that in-depth analysis is necessary to provide meaningful results when considering such a large sample of popular apps. The features identified through this study were very specific to the dataset, and time period, studied, and we therefore cannot claim that these features correlate with impact in other stores or samples. Rather, we conducted these experiments to provide some insight into features which correlate with success, and to demonstrate a means with which to do so. We encourage both researchers and app developers to try these methods, to identify other features that correlate with success.

Our implementation of causal impact analysis, `CIRA`, is available for use online via a web interface, and also has a built-in multi-platform desktop GUI. We implemented two other extensions, to maximise the utility of such an approach: an ensemble classifier, that runs multiple classifiers using alternative techniques for correcting baseline prediction; feature regression, that enables the training via linear regression of features, to help prediction. Unfortunately we did not apply these extensions in studies relating to app store analysis, but it is our hope that future studies will make use of them.

This thesis provides several contributions, through the method of extracting features via topic modelling, the identification of the app sampling problem, the implementation of `CIRA`, and the identification of some of the properties of impactful app releases. We have, in doing so, explored several avenues of what makes successful apps successful, yet there remain many unexplored avenues for answering this question. We hope to have provided some potential options or inspiration for future research, and to have identified gaps in the research through the literature survey. In synthesising these two contributions, we find that prediction is likely to be a powerful tool for app developers and researchers alike, and it may be that causal inference can be adapted to this task.

No matter how sophisticated the predictors or methodology for analysing data, it is important that any approaches continue to be applied to recent and relevant data. Doing this helps to mitigate the app sampling problem and other similar bias effects, and provides actionable findings to both researchers and developers, as was the aim of this thesis.

Bibliography

[1] N. A. S. Abdullah, N. I. A. Rusli, and M. F. Ibrahim, “Mobile game size estimation: Cosmic fsm rules, uml mapping model and unity3d game engine,” in *IEEE Conference on Open Systems (ICOS)*, 2014, pp. 42–47.

[2] B. Adams, S. Bellomo, C. Bird, T. Marshall-Keim, F. Khomh, and K. Moir, “The practice and future of release engineering: A roundtable with three release engineers,” *IEEE Software*, vol. 32, no. 2, pp. 42–49, 2015.

[3] N. Agarwal, R. Karimpour, and G. Ruhe, “Theme-based product release planning: An analytical approach,” in *System Sciences (HICSS), 2014 47th Hawaii International Conference on*. IEEE, 2014, pp. 4739–4748.

[4] A. Al-Subaihin, A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang, “App store mining and analysis,” in *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*, DeMobile 2015. ACM, 2015, pp. 1–2.

[5] C. Albanesius, “Mobile app reviews: Google ’Bouncer’ now scanning Android Market for malware,” http://uk.pcmag.com/apps/66697/news/google-bouncer-now-scanning-android-market-for-mal, 2012.

[6] K. Alharbi and T. Yeh, “Collect, decompile, extract, stats, and diff: Mining design pattern changes in Android apps,” in *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services*, MobileHCI ’15. ACM, 2015, pp. 515–524.

[7] Amazon.com, “Amazon Mechanical Turk,” https://www.mturk.com/mturk/welcome, 2013.

[8] Andrew, Kachites, and McCallum, “MALLET: A Machine Learning for Language Toolkit,” http://mallet.cs.umass.edu, 2002.

[9] androguard, “androguard,” https://github.com/androguard/androguard, 2015.

[10] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K. Rieck, and C. Siemens, “Drebin: Effective and explainable detection of Android malware in your pocket,” in *Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS)*, 2014.

[11] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, “FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps,” in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI ’14)*. ACM, 2014, pp. 259–269.

[12] G. Askalidis, “The impact of large scale promotions on the sales and ratings of mobile apps: Evidence from Apple’s App Store,” *CoRR*, vol. abs/1506.06857, 2015.

[13] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, “Pscout: analyzing the Android permission specification,” in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 217–228.

[14] V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, and E. Bodden, “Mining apps for abnormal usage of sensitive data,” in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE ’15, 2015, pp. 426–436.

[15] N. S. Awang Abu Bakar and I. Mahmud, “Empirical analysis of android apps permissions,” in *Advanced Computer Science Applications and Technologies (ACSAT), 2013 International Conference on*. IEEE, 2013, pp. 406–411.

[16] S. A. Azad, “Empirical studies of Android API usage: Suggesting related API calls and detecting license violations,” Master’s thesis, Concordia University, 2015.

[17] A. A. Bakar, N. S. Mahmud, and I. Mahmud, “OSSGrab: Software repositories and app store mining tool,” *Lecture Notes on Software Engineering*, vol. 1, no. 3, pp. 219–223, 2013.

[18] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji, “A methodology for empirical analysis of permission-based security models and its application to Android,” in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS ’10. ACM, 2010, pp. 73–84.

[19] A. Bartel, J. Klein, M. Monperrus, and Y. Le Traon, “Static analysis for extracting permission checks of a large scale framework: The challenges and solutions for analyzing Android,” *Software Engineering, IEEE Transactions on*, vol. 40, no. 6, pp. 617–632, 2014.

[20] O. Bastani, S. Anand, and A. Aiken, “Interactively verifying absence of explicit information flows in Android apps,” in *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA 2015. ACM, 2015, pp. 299–315.

[21] L. Batyuk, M. Herpich, S. A. Camtepe, K. Raddatz, A.-D. Schmidt, and S. Albayrak, “Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications,” in *Proceedings of the 2011 6th International Conference on Malicious and Unwanted Software*, MALWARE ’11. IEEE Computer Society, 2011, pp. 66–72.

[22] G. Bavota, M. Linares-Vasquez, C. E. Bernal-Cardenas, M. D. Penta, R. Oliveto, and D. Poshyvanyk, “The impact of API change-and fault-proneness on the user ratings of Android apps,” *IEEE Transactions on Software Engineering*, vol. 41, no. 4, pp. 384–407, 2015.

[23] BBC, "Flappy Bird creator removes game from app stores," http://www.bbc.co.uk/news/technology-26114364, 2014.

[24] Y. Bejamini and Y. Hochberg, "Controlling the false discovery rate: A practical and powerful approach to multiple testing," *Journal of the Royal statistical Society (Series B)*, vol. 57, no. 1, pp. 289–300, 1995.

[25] D. L. Ben Lulu and T. Kuflik, "Functionality-based clustering using short textual description: Helping users to find apps installed on their mobile device," in *Proceedings of the 2013 International Conference on Intelligent User Interfaces*, IUI '13. ACM, 2013, pp. 297–306.

[26] D. L. Ben Lulu and T. Kuflik, "Wise mobile icons organization: Apps taxonomy classification using functionality mining to ease apps finding," *Mobile Information Systems*, 2015, article ID: 3083450.

[27] G. Berardi, A. Esuli, T. Fagni, and F. Sebastiani, "Multi-store metadata-based supervised mobile app classification," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15. ACM, 2015, pp. 585–588.

[28] M. Bertrand, E. Duflo, and S. Mullainathan, "How much should we trust differences-in-differences estimates?" National Bureau of Economic Research, Tech. Rep., 2002.

[29] P. Bhattacharya, L. Ulanova, I. Neamtiu, and S. C. Koduru, "An empirical analysis of bug reports and bug fixing in open source Android apps," in *Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering*, CSMR '13. IEEE Computer Society, 2013, pp. 133–143.

[30] R. Bhoraskar, S. Han, J. Jeon, T. Azim, S. Chen, J. Jung, S. Nath, R. Wang, and D. Wetherall, "Brahmastra: Driving apps to test the security of third-party components," in *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14. USENIX Association, 2014, pp. 1021–1036.

[31] T. Bläsing, L. Batyuk, A. Schmidt, S. A. Çamtepe, and S. Albayrak, "An Android application sandbox system for suspicious software detection," in *Proceedings of the 5th International Conference on Malicious and Unwanted Software, MALWARE'10*, 2010, pp. 55–62.

[32] D. M. Blei, A. Ng, and M. Jordan, "Latent Dirichlet allocation," *JMLR*, vol. 3, pp. 993–1022, 2003.

[33] M. Böhmer, B. Hecht, J. Schöning, A. Krüger, and G. Bauer, "Falling asleep with angry birds, facebook and kindle: A large scale study on mobile application usage," in *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '11. ACM, 2011, pp. 47–56.

[34] T. Book, A. Pridgen, and D. S. Wallach, "Longitudinal analysis of android ad library permissions," *CoRR*, vol. abs/1303.0857, 2013.

[35] H. S. Borges and M. T. Valente, “Mining usage patterns for the Android API,” *PeerJ Computer Science*, no. 1:e12, 2015.

[36] K. H. Brodersen, “CausalImpact,” https://google.github.io/CausalImpact/CausalImpact.html, retrieved 28th May 2015.

[37] K. H. Brodersen, F. Gallusser, J. Koehler, N. Remy, and S. L. Scott, “Inferring causal impact using bayesian structural time-series models,” *Annals of Applied Statistics*, vol. 9, pp. 247–274, 2015.

[38] M. Butler, “Android: Changing the mobile landscape,” *IEEE Pervasive Computing*, vol. 10, no. 1, pp. 4–7, 2011.

[39] J. Callaham, “Google says there are now 1.4 billion active Android devices worldwide,” http://www.androidcentral.com/google-says-there-are-now-14-billion-active-android-devices-worldwide, 2015.

[40] J. C. Campbell, A. Hindle, and J. N. Amaral, “Syntax errors just aren’t natural: Improving error reporting with language models,” in *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR ’14. ACM, 2014, pp. 252–261.

[41] B. Carbunar and R. Potharaju, “A longitudinal study of the Google app market,” in *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, ASONAM ’15. ACM, 2015, pp. 242–249.

[42] M. Ceccarelli, L. Cerulo, G. Canfora, and M. Di Penta, “An eclectic approach for change impact analysis,” in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*. ACM, 2010, pp. 163–166.

[43] L. Cen, C. S. Gates, L. Si, and N. Li, “A probabilistic discriminative model for Android malware detection with decompiled source code,” *IEEE Trans. Dependable Sec. Comput.*, vol. 12, no. 4, pp. 400–412, 2015.

[44] L. Cen, D. Kong, H. Jin, and L. Si, “Mobile app security risk assessment: A crowdsourcing ranking approach from user comments,” in *Proceedings of the 2015 SIAM International Conference on Data Mining*, 2015, pp. 658–666.

[45] L. Cen, L. Si, N. Li, and H. Jin, “User comment analysis for Android apps and CSPI detection with comment expansion,” in *Proceeding of the 1st International Workshop on Privacy-Preserving IR (PIR)*, 2014, pp. 25–30.

[46] S. Chakradeo, B. Reaves, P. Traynor, and W. Enck, “Mast: Triage for market-scale mobile malware analysis,” in *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec ’13. ACM, 2013, pp. 13–24.

[47] R. Chandy and H. Gu, “Identifying spam in the iOS App Store,” in *Proceedings of the 2Nd Joint WICOW/AIRWeb Workshop on Web Quality*, WebQuality ’12. ACM, 2012, pp. 56–59.

[48] K. Chen, P. Wang, Y. Lee, X. Wang, N. Zhang, H. Huang, W. Zou, and P. Liu, "Finding unknown malice in 10 seconds: Mass vetting for new threats at the Google-Play scale," in *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15. USENIX Association, 2015, pp. 659–674.

[49] M. Chen and X. Liu, "Predicting popularity of online distributed applications: iTunes App Store case analysis," in *Proceedings of the 2011 iConference*, iConference '11. ACM, 2011, pp. 661–663.

[50] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang, "Ar-miner: Mining informative reviews for developers from mobile app marketplace," in *Proceedings of the 36th International Conference on Software Engineering*, ICSE '14. ACM, 2014, pp. 767–778.

[51] Y. Chen, H. Xu, Y. Zhou, and S. Zhu, "Is this app safe for children?: A comparison study of maturity ratings on Android and iOS applications," in *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13. International World Wide Web Conferences Steering Committee, 2013, pp. 201–212.

[52] P. H. Chia, Y. 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*, WWW '12. ACM, 2012, pp. 311–320.

[53] L. Cocco, K. Mannaro, G. Concas, and M. Marchesi, "Simulation of the best ranking algorithms for an app store," in *Mobile Web Information Systems*, Lecture Notes in Computer Science. Springer International Publishing, 2014, vol. 8640, pp. 233–247.

[54] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[55] S. Comino, F. M. Manenti, and F. Mariuzzo, "Updates management in mobile applications. iTunes vs Google Play," *Centre for Competition Policy (CCP), University of East Anglia*, 2015.

[56] F. T. Commission, "Complying with COPPA: Frequently Asked Questions," https://www.ftc.gov/tips-advice/business-center/guidance/complying-coppa-frequently-asked-questions, 2015.

[57] L. Corral and I. Fronza, "Better code for better apps: A study on source code quality and market success of Android applications," in *Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems*, MOBILESoft '15. IEEE Press, 2015, pp. 22–32.

[58] COSMIC, "Common software measurement international consortium," http://cosmic-sizing.org/, 2015.

[59] P. Coulton and W. Bamford, "Experimenting through mobile 'apps' and 'app stores'," *Int. J. Mob. Hum. Comput. Interact.*, vol. 3, no. 4, pp. 55–70, 2011.

[60] C. Couto, P. Pires, M. T. Valente, R. S. Bigonha, and N. Anquetil, "Predicting software defects with causality tests," *Journal of Systems and Software*, vol. 93, pp. 24–41, 2014.

[61] C. Couto, C. Silva, M. T. Valente, R. Bigonha, and N. Anquetil, "Uncovering causal relationships between software metrics and bugs," in *Proceedings of the 16th European Conference on Software Maintenance and Reengineering (CSMR'12)*, 2012, pp. 223–232.

[62] Creative Research Systems, "Sample size calculator," http://www.surveysystem.com/sscalc.htm, 2012.

[63] J. Crussell, C. Gibler, and H. Chen, "Attack of the clones: Detecting cloned applications on Android markets," in *Computer Security–ESORICS 2012*. Springer, 2012, pp. 37–54.

[64] J. Crussell, C. Gibler, and H. Chen, "Andarwin: Scalable detection of semantically similar Android applications," in *Computer Security–ESORICS 2013*. Springer, 2013, pp. 182–199.

[65] J. Crussell, R. Stevens, and H. Chen, "Madfraud: Investigating ad fraud in Android applications," in *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*. ACM, 2014, pp. 123–134.

[66] DAI-Labor, "Androlyzer," https://androlyzer.com/, 2015.

[67] D. Datta and S. Kajanan, "Do app launch times impact their subsequent commercial success? an analytical approach," in *Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference on*. IEEE, 2013, pp. 205–210.

[68] L. D'Avanzo, F. Ferrucci, C. Gravino, and P. Salza, "Cosmic functional measurement of mobile applications and code size estimation," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15. ACM, 2015, pp. 1631–1636.

[69] Dave Chaffey, "Mobile marketing statistics compilation," http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/, 2016.

[70] Z. Deng, B. Saltaformaggio, X. Zhang, and D. Xu, "iris: Vetting private API abuse in iOS applications," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 44–56.

[71] M. L. Dering and P. McDaniel, "Android Market reconstruction and analysis," in *Proceedings of the 2014 IEEE Military Communications Conference*, MILCOM '14. IEEE Computer Society, 2014, pp. 300–305.

[72] N. d'Heureuse, F. Huici, M. Arumaithurai, M. Ahmed, K. Papagiannaki, and S. Niccolini, "What's app?: a wide-scale measurement study of smart phone markets," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 16, no. 2, pp. 16–27, 2012.

[73] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10. USENIX Association, 2010, pp. 1–6.

[74] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09. ACM, 2009, pp. 235–245.

[75] D. Erić, R. Bačík, and I. Fedorko, "Rating decision analysis based on iOS App Store data," *Quality Innovation Prosperity*, vol. 18, no. 2, pp. 27–37, 2014.

[76] D. Ferreira, V. Kostakos, and A. K. Dey, "Lessons learned from large-scale user studies: Using Android Market as a source of data," *Int. J. Mob. Hum. Comput. Interact.*, vol. 4, no. 3, pp. 28–43, 2012.

[77] F. Ferrucci, C. Gravino, P. Salza, and F. Sarro, "Investigating functional and code size measures for mobile applications," in *41st Euromicro Conference series on Software Engineering and Advanced Applications*, SEAA '15. IEEE, 2015, pp. 365–368.

[78] F. Ferrucci, C. Gravino, P. Salza, and F. Sarro, "Investigating functional and code size measures for mobile applications: A replicated study," in *16th International Conference on Product-Focused Software Process Improvement*, PROFES '15, 2015.

[79] A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang, "App store analysis: Mining app stores for relationships between customer, business and technical characteristics," Tech. Rep., 2014, rN/14/10.

[80] R. Francese, C. Gravino, M. Risi, G. Scanniello, and G. Tortora, "On the use of requirements measures to predict software project and product measures in the context of Android mobile apps: a preliminary study," in *41st Euromicro Conference series on Software Engineering and Advanced Applications (SEAA '15)*. IEEE, 2015, pp. 357–364.

[81] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Mach. Learn.*, vol. 29, no. 2-3, pp. 131–163, 1997.

[82] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. 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. ACM, 2013, pp. 1276–1284.

[83] L. V. Galvis Carreño and K. Winbladh, "Analysis of user comments: An approach for software requirements evolution," in *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13. IEEE Press, 2013, pp. 582–591.

[84] C. Gao, H. Xu, J. Hu, and Y. Zhou, "Ar-tracker: Track the dynamics of mobile apps via user review mining," in *2015 IEEE Symposium on Service-Oriented System Engineering, SOSE '15*, 2015, pp. 284–290.

[85] R. Garg and R. Telang, "Inferring app demand from publicly available data," *MIS Q.*, vol. 37, no. 4, pp. 1253–1264, 2013.

[86] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "AndroidLeaks: Automatically detecting potential privacy leaks in Android applications on a large scale," in *Proceedings of the 5th International Conference on Trust and Trustworthy Computing*, TRUST'12. Springer-Verlag, 2012, pp. 291–307.

[87] C. Gibler, R. Stevens, J. Crussell, H. Chen, H. Zang, and H. Choi, "Adrob: Examining the landscape and impact of Android application plagiarism," in *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*. ACM, 2013, pp. 431–444.

[88] GinLemon, "Smart launcher 3 — simple. light. fast." http://www.smartlauncher.net/, 2011.

[89] GitHub, Inc., "Github," https://github.com/, 2014.

[90] W. Glodek and R. Harang, "Rapid permissions-based detection and analysis of mobile malware using random decision forests," in *Military Communications Conference, MILCOM 2013-2013 IEEE*. IEEE, 2013, pp. 980–985.

[91] M. Gómez, M. Martinez, M. Monperrus, and R. Rouvoy, "When app stores listen to the crowd to fight bugs in the wild," in *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ICSE '15. IEEE Press, 2015, pp. 567–570.

[92] M. Gomez, R. Rouvoy, M. Monperrus, and L. Seinturier, "A recommender system of buggy app checkers for app store moderators," in *2nd ACM International Conference on Mobile Software Engineering and Systems*, D. Dig and Y. Dubinsky, Eds. IEEE, 2015.

[93] Google, "GitHub – google/guava: Google Core Libraries for Java 6+," https://github.com/google/guava, 2016.

[94] Google and MobiSystems, "OfficeSuite Pro + PDF Android Apps on Google Play," https://play.google.com/store/apps/details?id=com.mobisystems.editor.office_registered, retrieved 8th March 2016.

[95] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proceedings of the 2014 International Conference on Software Engineering*, ICSE '14. ACM Press, 2014, pp. 292–302.

[96] M. Goul, O. Marjanovic, S. Baxley, and K. Vizecky, "Managing the enterprise business intelligence app store: Sentiment analysis supported requirements engineering," in *Proceedings of the 2012 45th Hawaii International Conference on System Sciences*, HICSS '12, 2012, pp. 4168–4177.

[97] M. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements," in *Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WISEC '12. ACM, 2012, pp. 101–112.

[98] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker: Scalable and accurate zero-day Android malware detection," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12. ACM, 2012, pp. 281–294.

[99] X. Gu and S. Kim, "What parts of your apps are loved by users?" in *In Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE'15)*, 2015.

[100] L. Guerrouj, S. Azad, and P. C. Rigby, "The influence of App churn on App success and StackOverflow discussions," in *Proceedings of the 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2015, pp. 321–330.

[101] J. Gui, S. Mcilroy, M. Nagappan, and W. G. J. Halfond, "Truth in advertising: The hidden cost of mobile ads for software developers," in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15. IEEE Press, 2015, pp. 100–110.

[102] E. Guzman, O. Aly, and B. Bruegge, "Retrieving diverse opinions from app reviews," in *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '15. IEEE, 2015, pp. 1–10.

[103] E. Guzman, M. El-Haliby, and B. Bruegge, "Ensemble methods for app review classification: An approach for software evolution (N)," in *30th IEEE/ACM International Conference on Automated Software Engineering*, ASE '15, 2015, pp. 771–776.

[104] E. Guzman and W. Maalej, "How do users like this feature? a fine grained sentiment analysis of app reviews," in *22nd IEEE International Requirements Engineering Conference (RE'14)*, 2014, pp. 153 – 162.

[105] E. Ha and D. Wagner, "Do Android users write about electric sheep? examining consumer reviews in Google Play," in *10th IEEE Consumer Communications and Networking Conference*, CCNC '13. IEEE, 2013, pp. 149–157.

[106] Y. J. Ham and H.-W. Lee, "Detection of malicious Android mobile applications based on aggregated system call events," *International Journal of Computer and Communication Engineering*, vol. 3, no. 2, pp. 149–154, 2014.

[107] S. Hao, B. Liu, S. Nath, W. G. Halfond, and R. Govindan, "PUMA: Programmable UI-Automation for Large-Scale Dynamic Analysis of Mobile Apps," in *Proceedings of the 12th International Conference on Mobile Systems, Applications, and Services (MobiSys'14)*, 2014.

[108] M. Harman, A. Al-Subaihin, Y. Jia, W. Martin, F. Sarro, and Y. Zhang, "Mobile app and app store analysis, testing and optimisation," in *Proceedings of the International Conference on Mobile Software Engineering and Systems*, MOBILESoft '16. ACM, 2016, pp. 243–244.

[109] M. Harman, Y. Jia, and Y. Zhang, "App Store Mining and Analysis: MSR for App Stores," in *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, MSR '12, 2012, pp. 108–111.

[110] X. He, W. Dai, G. Cao, R. Tang, M. Yuan, and Q. Yang, "Mining target users for online marketing based on app store data," in *2015 IEEE International Conference on Big Data*. IEEE, 2015, pp. 1043–1052.

[111] S. Hedegaard and J. G. Simonsen, "Extracting usability and user experience information from online user reviews," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13. ACM, 2013, pp. 2089–2098.

[112] H. Heitkötter, S. Hanschke, and T. A. Majchrzak, "Evaluating cross-platform development approaches for mobile applications," in *Web Information Systems and Technologies - 8th International Conference, WEBIST 2012, Porto, Portugal, April 18-21, 2012, Revised Selected Papers*, 2012, pp. 120–138.

[113] N. Henze and S. Boll, "Release your app on Sunday eve: Finding the best time to deploy apps," in *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI'11)*, 2011, pp. 581–586.

[114] A. Hindle, "Green software engineering: the curse of methodology," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 5. IEEE, 2016, pp. 46–55.

[115] T.-H. Ho, D. Dean, X. Gu, and W. Enck, "PREC: Practical root exploit containment for Android devices," in *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*, CODASPY '14. ACM, 2014, pp. 187–198.

[116] P. W. Holland, "Statistics and causal inference," *Journal of the American Statistical Association*, vol. 81, no. 396, pp. pp. 945–960, 1986.

[117] L. Hoon, M. Rodriguez-Garca, R. Vasa, R. Valencia-Garca, and J.-G. Schneider, "App reviews: Breaking the user and developer language barrier," in *Trends and Applications in Software Engineering*, Advances in Intelligent Systems and Computing. Springer International Publishing, 2016, vol. 405, pp. 223–233.

[118] L. Hoon, R. Vasa, G. Y. Martino, J.-G. Schneider, and K. Mouzakis, "Awesome!: Conveying satisfaction on the app store," in *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*, OzCHI '13. ACM, 2013, pp. 229–232.

[119] L. Hoon, R. Vasa, J.-G. Schneider, and J. Grundy, "An analysis of the mobile app review landscape: trends and implications," Faculty of Information and Communication Technologies, Swinburne University of Technology, Tech. Rep., 2013.

[120] L. Hoon, R. Vasa, J.-G. Schneider, and K. Mouzakis, "A preliminary analysis of vocabulary in mobile app user reviews," in *Proceedings of the 24th Australian Computer-Human Interaction Conference*, OzCHI '12. ACM, 2012, pp. 245–248.

[121] M. Hu and B. Liu, "Mining and summarizing customer reviews," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04. ACM, 2004, pp. 168–177.

[122] J. Huang, Z. Li, X. Xiao, Z. Wu, K. Lu, X. Zhang, and G. Jiang, "SUPOR: Precise and scalable sensitive user input detection for Android apps," in *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15. USENIX Association, 2015, pp. 977–992.

[123] W. Huang, Y. Dong, A. Milanova, and J. Dolby, "Scalable and precise taint analysis for Android," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ISSTA 2015. ACM, 2015, pp. 106–117.

[124] C. Iacob and R. Harrison, "Retrieving and analyzing mobile apps feature requests from online reviews," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13. IEEE Press, 2013, pp. 41–44.

[125] C. Iacob, R. Harrison, and S. Faily, "Online reviews as first class artifacts in mobile app development," in *Proceedings of the 5th International Conference on Mobile Computing, Applications, and Services. MobiCASE '13*, 2014.

[126] C. Iacob, V. Veerappa, and R. Harrison, "What are you complaining about?: A study of online reviews of mobile applications," in *Proceedings of the 27th International BCS Human Computer Interaction Conference*, BCS-HCI '13. British Computer Society, 2013, pp. 29:1–29:6.

[127] S.-Y. Ihm, W.-K. Loh, and Y.-H. Park, "App analytic: A study on correlation analysis of app ranking data," *International Conference on Cloud and Green Computing (CGC)*, vol. 0, pp. 561–563, 2013.

[128] J. Jeon, K. K. Micinski, J. A. Vaughan, A. Fogel, N. Reddy, J. S. Foster, and T. Millstein, "Dr. Android and mr. hide: Fine-grained permissions in Android applications," in *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '12. ACM, 2012, pp. 3–14.

[129] H. Jiang, H. Ma, Z. Ren, J. Zhang, and X. Li, "What makes a good app description?" in *Proceedings of the 6th Asia-Pacific Symposium on Internetware on Internetware*, INTERNETWARE 2014. ACM, 2014, pp. 45–53.

[130] N. Jindal and B. Liu, "Opinion spam and analysis," in *Proceedings of the 2008 International Conference on Web Search and Data Mining*, WSDM '08. ACM, 2008, pp. 219–230.

[131] N. Jindal, B. Liu, and E.-P. Lim, "Finding unusual review patterns using unexpected rules," in *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10. ACM, 2010, pp. 1549–1552.

[132] Y. Jing, G.-J. Ahn, Z. Zhao, and H. Hu, "Riskmon: Continuous and automated risk assessment of mobile applications," in *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*, CODASPY '14. ACM, 2014, pp. 99–110.

[133] Y. Jo and A. H. Oh, "Aspect and sentiment unification model for online review analysis," in *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11. ACM, 2011, pp. 815–824.

[134] M. E. Joorabchi, M. Ali, and A. Mesbah, "Detecting inconsistencies in multi-platform mobile apps," in *Proceedings of the 26th IEEE International Symposium on Software Reliability Engineering*, ISSRE '15, 2015.

[135] E.-Y. Jung, C. Baek, and J.-D. Lee, "Product survival analysis for the App Store," *Marketing Letters*, vol. 23, no. 4, pp. 929–941, 2012.

[136] JUnit, "JUnit – About," http://junit.org/junit4/, 2016.

[137] H. Khalid, M. Nagappan, and A. Hassan, "Examining the relationship between FindBugs warnings and end user ratings: A case study on 10,000 Android apps," *IEEE Transactions on Software Engineering*, 2015.

[138] H. Khalid, "On identifying user complaints of iOS apps," in *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13. IEEE Press, 2013, pp. 1474–1476.

[139] H. Khalid, M. Nagappan, E. Shihab, and A. E. Hassan, "Prioritizing the devices to test your app on: A case study of Android game apps," in *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014. ACM, 2014, pp. 610–620.

[140] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan, "What do mobile app users complain about?" *IEEE Software*, vol. 32, no. 3, pp. 70–77, 2015.

[141] M. Khalid, M. Asif, and U. Shehzaib, "Towards improving the quality of mobile app reviews," *International Journal of Information Technology and Computer Science (IJITCS)*, vol. 7, no. 10, p. 35, 2015.

[142] M. Khalid, U. Shehzaib, and M. Asif, "A case of mobile app reviews as a crowdsource," *International Journal of Information Engineering and Electronic Business (IJIEEB)*, vol. 7, no. 5, p. 39, 2015.

[143] K. Khanmohammadi, M. R. Rejali, and A. Hamou-Lhadj, "Understanding the service life cycle of Android apps: An exploratory study," in *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '15. ACM, 2015, pp. 81–86.

[144] F. Khomh, T. Dhaliwal, Y. Zou, and B. Adams, "Do faster releases improve software quality? an empirical case study of Mozilla Firefox," in *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (MSR'12)*, 2012, pp. 179–188.

[145] D. Kim, A. Gokhale, V. Ganapathy, and A. Srivastava, “Detecting plagiarized mobile apps using API birthmarks,” *Automated Software Engineering*, pp. 1–28, 2015.

[146] J. Kim, Y. Park, C. Kim, and H. Lee, “Mobile application service networks: Apple’s App Store,” *Service Business*, vol. 8, no. 1, pp. 1–27, 2014.

[147] B. A. Kitchenham, T. Dyba, and M. Jorgensen, “Evidence-based software engineering,” in *Proceedings of the 26th International Conference on Software Engineering*, ICSE ’04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 273–281.

[148] N. Klein, C. S. Corley, and N. A. Kraft, “New features for duplicate bug detection,” in *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR ’14. ACM, 2014, pp. 324–327.

[149] D. E. Krutz, M. Mirakhorli, S. A. Malachowsky, A. Ruiz, J. Peterson, A. Filipski, and J. Smith, “A dataset of open-source Android applications,” in *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR ’15. IEEE Press, 2015, pp. 522–525.

[150] N. Lageman, M. Lindsey, and W. Glodek, “Detecting malicious Android applications from runtime behavior,” in *Military Communications Conference, MILCOM 2015-2015 IEEE*. IEEE, 2015, pp. 324–329.

[151] G. Lee and T. S. Raghu, “Product portfolio and mobile apps success: Evidence from App Store market,” in *Proceedings of the 17th Americas Conference on Information Systems AMCIS ’11*, V. Sambamurthy and M. Tanniru, Eds. Association for Information Systems, 2011.

[152] G. Lee and T. Raghu, “Determinants of mobile apps’ success: Evidence from the app store market,” *Journal of Management Information Systems*, vol. 31, no. 2, pp. 133–170, 2014.

[153] L. Li, A. Bartel, T. F. D. A. Bissyande, J. Klein, Y. Le Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. McDaniel, “Iccta: detecting inter-component privacy leaks in Android apps,” in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, ICSE ’15, 2015.

[154] T.-P. Liang, X. Li, C.-T. Yang, and M. Wang, “What in consumer reviews affects the sales of mobile apps: A multifacet sentiment analysis approach,” *International Journal of Electronic Commerce*, vol. 20, no. 2, pp. 236–260, 2015.

[155] S. Lim, P. Bentley, N. Kanakam, F. Ishikawa, and S. Honiden, “Investigating country differences in mobile app user behavior and challenges for software engineering,” *IEEE Transactions on Software Engineering (TSE)*, 2015.

[156] S. L. Lim and P. J. Bentley, “App epidemics: Modelling the effects of publicity in a mobile app ecosystem,” in *Artificial Life 13: Proceedings of the Thirteenth International Conference on the Simulation and Synthesis of Living Systems (ALIFE)*, 2012.

[157] S. L. Lim and P. J. Bentley, "How to be a successful app developer: lessons from the simulation of an app ecosystem," *ACM SIGEVOlution*, vol. 6, no. 1, pp. 2–15, 2012.

[158] S. L. Lim and P. J. Bentley, "Investigating app store ranking algorithms using a simulation of mobile app ecosystems," in *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2013*, 2013, pp. 2672–2679.

[159] S. L. Lim, P. J. Bentley, and F. Ishikawa, "The effects of developer dynamics on fitness in an evolutionary ecosystem model of the App Store," *IEEE Transactions on Evolutionary Computation (TEVC)*, vol. PP, 2015.

[160] J. Lin, S. Amini, J. I. Hong, N. Sadeh, J. Lindqvist, and J. Zhang, "Expectation and purpose: Understanding users' mental models of mobile app privacy through crowdsourcing," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12. ACM, 2012, pp. 501–510.

[161] J. Lin, K. Sugiyama, M.-Y. Kan, and T.-S. Chua, "Addressing cold-start in app recommendation: Latent user models constructed from twitter followers," in *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13. ACM, 2013, pp. 283–292.

[162] J. Lin, K. Sugiyama, M.-Y. Kan, and T.-S. Chua, "New and improved: Modeling versions to improve app recommendation," in *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '14. ACM, 2014, pp. 647–656.

[163] M. Linares-Vásquez, "Supporting evolution and maintenance of Android apps," in *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014. ACM, 2014, pp. 714–717.

[164] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. 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. ACM, 2013, pp. 477–487.

[165] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Mining energy-greedy API usage patterns in Android apps: An empirical study," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR '14. ACM, 2014, pp. 2–11.

[166] M. Linares-Vásquez, G. Bavota, C. E. B. Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Optimizing energy consumption of guis in android apps: A multi-objective approach," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, 2015, pp. 143–154.

[167] M. Linares-Vásquez, A. Holtzhauer, C. Bernal-Cárdenas, and D. Poshyvanyk, "Revisiting Android reuse studies in the context of code obfuscation and library usages," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR '14. ACM, 2014, pp. 242–251.

[168] B. Liu, D. Kong, L. Cen, N. Z. Gong, H. Jin, and H. Xiong, "Personalized mobile app recommendation: Reconciling app functionality and user privacy preference," in *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, WSDM '15. ACM, 2015, pp. 315–324.

[169] B. Liu, B. Liu, H. Jin, and R. Govindan, "Efficient privilege de-escalation for ad libraries in mobile apps," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '15. ACM, 2015, pp. 89–103.

[170] B. Liu, S. Nath, R. Govindan, and J. Liu, "Decaf: Detecting and characterizing ad fraud in mobile apps," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14. USENIX Association, 2014, pp. 57–70.

[171] X. Liu and J. Liu, "A two-layered permission-based Android malware detection scheme," in *Proceedings of the 2014 2Nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, MOBILECLOUD '14. IEEE Computer Society, 2014, pp. 142–148.

[172] LLVM Developer Group, "The LLVM Compiler Infrastructure Project," http://llvm.org/, 2003.

[173] S. Ma, S. Wang, D. Lo, R. H. Deng, and C. Sun, "Active semi-supervised approach for checking app behavior against its description," in *39th IEEE Annual Computer Software and Applications Conference, COMPSAC 2015, Taichung, Taiwan, July 1-5, 2015. Volume 2*, 2015, pp. 179–184.

[174] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? on automatically classifying app reviews," *Requirements Engineering (RE15)*, 2015.

[175] M. H. Maathuis and P. Nandy, "A review of some recent advances in causal inference," *Handbook of Big Data*, p. 387, 2016.

[176] I. Malavolta, S. Ruberto, T. Soru, and V. Terragni, "End users' perception of hybrid mobile apps in the Google Play store," in *Proceedings of the 4th International Conference on Mobile Services (MS)*. IEEE, 2015.

[177] I. Malavolta, S. Ruberto, V. Terragni, and T. Soru, "Hybrid mobile apps in the Google Play store: an exploratory investigation," in *Proceedings of the 2nd ACM International Conference on Mobile Software Engineering and Systems*. ACM, 2015.

[178] L. M. Manevitz and M. Yousef, "One-class svms for document classification," *J. Mach. Learn. Res.*, vol. 2, pp. 139–154, 2002.

[179] C. D. Manning, P. Raghavan, and H. Schütze, "Scoring, term weighting, and the vector space model," in *Introduction to Information Retrieval*. Cambridge University Press, 2008, pp. 100–123.

[180] K. Mao, L. Capra, M. Harman, and Y. Jia, "A survey of the use of crowdsourcing in software engineering," Tech. Rep., 2016, rN/15/01.

[181] marketsandmarkets.com, "World Mobile Applications Market - Advanced Technologies, Global Forecast," http://www.marketsandmarkets.com/Market-Reports/mobile-applications-228.html, 2010.

[182] W. Martin, "Causal impact for app store analysis," in *Companion Proceedings of the 38th International Conference on Software Engineering*, ICSE Companion '16. ACM, 2016.

[183] W. Martin, M. Harman, Y. Jia, F. Sarro, and Y. Zhang, "The app sampling problem for app store mining," in *Proceedings of the 12th IEEE Working Conference on Mining Software Repositories*, MSR '15, 2015, pp. 123–133.

[184] W. Martin, F. Sarro, and M. Harman, "Causal impact analysis applied to app releases in Google Play and Windows Phone Store," University College London, Tech. Rep., 2015, rN/15/07.

[185] W. Martin, F. Sarro, and M. Harman, "Causal impact analysis for app releases in Google Play," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, FSE '16, 2016, to appear.

[186] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A survey of app store analysis for software engineering," *IEEE Transactions on Software Engineering*, 2016.

[187] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A survey of app store analysis for software engineering," University College London, Tech. Rep., 2016, rN/16/02.

[188] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A survey of app store analysis for software engineering," *IEEE Transactions on Software Engineering (TSE)*, 2016, under review.

[189] W. Martin and J. Shawe-Taylor, "Automatic correction of topic coherence," UCL, Tech. Rep., 2012, rN/12/16.

[190] T. McDonnell, B. Ray, and M. Kim, "An empirical study of API stability and adoption in the Android ecosystem," in *Proceedings of the 2013 IEEE International Conference on Software Maintenance*, ICSM '13. IEEE Computer Society, 2013, pp. 70–79.

[191] S. McIlroy, N. Ali, and A. E. Hassan, "Fresh apps: an empirical study of frequently-updated mobile apps in the google play store," *Empirical Software Engineering*, pp. 1–25, 2015.

[192] S. McIlroy, N. Ali, H. Khalid, and A. E. Hassan, "Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews," *Empirical Software Engineering*, pp. 1–40, 2015.

[193] S. McIlroy, W. Shang, N. Ali, and A. Hassan, "Is it worth responding to reviews? a case study of the top free apps in the Google Play store," *IEEE Software*, vol. PP, 2015.

[194] D. Mimno, H. Wallach, E. Talley, M. Leenders, and A. McCallum, "Optimizing semantic coherence in topic models," in *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2011, pp. 262–272.

[195] R. Minelli and M. Lanza, "Samoa — a visual software analytics platform for mobile applications," in *Proceedings of ICSM 2013 (29th International Conference on Software Maintenance)*. IEEE CS Press, 2013, pp. 476–479.

[196] R. Minelli and M. Lanza, "Software analytics for mobile applications–insights &amp; lessons learned," *2013 15th European Conference on Software Maintenance and Reengineering*, vol. 0, pp. 144–153, 2013.

[197] I. J. Mojica, M. Nagappan, B. Adams, T. Berger, S. Dienst, and A. E. Hassan, "An examination of the current rating system used in mobile app stores," *IEEE Software*, vol. PP, 2015.

[198] S. Mokarizadeh, M. T. Rahman, and M. Matskin, "Mining and analysis of apps in Google Play," in *Web Information Systems and Technologies - 9th International Conference, WEBIST '13*, 2013.

[199] J. E. Montandon, H. Borges, D. Felix, and M. T. Valente, "Documenting APIs with examples: Lessons learned with the APIMiner platform," in *20th Working Conference on Reverse Engineering (WCRE)*. IEEE, 2013, pp. 401–408.

[200] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, and D. Poshyvanyk, "FUSION: A tool for facilitating and augmenting Android bug reporting," in *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16. ACM, 2016, pp. 609–612.

[201] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, and D. Poshyvanyk, "Auto-completing bug reports for Android applications," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 2015, pp. 673–686.

[202] F. Morstatter, J. Pfeffer, H. Liu, and K. M. Carley, "Is the sample good enough? comparing data from twitter's streaming API with twitter's firehose," *CoRR*, vol. abs/1306.5204, 2013.

[203] P. Mutchler, A. Doupé, J. Mitchell, C. Kruegel, and G. Vigna, "A Large-Scale Study of Mobile Web App Security," in *Proceedings of the Mobile Security Technologies Workshop (MoST)*, 2015.

[204] A. Mller, S. Diewald, L. Roalter, T. U. Mnchen, F. Michahelles, and M. Kranz, "Update behavior in app markets and security implications: A case study in Google Play," in *In Proc. of the 3rd Intl. Workshop on Research in the Large. Held in Conjunction with Mobile HCI*, 2012, pp. 3–6.

[205] M. Nayebi, B. Adams, and G. Ruhe, "Mobile app releases – a survey research on developers and users perception," in *IEEE 23rd International Conference on Software Analysis, Evolution and Reengineering (SANER'16)*. IEEE, 2016.

[206] M. Nayebi, B. Adams, and G. Ruhe, "Release practices in mobile apps – users and developers perception," in *Proceedings of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1. IEEE, 2016, pp. 552–562.

[207] M. Nayebi and G. Ruhe, "Trade-off service portfolio planning–a case study on mining the Android app market," *PeerJ PrePrints*, 2015.

[208] G. Neumann, M. Harman, and S. M. Poulding, "Transformed vargha-delaney effect size," in *Search-Based Software Engineering - 7th International Symposium, SSBSE*, 2015, pp. 318–324.

[209] D. Newman, A. Asuncion, P. Smyth, and M. Welling, "Distributed algorithms for topic models," *J. Mach. Learn. Res.*, vol. 10, pp. 1801–1828, 2009.

[210] Y. Y. Ng, H. Zhou, Z. Ji, H. Luo, and Y. Dong, "Which Android app store can be trusted in China?" in *Proceedings of the 2014 IEEE 38th Annual Computer Software and Applications Conference*, COMPSAC '14. IEEE Computer Society, 2014, pp. 509–518.

[211] Object Refinery Limited, "JFreeChart," http://www.jfree.org/jfreechart/, 2014.

[212] J. Oh, D. Kim, U. Lee, J.-G. Lee, and J. Song, "Facilitating developer-user interactions with mobile app review digests," in *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '13. ACM, 2013, pp. 1809–1814.

[213] Oracle, "Java SE — Oracle Technology Network — Oracle," http://www.oracle.com/technetwork/java/javase/overview/index.html, 2016.

[214] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study." in *Proceedings of the 21st. IEEE International Requirements Engineering Conference (RE) '13*. IEEE, 2013.

[215] F. Palomba, M. Linares-Vásquez, G. Bavota, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "User reviews matter! tracking crowdsourced reviews to support evolution of successful apps," in *31st International Conference on Software Maintenance and Evolution*, ICSME '15, 2015.

[216] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "Whyper: Towards automating risk assessment of mobile applications," in *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13. USENIX Association, 2013, pp. 527–542.

[217] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms," in *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13. IEEE Press, 2013, pp. 522–531.

[218] S. Panichella, A. D. Sorbo, E. Guzman, A. Visaggio, G. Canfora, and H. Gall, "How can i improve my app? classifying user reviews for software maintenance and evolution," *31st IEEE International Conference on Software Maintenance and Evolution*, 2015.

[219] D. H. Park, M. Liu, C. Zhai, and H. Wang, "Leveraging user reviews to improve accuracy for mobile app retrieval," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15. ACM, 2015, pp. 533–542.

[220] N. Peiravian and X. Zhu, "Machine learning for Android malware detection using permission and API calls," in *Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, ICTAI '13. IEEE Computer Society, 2013, pp. 300–305.

[221] H. Peng, C. S. Gates, B. P. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Using probabilistic generative models for ranking risks of Android apps," in *ACM Conference on Computer and Communications Security*. ACM, 2012, pp. 241–252.

[222] M. Peters, "Nokia WidSets for mobile phones," http://www.letsgomobile.org/en/3465/nokia-widsets/, 2008.

[223] T. Petsas, A. Papadogiannakis, M. Polychronakis, E. P. Markatos, and T. Karagiannis, "Rise of the planet of the apps: A systematic study of the mobile app ecosystem," in *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13. ACM, 2013, pp. 277–290.

[224] X.-H. Phan and C.-T. Nguyen, "Gibbslda++," http://gibbslda.sourceforge.net, retrieved 3rd March 2016.

[225] X.-H. Phan and C.-T. Nguyen, "Jgibblda," http://jgibblda.sourceforge.net, retrieved 3rd March 2016.

[226] R. Potharaju, A. Newell, C. Nita-Rotaru, and X. Zhang, "Plagiarizing smartphone applications: Attack strategies and defense techniques," in *Proceedings of the 4th International Conference on Engineering Secure Software and Systems (ESSoS'12)*. Springer-Verlag, 2012, pp. 106–120.

[227] T. Preuss, "Mobile applications, function points and cost estimating," in *International Cost Estimation & Analysis Association Conference*, 2013.

[228] T. Preuss, "Mobile applications, functional analysis, and the customer experience," in *The IFPUG Guide to IT and Software Measurement*, IFPUG, Ed. Auerbach Publications, 2012, pp. 408–433.

[229] Python Software Foundation, "20.5. urllib – Open arbitrary resources by URL – Python 2.7.12 documentation," https://docs.python.org/2/library/urllib.html, 2016.

[230] Python Software Foundation, “spynner 2.19 : Python Package Index,” https://pypi.python.org/pypi/spynner, 2016.

[231] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen, “AutoCog: Measuring the description-to-permission fidelity in Android applications,” in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 1354–1365.

[232] V. Rastogi, Y. Chen, and W. Enck, “Appsplayground: automatic security analysis of smartphone applications,” in *Proceedings of the third ACM conference on Data and application security and privacy*. ACM, 2013, pp. 209–220.

[233] L. Ravindranath, S. Nath, J. Padhye, and H. Balakrishnan, “Automatic and scalable fault detection for mobile applications,” in *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys ’14. ACM, 2014, pp. 190–203.

[234] A.-D. Rein and J. Münch, “Feature prioritization based on mock-purchase: A mobile case study,” in *Proceedings of the Lean Enterprise Software and Systems Conference (LESS 2013)*. Springer, 2013, pp. 165–179.

[235] S. Robertson, “Understanding inverse document frequency: On theoretical arguments for idf,” *Journal of Documentation*, vol. 60, 2004.

[236] S. Roy, J. DeLoach, Y. Li, N. Herndon, D. Caragea, X. Ou, V. P. Ranganath, H. Li, and N. Guevara, “Experimental study with real-world data for Android app security analysis using machine learning,” in *Proceedings of the 31st Annual Computer Security Applications Conference*, ACSAC 2015. ACM, 2015, pp. 81–90.

[237] I. J. M. Ruiz, B. Adams, M. Nagappan, S. Dienst, T. Berger, and A. E. Hassan, “A large scale empirical study on software reuse in mobile apps,” *IEEE Software*, vol. 31, no. 2, pp. 78–86, 2014.

[238] I. J. M. Ruiz, M. Nagappan, B. Adams, T. Berger, S. Dienst, and A. E. Hassan, “Impact of ad libraries on ratings of Android mobile apps,” *IEEE Software*, vol. 31, no. 6, pp. 86–92, 2014.

[239] I. J. M. Ruiz, M. Nagappan, B. Adams, T. Berger, S. Dienst, and A. E. Hassan, “Analyzing ad library updates in android apps,” *IEEE Software*, vol. 33, no. 2, pp. 74–80, 2016.

[240] I. J. M. Ruiz, M. Nagappan, B. Adams, and A. E. Hassan, “Understanding reuse in the Android market,” in *20th IEEE International Conference on Program Comprehension*, ICPC ’12, 2012, pp. 113–122.

[241] A. Sahami Shirazi, N. Henze, A. Schmidt, R. Goldberg, B. Schmidt, and H. Schmauder, “Insights into layout patterns of mobile user interfaces by an automatic analysis of Android apps,” in *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS ’13. ACM, 2013, pp. 275–284.

[242] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, and P. G. Bringas, “On the automatic categorisation of Android applications,” in *CCNC’12*, 2012, pp. 149–153.

[243] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, J. Nieves, P. G. Bringas, and G. Álvarez, “MAMA: Manifest analysis for malware detection in Android,” *Cybernetics and Systems - Intelligent Network Security and Survivability*, vol. 44, no. 6-7, pp. 469–488, 2013.

[244] B. Sanz, I. Santos, J. Nieves, C. Laorden, I. Alonso-Gonzalez, and P. G. Bringas, “Mads: malicious Android applications detection through string analysis,” in *Network and System Security*. Springer, 2013, pp. 178–191.

[245] B. Sanz, I. Santos, X. Ugarte-Pedrero, C. Laorden, J. Nieves, and P. G. Bringas, “Instance-based anomaly method for Android malware detection,” in *Proceedings of the 10th International Conference on Security and Cryptography*, SECRYPT ’13, 2013, pp. 387–394.

[246] F. Sarro, “The UCLAppA repository: A repository of research articles on mobile software engineering and app store analysis,” http://www0.cs.ucl.ac.uk/staff/F.Sarro/projects/UCLappA/UCLappArepository.html.

[247] F. Sarro, A. Al-Subaihin, M. Harman, Y. Jia, W. Martin, and Y. Zhang, “Feature lifecycles as they spread, migrate, remain and die in app stores,” in *Proceedings of the Requirements Engineering Conference, 23rd IEEE International (RE’15)*. IEEE, 2015.

[248] J. Schütte, R. Fedler, and D. Titze, “ConDroid: Targeted dynamic analysis of Android applications,” in *29th IEEE International Conference on Advanced Information Networking and Applications*, AINA ’15, 2015, pp. 571–578.

[249] S. L. Scott and H. R. Varian, “Predicting the present with bayesian structural time series,” *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 5, no. 1-2, pp. 4–23, 2014.

[250] S. Seneviratne, H. Kolamunna, and A. Seneviratne, “A measurement study of tracking in paid mobile applications,” in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec ’15. ACM, 2015, pp. 7:1–7:6.

[251] S. Seneviratne, A. Seneviratne, M. A. Kaafar, A. Mahanti, and P. Mohapatra, “Early detection of spam mobile apps,” in *Proceedings of the 24th International Conference on World Wide Web*, WWW ’15. International World Wide Web Conferences Steering Committee, 2015, pp. 949–959.

[252] G. Sethumadhavan, “Sizing Android mobile applications,” 2011, presentation at 6th IFPUG International Software Measurement and Analysis Conference, ISMA ’11.

[253] A. Shabtai, Y. Fledel, and Y. Elovici, “Automated static code analysis for classifying Android applications using machine learning,” in *Proceedings of the 2010 International Conference on Computational Intelligence and Security*, CIS ’10. IEEE Computer Society, 2010, pp. 329–333.

[254] C. Sharma, "Sizing up the global mobile apps market," *Report, Chetan Sharma Consulting, Issaquah, WA*, 2010.

[255] C. Shuler, "iLearnII; an analysis of the education category of the iTunes App Store," *The Joan Ganz Cooney Center at Sesame Workshop*, 2012.

[256] C. E. Spearman, "The proof and measurement of association between two things," *The American Journal of Psychology*, vol. 15, no. 1, pp. 72–101, 1904.

[257] B. Staff, "Mobile marketing statistics compilation," http://www.bostoncommons.net/smartphone-adoption-in-developing-world/, 2015.

[258] R. Stevens, J. Ganz, V. Filkov, P. Devanbu, and H. Chen, "Asking for (and about) permissions used by Android apps," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13. IEEE Press, 2013, pp. 31–40.

[259] Z. Svedic, "The effect of informational signals on mobile apps sales ranks across the globe," Ph.D. dissertation, SIMON FRASER UNIVERSITY, 2015.

[260] M. D. Syer, B. Adams, Y. Zou, and A. 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. IEEE Computer Society, 2011, pp. 55–64.

[261] M. D. Syer, M. Nagappan, B. Adams, and A. E. Hassan, "Studying the relationship between source code quality and mobile platform dependence," *Software Quality Journal*, vol. 23, no. 3, pp. 485–508, 2015.

[262] M. D. Syer, M. Nagappan, A. E. Hassan, and B. Adams, "Revisiting prior empirical findings for mobile apps: An empirical case study on the 15 most popular open-source Android apps," in *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '13. IBM Corp., 2013, pp. 283–297.

[263] J. Tan, K. Nguyen, M. Theodorides, H. Negrón-Arroyo, C. Thompson, S. Egelman, and D. Wagner, "The effect of developer-specified explanations for permission requests on smartphone user behavior," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2014, pp. 91–100.

[264] P. Teufl, M. Ferk, A. Fitzek, D. Hein, S. Kraxberger, and C. Orthacker, "Malware detection by applying knowledge discovery processes to application metadata on the Android Market (Google Play)," *Security and Communication Networks*, 2013.

[265] P. Teufl, S. Kraxberger, C. Orthacker, G. Lackner, M. Gissing, A. Marsalek, J. Leibetseder, and O. Prevenhueber, "Android Market analysis with activation patterns," in *Security and Privacy in Mobile Information and Communication Systems (MobiSec)*, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg, 2012, vol. 94, pp. 1–12.

[266] The Apache Software Foundation, "Apache Tomcat – Welcome!" http://tomcat.apache.org/, 2016.

[267] The Apache Software Foundation, "Math Commons Math: The Apache Commons Mathematics Library," http://commons.apache.org/proper/commons-math/, 2016.

[268] The Apache Software Foundation, "Maven Welcome to Apache Maven," https://maven.apache.org/, 2016.

[269] Y. Tian, M. Nagappan, D. Lo, and A. E. Hassan, "What are the characteristics of high-rated apps? a case study on free Android applications," in *31st International Conference on Software Maintenance and Evolution*, ICSME '15, 2015, pp. 1–10.

[270] Y.-X. Tong, J. She, and L. Chen, "Towards better understanding of app functions," *Journal of Computer Science and Technology*, vol. 30, no. 5, pp. 1130–1140, 2015.

[271] S. Vakulenko, O. Müller, and J. v. Brocke, "Enriching iTunes App Store categories via topic modeling," in *International Conference on Information Systems (ICIS'14)*, 2014.

[272] H. van Heeringen and E. Van Gorp, "Measure the functional size of a mobile app: Using the cosmic functional size measurement method," in *Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2014 Joint Conference of the International Workshop on*. IEEE, 2014, pp. 11–16.

[273] A. Vargha and H. D. Delaney, "A critique and improvement of the "CL" common language effect size statistics of McGraw and Wong," *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. pp. 101–132, 2000.

[274] R. Vasa, L. Hoon, K. Mouzakis, and A. Noguchi, "A preliminary analysis of mobile app user reviews," in *Proceedings of the 24th Australian Computer-Human Interaction Conference*, OzCHI '12. ACM, 2012, pp. 241–244.

[275] N. Viennot, "GitHub - nviennot/playdrone: Google Play Crawler," https://github.com/nviennot/playdrone, 2014.

[276] N. Viennot, E. Garcia, and J. Nieh, "A measurement study of Google Play," in *The 2014 ACM international conference on Measurement and modeling of computer systems*, SIGMETRICS '14. ACM, 2014, pp. 221–233.

[277] L. Vigneri, J. Chandrashekar, I. Pefkianakis, and O. Heen, "Taming the Android appstore: Lightweight characterization of Android applications," *CoRR*, vol. abs/1504.06093, 2015.

[278] L. Villarroel Pérez, "Mining mobile apps reviews to support release planning," Master's thesis, ETSI_Informatica, 2015.

[279] VirusShare, "Virusshare.com," http://virusshare.com/, 2011.

[280] Vision Mobile, “Developer Economics 2013: The tools report,” http://www.visionmobile.com/product/developer-economics-2013-the-tools-report/, 2013.

[281] Vision Mobile, “Developer Economics Q1 2015: State of the Developer Nation,” http://www.visionmobile.com/product/developer-economics-q1-2015-state-developer-nation/, 2015.

[282] A. von Rhein, T. Berger, N. S. Johansson, M. M. Hardø, and S. Apel, “Lifting inter-app data-flow analysis to large app sets,” Fakultät für Informatik und Mathematik, Universität Passau, Tech. Rep., 2015.

[283] P. M. Vu, T. T. Nguyen, H. V. Pham, and T. T. Nguyen, “Mining user opinions in mobile app reviews: A keyword-based approach,” in *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE ’15)*. IEEE, 2015, pp. 749–759.

[284] P. M. Vu, H. V. Pham, T. T. Nguyen, and T. T. Nguyen, “Tool support for analyzing mobile app reviews,” in *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE ’15)*. IEEE, 2015, pp. 789–794.

[285] T. J. Walid Maalej, Maleknaz Nayebi and G. Ruhe, “Toward data-driven requirements engineering,” *IEEE Software Jan/Feb 2016: Special Issue on the Future of Software Engineering*, 2016, to appear.

[286] H. M. Wallach, I. Murray, R. Salakhutdinov, and D. Mimno, “Evaluation methods for topic models,” in *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 1105–1112.

[287] M. Wan, Y. Jin, D. Li, and W. G. Halfond, “Detecting display energy hotspots in Android apps,” in *Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on*. IEEE, 2015, pp. 1–10.

[288] H. Wang, Y. Guo, Z. Ma, and X. Chen, “WuKong: A scalable and accurate two-phase approach to Android app clone detection,” in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ISSTA 2015. ACM, 2015, pp. 71–82.

[289] H. Wang, J. Hong, and Y. Guo, “Using text mining to infer the purpose of permission use in mobile apps,” in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp ’15. ACM, 2015, pp. 1107–1118.

[290] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, “Exploring permission-induced risk in Android applications for malicious application detection,” *IEEE Transactions on Information Forensics and Security*, pp. 1869–1882, 2014.

[291] Y. Wang, J. Zheng, C. Sun, and S. Mukkamala, “Quantitative security risk assessment of Android permissions and applications,” in *Proceedings of the 27th International Conference on Data and Applications Security and Privacy XXVII*, DBSec’13. Springer-Verlag, 2013, pp. 226–241.

[292] M. Wano and J. Iio, "Relationship between reviews at app store and the categories for software," in *Proceedings of the 2014 17th International Conference on Network-Based Information Systems*, NBIS '14. IEEE Computer Society, 2014, pp. 580–583.

[293] T. Watanabe, M. Akiyama, T. Sakai, H. Washizaki, and T. Mori, "Understanding the inconsistencies between text descriptions and the use of privacy-sensitive resources of mobile apps," in *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*. USENIX Association, 2015.

[294] S. Wenxuan and Y. Airu, "Interoperability-enriched app recommendation," in *Data Mining Workshop (ICDMW), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1242–1245.

[295] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[296] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14. ACM, 2014, pp. 38:1–38:10.

[297] Z. Xie and S. Zhu, "AppWatcher: Unveiling the underground market of trading mobile app reviews," in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '15. ACM, 2015, pp. 10:1–10:11.

[298] W. Xu, F. Zhang, and S. Zhu, "Permlyzer: Analyzing permission usage in Android applications," in *24th IEEE International Symposium on Software Reliability Engineering*, ISSRE '13. IEEE, 2013, pp. 400–410.

[299] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, "AppContext: Differentiating malicious and benign mobile app behaviors using context," in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, 2015, pp. 303–313.

[300] Y. Yang, J. Stella Sun, and M. W. Berry, "APPIC: Finding the hidden scene behind description files for Android apps," Dept. of Electrical Engineering and Computer Science University of Tennessee, Tech. Rep., 2014.

[301] yannrichet, "GitHub – yannrichet/jmathplot: Java interactive 2D and 3D plots (no OpenGL)," https://github.com/yannrichet/jmathplot, 2016.

[302] P. Yin, P. Luo, W.-C. Lee, and M. Wang, "App recommendation: A contest between satisfaction and temptation," in *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13. ACM, 2013, pp. 395–404.

[303] F. Zhang, H. Huang, S. Zhu, D. Wu, and P. Liu, "Viewdroid: Towards obfuscation-resilient mobile application repackaging detection," in *Proceedings of the 7th ACM Conference on Security and Privacy in Wireless & Mobile Networks*, WiSec '14, 2014, pp. 25–36.

[304] H. Zhang, L. Gong, and S. Versteeg, “Predicting bug-fixing time: An empirical study of commercial software projects,” in *Proceedings of the 2013 International Conference on Software Engineering*, ICSE ’13. IEEE Press, 2013, pp. 1042–1051.

[305] M. Zhang, Y. Duan, Q. Feng, and H. Yin, “Towards automatic generation of security-centric descriptions for Android apps,” in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS ’15. ACM, 2015, pp. 518–529.

[306] P. Zheng, Y. Zhou, M. R. Lyu, and Y. Qi, “Granger causality-aware prediction and diagnosis of software degradation,” in *IEEE International Conference on Services Computing (SCC’14)*, 2014, pp. 528–535.

[307] N. Zhong and F. Michahelles, “Google Play is not a long tail market: An empirical analysis of app adoption on the Google Play app market,” in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC ’13. ACM, 2013, pp. 499–504.

[308] Y. Zhou and X. Jiang, “Dissecting Android malware: Characterization and evolution,” in *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012, pp. 95–109.

[309] Y. Zhou, L. Wu, Z. Wang, and X. Jiang, “Harvesting developer credentials in Android apps,” in *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec ’15. ACM, 2015, pp. 23:1–23:12.

[310] H. Zhu, C. Liu, Y. Ge, H. Xiong, and E. Chen, “Popularity modeling for mobile apps: A sequential approach,” *IEEE Transactions on Cybernetics*, vol. 45, no. 7, pp. 1303–1314, 2014.

[311] H. Zhu, H. Cao, E. Chen, H. Xiong, and J. Tian, “Exploiting enriched contextual information for mobile app classification,” in *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 2012, pp. 1617–1621.

[312] H. Zhu, E. Chen, H. Xiong, H. Cao, and J. Tian, “Mobile app classification with enriched contextual information,” *IEEE Transactions on Mobile Computing*, vol. 13, no. 7, pp. 1550–1563, 2014.

[313] H. Zhu, H. Xiong, Y. Ge, and E. Chen, “Ranking fraud detection for mobile apps: A holistic view,” in *Proceedings of the 22Nd ACM International Conference on Conference on Information & Knowledge Management*, CIKM ’13. ACM, 2013, pp. 619–628.

[314] H. Zhu, H. Xiong, Y. Ge, and E. Chen, “Mobile app recommendations with security and privacy awareness,” in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’14. ACM, 2014, pp. 951–960.

[315] H. Zhu, H. Xiong, Y. Ge, and E. Chen, “Discovery of ranking fraud for mobile apps,” *Knowledge and Data Engineering, IEEE Transactions on*, vol. 27, no. 1, pp. 74–87, 2015.

[316] J. Zhu, Z. Guan, Y. Yang, L. Yu, H. Sun, and Z. Chen, “Permission-based abnormal application detection for Android,” in *Proceedings of the 14th International Conference on Information and Communications Security*, ICICS’12. Springer-Verlag, 2012, pp. 228–239.

Appendix A

Closely Related Literature

The following literature is important to the field of App Store Analysis, yet itself does not meet our exact definition of App Store Analysis. Nevertheless, since this work meets aspects of the definition, we regard it as closely related. We do not claim to comprehensively survey this literature, but provide it to add context to the App Store Analysis literature discussed in Chapter 2.

A.1 User Surveys and Studies

There is a cross section of App Analysis studies which survey or study user behaviour and feedback, but the information is not specific to observed apps, and is therefore not combined with technical information. These studies are important to the field of App Store Analysis and so are included here.

In 2011 Böhmer et al. [33] studied 4,100 Android users for app usage statistics. This was done using `AppSensor`, an application that monitors the usage of other apps on an Android device. They found that the average application usage session was less than 72 seconds long, and that smartphones were used for almost 60 minutes every day. The type of application was found to differ between times of day, such as news applications in the morning and games at night. The exception to this rule was communication apps, which were used throughout the day.

In 2012 Ferreira et al. [76] surveyed 4,035 Android user charging habits, using an app to record their behaviour. Lin et al. [160] conducted a survey on 179 Android users, that asked about their expectations of the purpose and sensitive data handling of apps. They found that the problem of apps not meeting expectations or utilising sensitive data unexpectedly was prevalent, and outlined potential store interface changes to rectify the issue. Rein and Münch [234] carried out a user study involving mock purchasing for planned app features, in order to determine both the priority and ideal pricing for the features. In 2013 Oh et al. [212] surveyed 100 app users and found that the users were more likely to take a passive approach and delete apps, rather than reviewing or contacting developers, but when users took an active approach, reviewing was the most popular approach. In 2014 Tan et al. [263] surveyed users and developers of the Apple App Store, regarding the iOS permission request explanation feature. The feature was infrequently used, but the survey found that users would be significantly more likely to accept a permissions request if an explanation was given.

In 2015 Lim et al. [155] surveyed app users from 15 countries to understand how usage of apps and app stores differs by region. They found that behaviour differed significantly by region in many regards. In Eastern regions such as China and India, a greater proportion of users participated in recommendation and rating of apps, almost 4 times the proportion of Western users. Additionally, the survey found that app abandonment due to issues was higher on average in Brazil and the UK, and lower on average in Japan and France, indicating that any differences were affected by more than global region. It is a unique study as it gathered user information regarding multiple app stores across a large number of global app users: the focus is on usage, not on apps, yet the authors identified actionable findings for app developers.

A.2 Related Security

I present some of the key app security studies that do not perform App Store Analysis, but that influenced some of the papers described in Section 2.10.

Enck et al. [74] introduced `Kirin`, an Android app certification tool for flagging potential malware using a set of rules. In 2010 Enck et al. introduced `TaintDroid` [73], a tool for tracking the flow of sensitive information within an Android app. `TaintDroid` was one of the first static analysis tools for Android and was built on extensively in subsequent work. Another information flow extraction tool was created by Arzt et al. [11] in 2014, called `FlowDroid`. This tool statically analyses information flow to find all possible flows.

Some authors have used sets mined from Google Play as benign app sets to test against known malware: Xu et al. [298], Rastogi et al. [232], Jing et al. [132], Arp et al. [10], Wang et al. [290], Liu and Liu [171], Roy et al. [236] and Khanmohammadi et al. [143]. Ho et al. [115] used the top 10 most popular apps in each category as a benign set, upon which to test their framework for root kit exploit containment.

Other authors have used sets mined from app stores to test their tools on large real-world datasets: Barrera et al. [18], Jeon et al. [128], Grace et al. [98], Crussell et al. [63, 64], Ravindranath et al. [233], von Rhein et al. [282], Li et al. [153], Huang et al. [123], Cen et al. [43], Liu et al. [169] and Bastani et al. [20].

A.3 Reports

Initial studies such as the 2010 work by Sharma et al. [254] evaluated the size and growth of the apps market up to the time. In 2011 Butler [38] conducted a study on the Android system, highlighting how it was changing mobile development by enabling people with no prior development experience to release an app. In 2012 Shuler [255] published a report on the Apple App Store Education category, comparing it with their previous study in 2009. They found that over 72% of the top-selling apps in this category targeted children aged 10 or below, a number that had significantly increased from 47% in 2009. Additionally, the average price of an app had risen by 1 USD since 2009, and the majority of top Education developers in 2012 had not been present in 2009.

The 2013 report by Vision Mobile [280] on app industry monetary value and growth found that 72% of developers were dedicated to Android. iOS and Android developers earned on average double that of developers of other platforms, and iOS was considered the highest priority platform. As of 2013, iOS, Android and Blackberry were the leading platforms, despite Blackberry's decline, and the launch of the prospect Windows Phone Store in late 2010. Vision Mobile have released yearly reports since 2012 on aspects such as developer share, industry revenue and growth. The organisation gathers information by surveying developers worldwide.

A.4 Mining Tools

Due to the plethora of analysis and research opportunities presented by app store data, and indeed also due to the difficulties involved with mining app stores, several mining tools have been published.

In 2013, Bakar et al. [17] published `OSSGrab` which mines `HTML` pages from Google Play. The tool was built in order to facilitate their app permissions study [15]. In 2014, Viennot et al. introduced the `PlayDrone` Google Play crawler [275], to facilitate their large scale API study [276].

The Android Malware Genome Project [308] is a popular source of malware applications for testing security tools. In 2015 Krutz et al. [149] made available a dataset containing 1,179 open source applications.

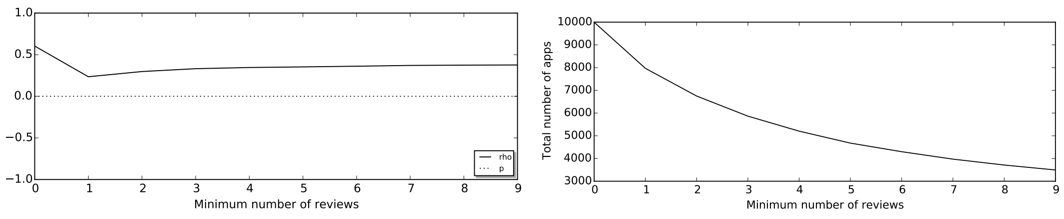
Appendix B

App Feature Extraction using Topic Models

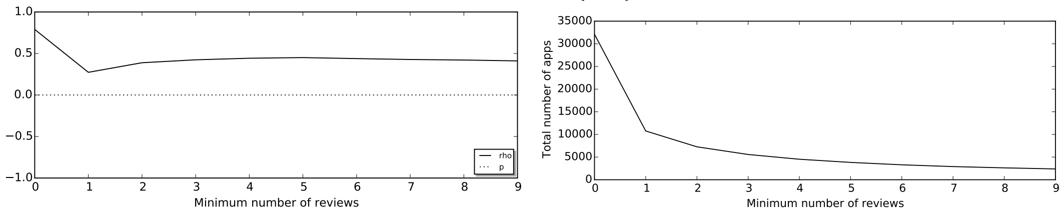
Here we include supplementary result graphs for the Chapter 4 study on app feature extraction using topic models.

B.1 Sliding Window Rating/Download Correlation Analysis Results

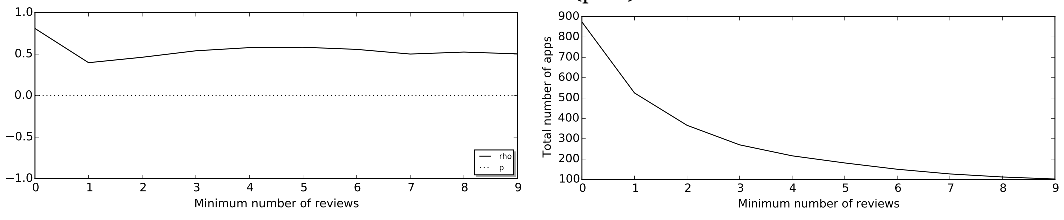
Figure B.1: **Sliding Window Rating/Download Correlation Analysis results**



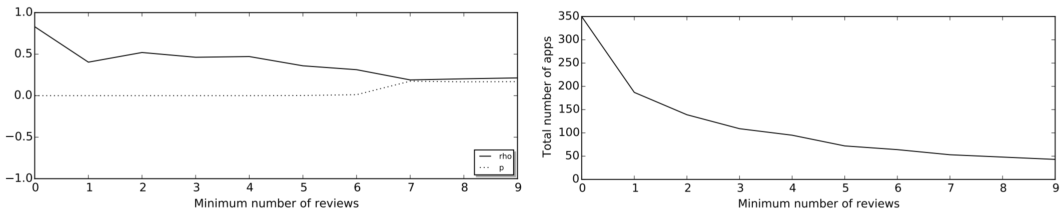
All categories (free)



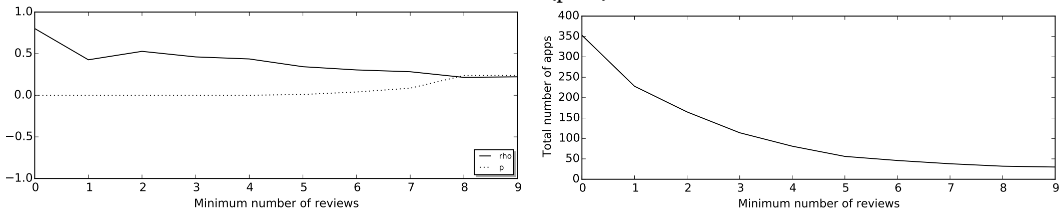
All categories (paid)



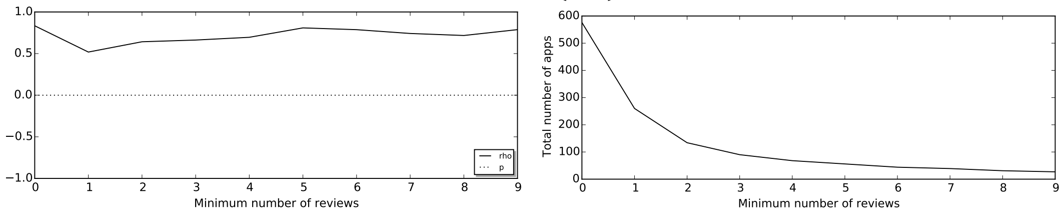
Business (free)



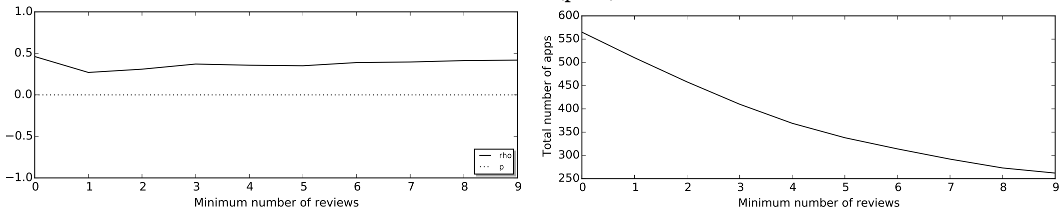
Business (paid)



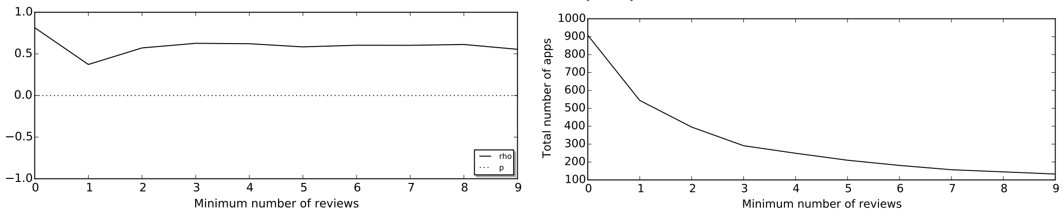
Education (free)



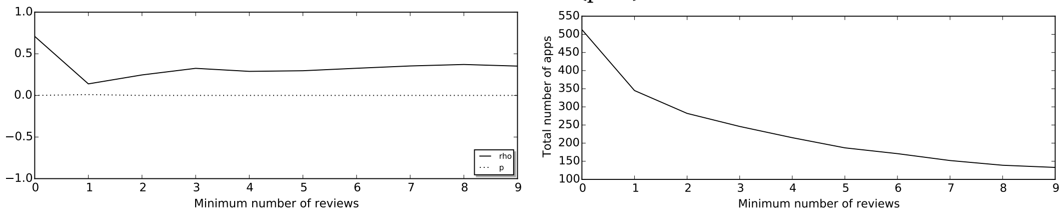
Education (paid)



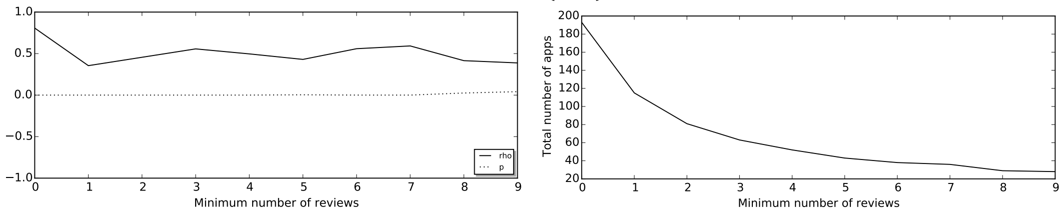
Entertainment (free)



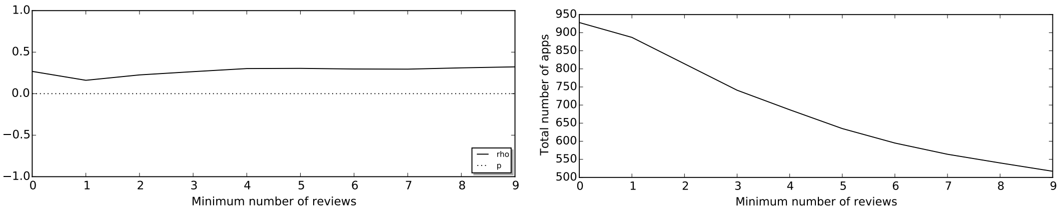
Entertainment (paid)



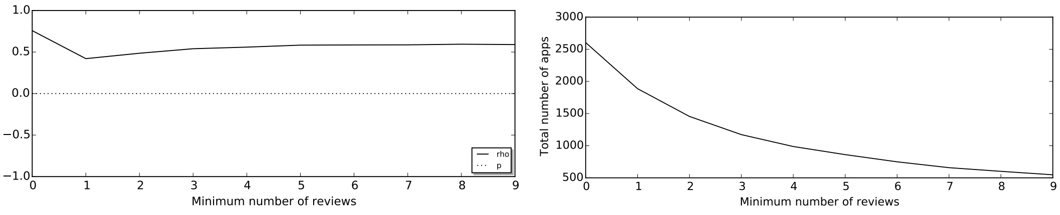
Finance (free)



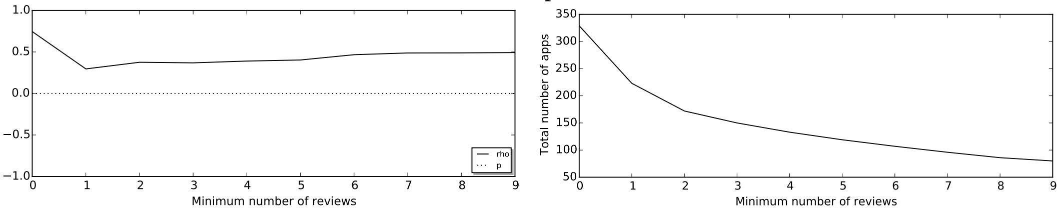
Finance (paid)



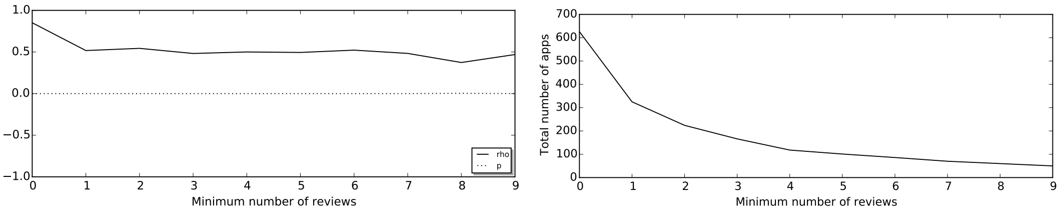
Games (free)



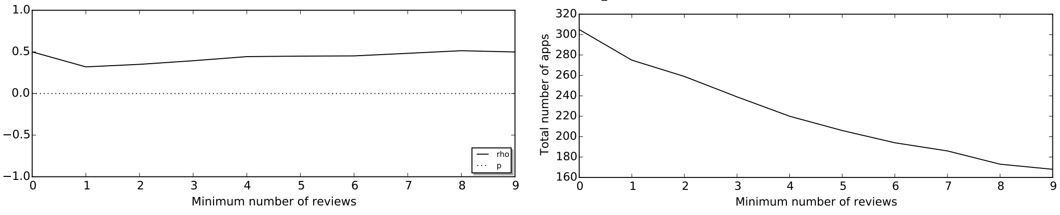
Games (paid)



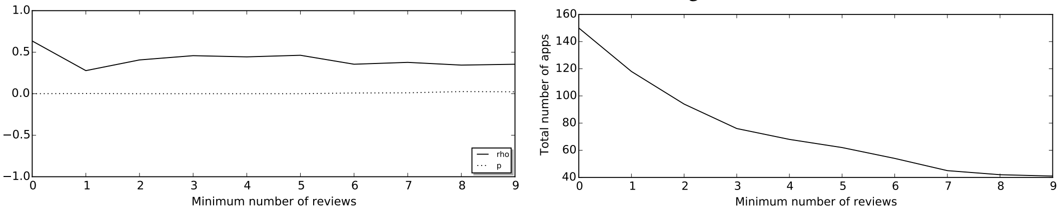
Health & Wellness (free)



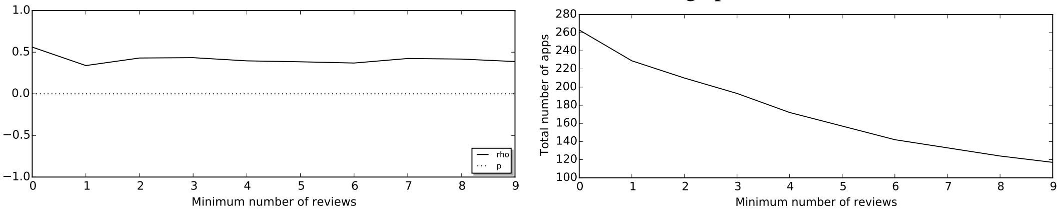
Health &Wellness (paid)



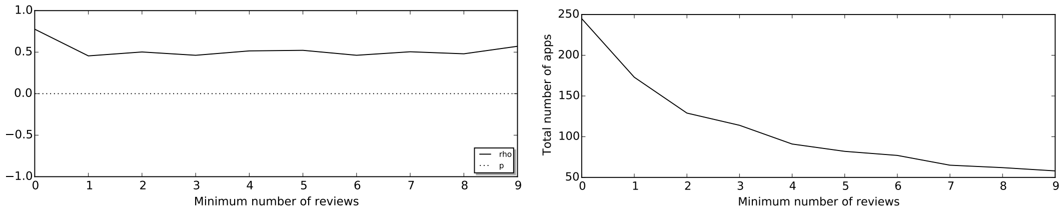
Instant Messaging & Social Networking (free)



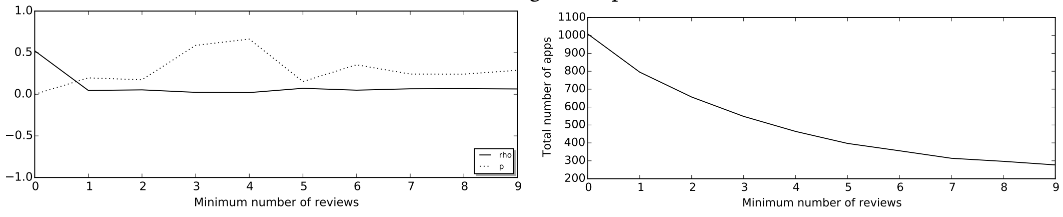
Instant Messaging & Social Networking (paid)



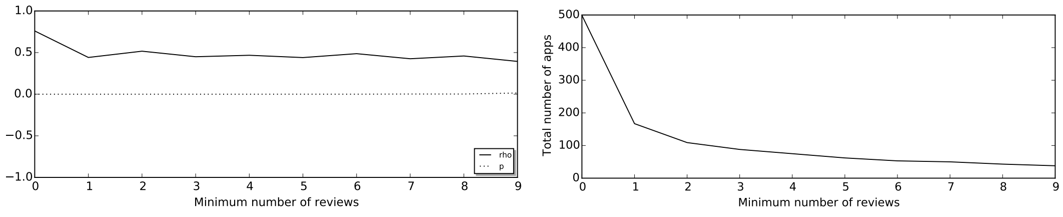
Maps & Navigation (free)



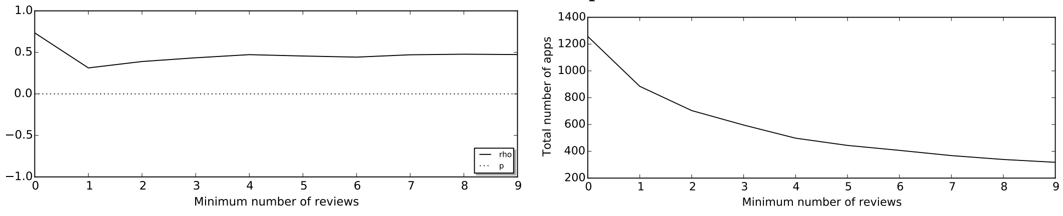
Maps & Navigation (paid)



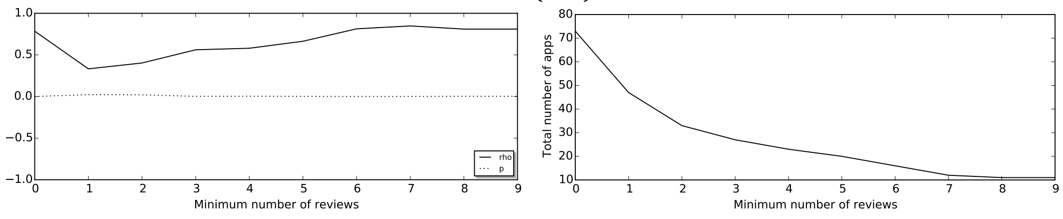
Music & Audio (free)



Music & Audio (paid)



News (free)



News (paid)

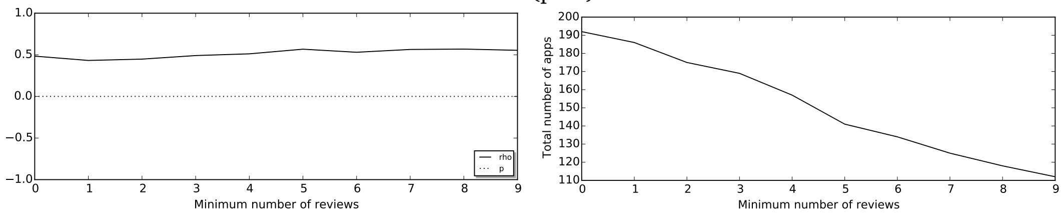


Photo & Video (free)

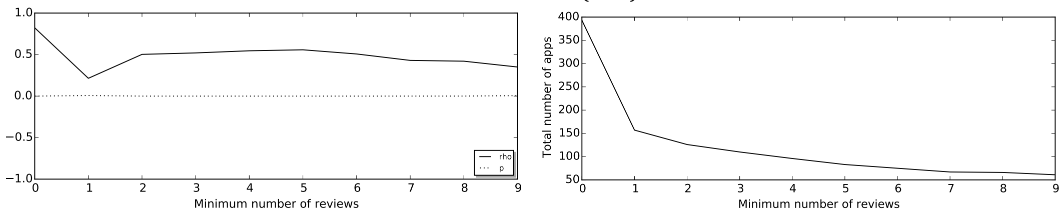
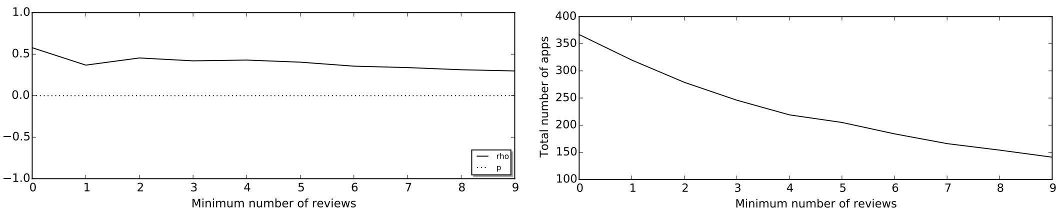
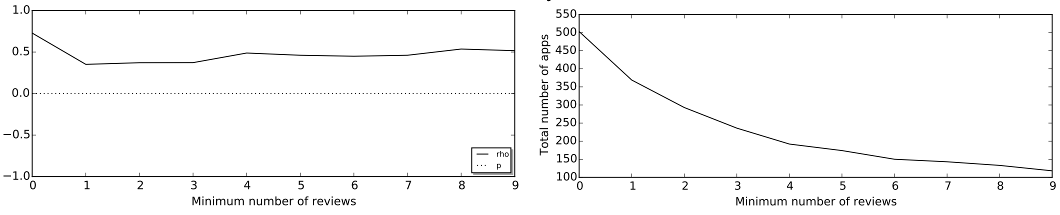


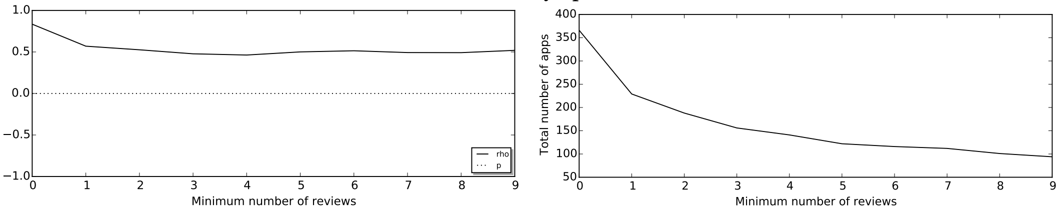
Photo & Video (paid)



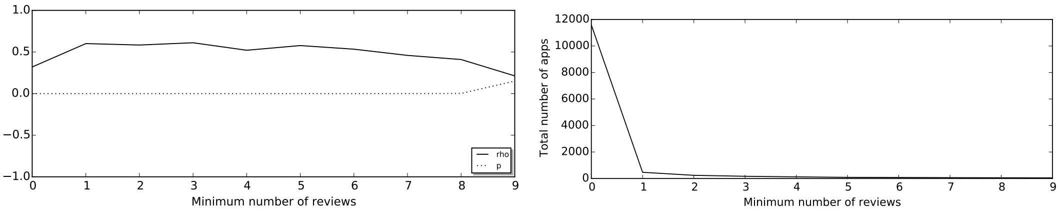
Productivity (free)



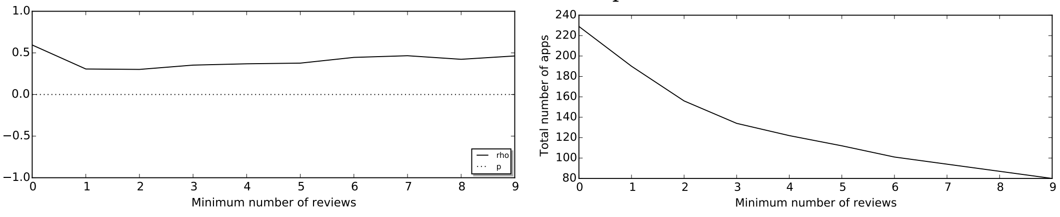
Productivity (paid)



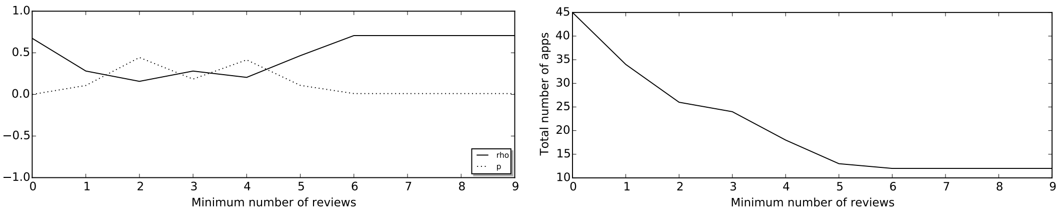
Reference & eBooks (free)



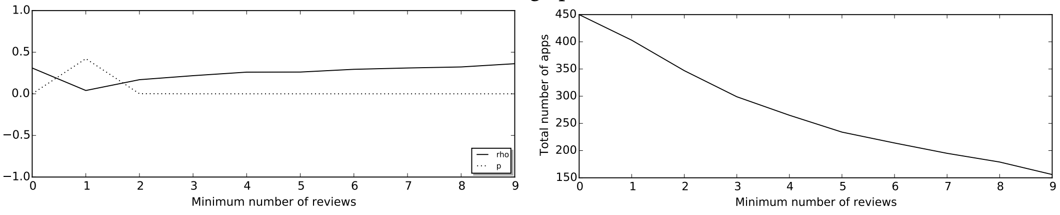
Reference & eBooks (paid)



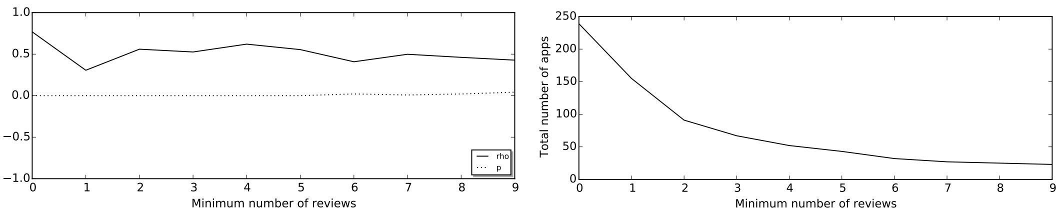
Shopping (free)



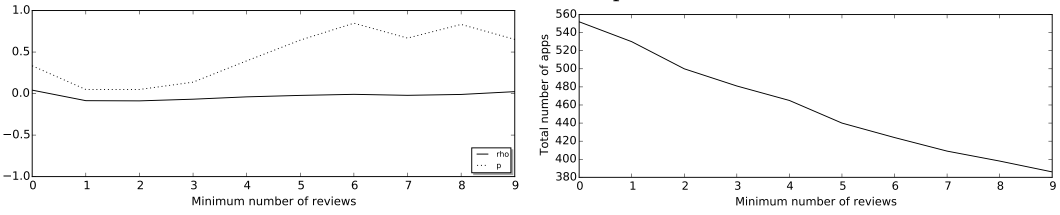
Shopping (paid)



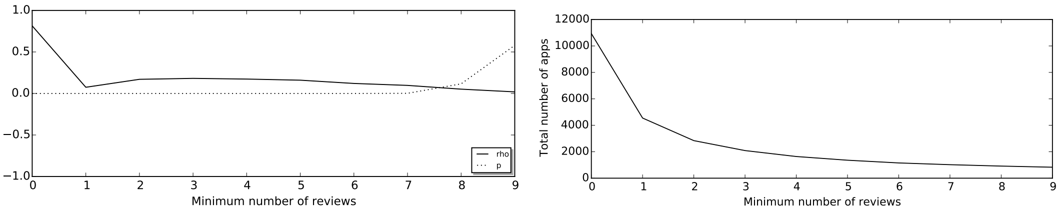
Sports & Recreation (free)



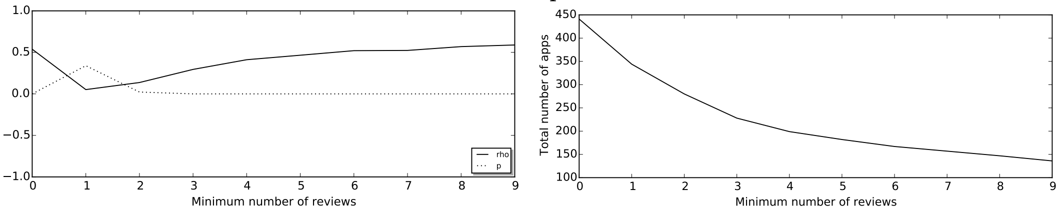
Sports & Recreation (paid)



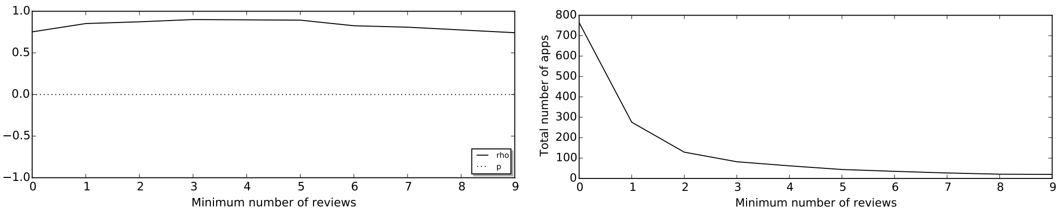
Themes (free)



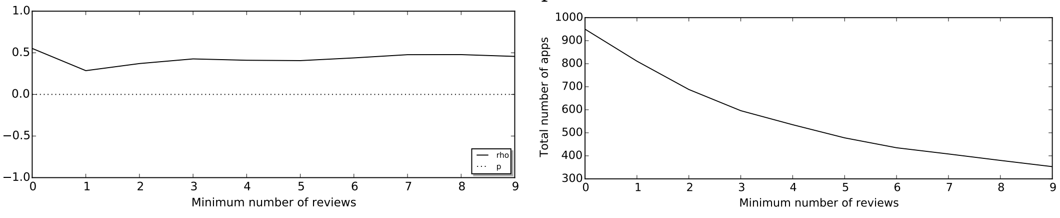
Themes (paid)



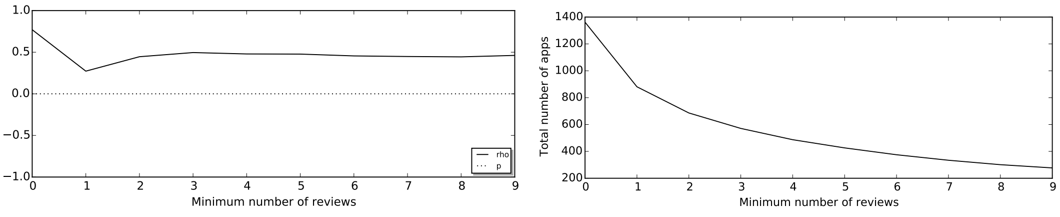
Travel (free)



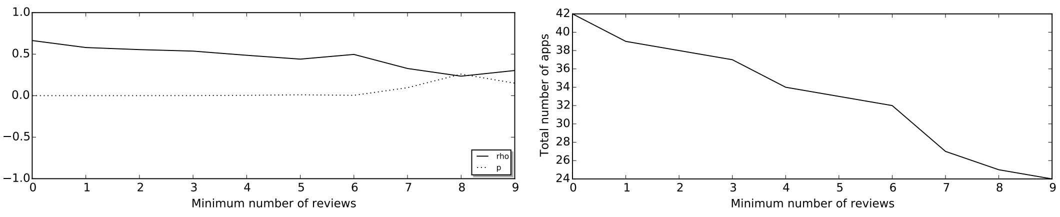
Travel (paid)



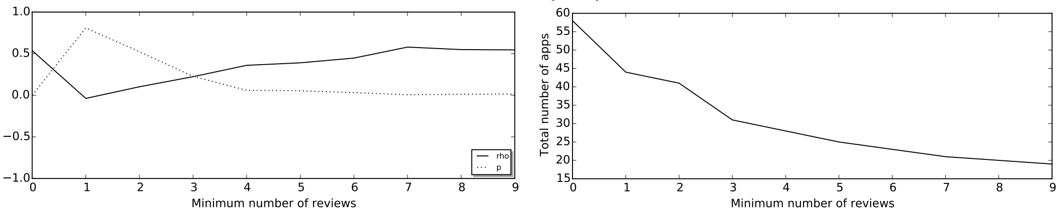
Utilities (free)



Utilities (paid)



Weather (free)



Weather (paid)

Appendix C

What are the Features that are Added to Impactful Releases?

C.1 Supplementary Category-Specific Topic Results

Here we include supplementary results for the Section 7.5 study on features that are specific to impactful releases.

Table C.1: **RQ1 + RQ2: Category topics that changed at release time**
NI: non-impact releases
I: impactful releases
N: releases that decreased rating
P: releases that increased rating
Top: proportion of topics that were added in I/P releases, and taken away or not added to NI/N releases

Bottom: proportion of topics that were removed or not changed in I/P releases, but were added in NI/N releases

NI	I	Topic	N	P	Topic
-0.14	0.76	review help really like nice	-1.33	0.00	copy paste clipboard feature design
-0.85	0.00	import export file csv html	-1.33	0.00	improved access feature early time
-0.28	0.51	ajout plus mail de pour	0.00	1.20	android wear watch smartwatch support
0.28	-0.51	de le pour correction vous	0.44	-0.60	image gallery preview load crop
0.28	-0.76	copy paste clipboard feature design	4.42	2.40	permission read used storage external
3.84	1.78	bible verse reading chapter remove	3.10	0.00	bible verse reading chapter remove

Books & Reference

NI	I	Topic	N	P	Topic
-2.63	3.85	expense report transaction income bill	-9.09	0.00	orientation landscape portrait tablet mode
0.00	3.85	contact list phone using copy	-9.09	0.00	display option screen title default
0.00	3.85	message notification send sending receive	-9.09	0.00	tablet phone available mobile optimised
1.32	0.00	device range mode wider extended	0.00	6.67	log change multiple list highlight
2.63	0.00	task reminder time set view	9.09	0.00	expense report transaction income bill
2.63	0.00	bug fix minor space file	9.09	0.00	backup restore cloud google drive

Business

NI	I	Topic	N	P	Topic
0.00	2.78	tablet phone available mobile optimised	-2.86	2.70	book reading comic page reader
-0.54	1.39	document pdf text multiple file	-2.86	2.70	download downloads file downloaded downloading
-0.54	1.39	improved handling handle startup performance	-2.86	0.00	setting layer color menu background
0.54	0.00	close force fixed caused compatible	2.86	0.00	document pdf text multiple file
1.63	0.00	faster load loading scrolling speed	2.86	0.00	improved handling handle startup performance
0.54	-1.39	news feed article section read	5.71	0.00	tablet phone available mobile optimised

Comics

NI	I	Topic	N	P	Topic
-0.16	1.22	new feature create creating geotag	-2.06	0.00	feature allows requested announce excited
-0.48	0.82	device android lollipop work mobile	-2.06	0.00	feedback suggestion comment send app
-0.32	0.82	chinese language simplified traditional japanese	-1.03	0.68	email send comment request suggestion
0.16	-0.82	menu button setting section parent	2.06	0.00	free feel email contact app
0.16	-0.82	feedback suggestion comment send app	2.06	0.00	android lollipop compatibility kitkat pre
0.32	-0.82	card credit payment account transaction	2.06	-0.68	language spanish french german portuguese

Communication

NI	I	Topic	N	P	Topic
-0.64	1.82	bug fix minor doe symptom	0.00	12.90	weve easier making squashed feedback
-0.64	1.82	language franais deutsch english meeting	-8.33	-3.23	word dictionary letter day offline
0.00	1.82	permission read used storage external	-4.17	0.00	building build city quest decoration
0.64	-1.82	fish fishing scene new implemented	4.17	0.00	bug fix minor doe symptom
0.64	-1.82	building build city quest decoration	4.17	0.00	let know review store like
1.27	-3.64	weve youll just got thing	4.17	0.00	brush color pen drawing paint

Education

NI	I	Topic	N	P	Topic
-0.28	0.63	baby fun accessory make clothes	0.00	1.20	key keyboard input layout press
-0.14	0.63	free user chaatz number india	0.00	1.20	free user chaatz number india
0.00	0.63	removed permission unnecessary needed unused	-0.65	0.00	android wear watch smartwatch support
0.42	-0.31	support chromecast stream beta video	0.65	0.00	resolution high device higher screen
0.28	-0.63	notification push receive alert receiving	0.65	-0.60	spider man including event power
0.55	-0.94	movie show content browse film	1.31	0.00	removed permission unnecessary needed unused

Entertainment

NI	I	Topic	N	P	Topic
-0.48	1.41	expense report transaction income bill	-0.68	1.44	app feedback release improve experience
-0.36	0.56	version til med role fejlrettelser	-0.68	0.48	application launch implemented launching program
-0.24	0.56	support additional functionality tablet operation	-0.68	0.48	app includes using version better
0.24	-0.28	update future big near hand	0.68	-0.96	calculator tax rate calculation loan
0.12	-0.56	aplikacji bdw poprawki dla wersji	1.37	-0.48	mobile app customer banking service
0.24	-0.56	offer app working continuously better	0.68	-1.44	aplikacji bdw poprawki dla wersji

Finance

NI	I	Topic	N	P	Topic
-0.64	4.44	app feedback release improve experience	-14.29	2.63	flight hotel booking app book
-0.64	2.22	run smoother make faster content	-14.29	2.63	improved touch suit partner play
-0.64	2.22	weve youll just got thing	-14.29	2.63	game mini house run smooth
2.55	0.00	love feedback hear wed suggestion	0.00	2.63	brush color pen drawing paint
1.27	-2.22	hard working weve work bring	14.29	0.00	google drive cloud integration integrated
1.27	-2.22	app way need update problem	14.29	-5.26	goal tracking track progress activity

Health & Fitness

NI	I	Topic	N	P	Topic
-0.51	1.06	support device android version initial	0.00	3.45	para melhorias com mais nova
0.00	1.06	version doubled displacement app randomization	0.00	3.45	issue fixed crashing recovery related
0.00	1.06	fix bug better localization introduced	0.00	3.45	release note detail github stability
0.51	-1.06	refresh auto pull change map	1.54	0.00	fix bug better localization introduced
0.51	-1.06	language spanish french german portuguese	1.54	0.00	support device android version initial
1.02	-1.06	scan code scanning barcode scanner	4.62	0.00	notification push receive alert receiving

Libraries & Demo

NI	I	Topic	N	P	Topic
-0.61	1.49	online lot avatar description table	-8.33	0.00	youre looking know having people
0.00	1.49	article reading read save link	-4.17	0.00	location current based network wont
0.00	1.49	version previous upgrading cancelling mixcloud	-4.17	0.00	version algorithm major photo recovery
0.61	-2.99	youre looking know having people	4.17	0.00	weather widget forecast temperature current
0.61	-2.99	alarm time mode timer sound	4.17	0.00	backup restore cloud google drive
0.61	-2.99	recipe like kitchen cooking feature	4.17	-2.33	weve youll just got thing

Lifestyle

NI	I	Topic	N	P	Topic
-0.85	1.33	live stream watch video broadcast	-1.94	0.82	sticker testing line bot chat
-0.42	0.89	card external storage support file	-1.94	0.82	video youtube player playing thumbnail
-0.64	0.44	language swedish turkish translation polish	-0.97	0.82	para versin la que mejoras
0.42	-0.44	fixed working properly premium older	0.97	-0.82	fixed bug crash opening navigationdrawer
0.21	-0.89	fix bug minor extended reaction	1.94	0.00	album music playlist track artist
0.21	-0.89	add improve refine ons euro	1.94	0.00	support display click download issue

Media & Video

NI	I	Topic	N	P	Topic
-0.24	1.71	language spanish french german portuguese	-8.70	0.00	sticker testing line bot chat
0.00	0.85	addition available club functionality reference	-2.17	0.00	android device running compatible lollipop
0.00	0.85	support android cloud guideline risk	0.00	2.08	support android cloud guideline risk
0.72	0.00	weve youll just got thing	0.72	-1.04	stuff lot cool great check
0.48	-0.43	social share network medium sharing	0.72	-1.04	import export file csv html
0.48	-0.43	backup restore cloud google drive	0.72	-1.04	health improvement updated check doctor

Medical

NI	I	Topic	N	P	Topic
-1.01	2.00	version cheating run easter continue	-12.50	0.00	version previous upgrading cancelling mixcloud
-0.50	2.00	crash fixed startup causing mark	-12.50	0.00	new bamboo dozen enhanced size
-0.50	2.00	fast news forward slow agent	0.00	2.38	album music playlist track artist
0.50	-2.00	design material layout element original	12.50	0.00	issue poker thanks playing governor
0.50	-2.00	lock password security pin apps	12.50	0.00	page saved list bookmark search
0.50	-2.00	question answer quiz correct ask	12.50	0.00	version cheating run easter continue

Music & Audio

NI	I	Topic	N	P	Topic
0.00	1.99	hard working weve work bring	-3.61	0.00	article reading read save link
0.00	1.00	aplikacji bdw poprawki dla wersji	0.00	3.39	hard working weve work bring
-0.30	0.50	make better easier experience faster	-1.20	0.85	live stream watch video broadcast
0.30	-0.50	love feedback hear wed suggestion	2.41	0.00	news feed article section read
0.30	-0.50	story directly deep feed include	2.41	0.00	aplikacji bdw poprawki dla wersji
0.15	-1.49	article reading read save link	2.41	-0.85	feature app latest free news

News & Magazines

NI	I	Topic	N	P	Topic
-0.19	0.56	theme icon latest rate like	-8.48	-0.53	facebook follow like twitter news
-0.06	0.56	color background text picker scheme	-8.48	-1.05	wallpaper live muzei beautiful picker
-0.44	0.14	google play hangout gmail theme	-0.61	1.05	icon request activity wallpaper thank
0.31	-0.28	icon white text background setting	0.30	-0.53	resolution high device higher screen
0.31	-0.70	watch face wear android phone	1.21	-0.79	icon activity request launcher fixed
1.45	-4.51	wallpaper live muzei beautiful picker	9.09	0.53	sound thanks rate quality really

Personalisation

NI	I	Topic	N	P	Topic
-0.61	2.44	facebook twitter share sharing instagram	-5.26	4.55	theme dark light material design
-0.61	2.44	android support lollipop samsung device	-5.26	0.00	got just weve awesome perfect
-0.61	2.44	list support update menu editor	-5.26	0.00	look feel fresh modern refreshed
1.23	-2.44	device samsung htc nexus support	5.26	0.00	list support update menu editor
1.84	-2.44	image new filter effect tool	5.26	0.00	button load save toolbar using
0.61	-4.88	mode device camera setting focus	5.26	0.00	change minor api website visit

Photography

NI	I	Topic	N	P	Topic
-0.34	2.15	language translation translate thanks error	-3.85	2.99	file folder zip failure open
-2.03	0.00	event calendar day agenda exchange	-3.85	2.99	alarm time mode timer sound
-0.68	1.08	android support lollipop lexicon licensing	-3.85	1.49	backup restore cloud google drive
0.68	-2.15	setting default changed disabled shift	3.85	0.00	version tool minor added wind
0.68	-2.15	date view widget day month	3.85	0.00	flash add widget version language
1.02	-3.23	apps device support setting switching	3.85	0.00	fix bug error separately device

Productivity

NI	I	Topic	N	P	Topic
0.00	20.00	product shopping page talk shop	-14.29	0.00	make sure update latest thanks
-2.70	4.00	day valentine love special heart	0.00	5.56	policy google privacy term analytics
-0.90	4.00	policy google privacy term analytics	0.00	5.56	offer app working continuously better
0.90	0.00	fix available link clicking grand	28.57	16.67	product shopping page talk shop
1.80	0.00	item inventory price rare sell	14.29	0.00	und der die mit auf
1.80	-4.00	facebook twitter share sharing instagram	14.29	0.00	sign account using single register

Shopping

NI	I	Topic	N	P	Topic
-0.85	2.70	group member create join chat	-9.09	0.00	weve easier making squashed feedback
0.00	2.70	fixed issue known startup way	0.00	3.85	fixed issue known startup way
0.00	2.70	search favourite deal nearby local	0.00	3.85	tab custom chrome galaxy specific
0.85	-2.70	store apps detail play check	9.09	0.00	group member create join chat
0.85	-2.70	weve easier making squashed feedback	9.09	0.00	search favourite deal nearby local
1.69	-2.70	notification push receive alert receiving	9.09	0.00	search result recent suggestion radius

Social

NI	I	Topic	N	P	Topic
-0.09	0.84	tournament player online win ranking	0.00	1.51	live stream watch video broadcast
0.00	0.63	notification sound setting alert vibration	-0.94	0.38	version doubled displacement app randomization
0.00	0.63	news feed article section read	-0.94	0.38	language spanish french german portuguese
0.36	0.00	team match cheer vault dream	0.47	-0.75	round course shot golf hole
0.18	-0.21	fixed bug crash opening navigationdrawer	0.47	-0.75	player stats single profile starting
0.18	-0.21	notification user showing event loading	1.88	0.00	alarm time mode timer sound

Sports

NI	I	Topic	N	P	Topic
-0.17	0.66	bug minor fix single end	-0.58	1.05	map location gps data zoom
-0.33	0.44	card external storage support file	-1.16	0.35	import export file csv html
-0.08	0.66	app apps clean installed option	0.00	1.05	app apps clean installed option
0.17	-0.44	fix bug splash prevent process	0.58	-0.35	calculator tax rate calculation loan
0.08	-0.66	device android compatibility older newer	1.16	0.00	language english spanish localization french
0.75	-0.44	language swedish turkish translation polish	1.16	0.00	data weight device setting body

Tools

NI	I	Topic	N	P	Topic
0.00	2.60	flash add widget version language	-0.88	2.58	route map location search address
0.00	1.86	booking app taxi favourite address	-0.88	1.94	map location gps data zoom
-0.17	1.12	route map location search address	-0.88	0.65	flight hotel booking app book
0.17	-0.37	removed permission unnecessary needed unused	1.75	0.00	time hour format zone display
0.17	-0.37	device samsung htc nexus support	2.63	0.65	ticket purchase app send email
1.02	0.00	bus stop journey station line	6.14	0.00	flash add widget version language

Transport

NI	I	Topic	N	P	Topic
0.00	3.54	start stop data gps directly	-9.16	-0.61	better performance faster optimisation enhancement
-1.33	-0.37	value select tip graph altitude	-2.96	0.61	map location gps data zoom
-1.45	-0.75	screen selected current tapping displayed	-3.50	0.00	android app time improved present
1.09	0.19	log available view device connected	4.31	0.61	trip travel destination time guide
0.97	0.00	booking app taxi favourite address	4.85	0.61	start stop data gps directly
0.85	-0.93	option optimization removed temporarily shadow	5.12	0.61	app inside update mean just

Travel & Local

NI	I	Topic	N	P	Topic
-1.54	4.35	option menu hide using context	-28.57	6.25	weather widget forecast temperature current
-4.62	0.00	fix weather version display option	-14.29	6.25	language swedish turkish translation polish
0.00	4.35	traffic information map live alert	0.00	6.25	plane game free pilot waiting
1.54	-4.35	added physical global turned improved	0.00	6.25	traffic information map live alert
1.54	-4.35	change minor api website visit	0.00	6.25	photo app child kid year
3.08	-4.35	weather widget forecast temperature current	14.29	0.00	design material flat modern guideline

Weather

Table C.-4: **RQ1 + 2: Games category topics that changed at release time**
NI: non-impact releases
I: impactful releases
N: releases that decreased rating
P: releases that increased rating
Top: proportion of topics that were added in I/P releases, and taken away or not added to NI/N releases

Bottom: proportion of topics that were removed or not changed in I/P releases, but were added in NI/N releases

NI	I	Topic	N	P	Topic
-0.65	1.26	key keyboard input layout press	-3.92	0.93	guild battle raid event hero
-0.65	1.26	weapon new mission gun sniper	-1.96	0.93	graphic control environment realistic driving
-0.33	1.26	clan battle attack unit troop	-1.96	0.93	new random item character tool
0.16	-1.26	new ton display call weekend	1.96	0.00	ad removed remove banner interstitial
0.33	-1.89	hero dungeon equipment skill item	1.96	-0.93	alliance war battle unit attack
1.14	-2.52	power level ups unlock time	1.96	-0.93	level pack designed newly fun

Action

NI	I	Topic	N	P	Topic
0.00	1.46	pack available free expansion includes	0.00	2.47	playing thank description date minor
-0.45	0.73	challenge reward win daily earn	-3.57	-1.23	play google service achievement leaderboards
-0.45	0.73	problem issue updated help contact	-1.79	0.00	day valentine love special heart
0.23	-0.73	fix option improvement bug screen	1.79	0.00	phone support model android send
2.50	0.73	adventure level island explore come	1.79	0.00	day week month view calendar
0.68	-2.19	play google service achievement leaderboards	1.79	0.00	problem issue updated help contact

Adventure

NI	I	Topic	N	P	Topic
-1.47	1.89	level candy latest sweet download	-9.09	0.00	leaderboard achievement leaderboards game score
-0.49	1.89	review help really like nice	-9.09	0.00	friend share facebook connect chip
-0.49	1.89	new horse set stable need	-9.09	0.00	version upload delete folder advanced
0.49	-1.89	level difficulty adjusted adjustment adjust	0.00	2.38	add improve refine ons euro
0.49	-1.89	mode immersive night preview endless	9.09	-2.38	christmas holiday winter gift santa
0.49	-1.89	leaderboard achievement leaderboards game score	9.09	-2.38	feedback thanks based coming thank

Arcade

NI	I	Topic	N	P	Topic
-1.81	0.00	related stability ad option use	-4.35	0.00	device samsung htc nexus support
-0.23	1.44	size reduced apk reduce smaller	-2.17	2.15	leaderboard achievement leaderboards game score
-0.23	0.72	card box draw feature multiple	-2.17	1.08	feature available update august timing
0.23	-0.72	new brand reaction prank dbz	2.17	0.00	added device bacon restored showroom
0.45	-0.72	tournament player online win ranking	4.35	0.00	puzzle pack update piece daily
0.45	-0.72	ajout plus mail de pour	4.35	0.00	support device better minor updated

Board

NI	I	Topic	N	P	Topic
0.00	2.19	nng thm khi hin tnh	-1.67	1.30	leaderboard achievement leaderboards game score
0.00	1.46	policy google privacy term analytics	-1.67	1.30	option added advertising modified disabling
-0.65	0.73	related stability ad option use	-1.67	0.00	game played brain trivia organised
0.22	-0.73	support android update hoptime basiskatalog	1.67	0.00	fix option improvement bug screen
0.22	-0.73	link browser open page web	3.33	1.30	nng thm khi hin tnh
0.43	-1.46	multiplayer game mode online player	3.33	0.00	card deck collection pack cake

Card

NI	I	Topic	N	P	Topic
0.00	5.00	look feel fresh modern refreshed	0.00	18.75	slot machine game win bonus
0.00	5.00	character new unlock jump gender	0.00	6.25	look feel fresh modern refreshed
0.00	5.00	stage new user item upgrade	0.00	6.25	character new unlock jump gender
1.22	0.00	run smoother make faster content	0.00	6.25	new update car extreme road
1.22	0.00	update thank issue reach enjoy	0.00	6.25	event special reward limited time
1.22	0.00	release note detail github stability	0.00	6.25	save cloud game saved progress

Casino

NI	I	Topic	N	P	Topic
0.00	3.09	dragon ancient epic kingdom knight	0.00	4.84	dragon ancient epic kingdom knight
0.00	2.06	fixed crash bug disappearing freezing	-2.86	1.61	language spanish french german portuguese
-0.51	1.03	new portal plant level zombie	-2.86	0.00	display change menu direct nav
1.53	0.00	baby fun accessory make clothes	2.86	0.00	like review write help play
4.09	2.06	run smoother make faster content	2.86	0.00	building build city quest decoration
0.51	-2.06	dont forget review rate leave	2.86	0.00	circle look beauty photo eye

Casual

NI	I	Topic	N	P	Topic
-0.41	1.96	dont forget review rate leave	0.00	3.33	language franais deutsch english meeting
-0.21	1.31	language franais deutsch english meeting	-1.08	1.67	game played brain trivia organised
0.00	1.31	touch control area little pad	-1.08	1.67	para melhorias com mais nova
1.65	0.65	size reduced apk reduce smaller	4.30	1.67	free feel email contact app
1.44	0.00	better performance faster optimisation enhancement	1.08	-1.67	baby fun accessory make clothes
0.41	-1.31	sticker testing line bot chat	3.23	0.00	dont forget review rate leave

Educational

NI	I	Topic	N	P	Topic
-1.05	3.08	song music piano lyric chord	0.00	6.45	song music piano lyric chord
-0.53	1.54	mode landscape survival renamed phone	0.00	3.23	fixed bug major play bajiquan
0.00	1.54	brush color pen drawing paint	0.00	3.23	friend share facebook connect chip
0.53	0.00	let know review store like	2.94	0.00	mode landscape survival renamed phone
0.53	0.00	game played brain trivia organised	2.94	0.00	new mod added map change
0.53	0.00	new sound sleep pony bugfixes	2.94	0.00	bug fixed optimization searching waiting

Music

NI	I	Topic	N	P	Topic
-0.37	3.33	update thank issue reach enjoy	-15.00	2.50	language spanish french german portuguese
-0.74	1.67	chinese language simplified traditional japanese	0.00	5.00	adventure level island explore come
-0.37	1.67	energy far time use thank	-5.00	0.00	free version paid ad limit
0.74	-1.67	character new unlock jump gender	5.00	0.00	bug fixed minor improved hint
1.12	-1.67	christmas holiday winter gift santa	5.00	0.00	game gameplay balance like waiting
1.12	-3.33	language spanish french german portuguese	10.00	0.00	update thank issue reach enjoy

Puzzle

NI	I	Topic	N	P	Topic
-0.31	2.30	language spanish french german portuguese	0.00	2.78	language spanish french german portuguese
-0.31	1.15	chinese language simplified traditional japanese	0.00	1.39	save cloud game saved progress
-0.31	1.15	tournament player online win ranking	0.00	1.39	improvement performance stability ground extreme
0.63	0.00	leaderboard achievement leaderboards game score	0.00	1.39	car vehicle racing ford lamborghini
0.31	-1.15	challenge reward win daily earn	6.67	0.00	new update car extreme road
1.88	0.00	car vehicle wheel race steering	6.67	-1.39	car vehicle wheel race steering

Racing

NI	I	Topic	N	P	Topic
-1.11	2.52	reward arena class skill season	-4.65	0.86	coin free earn gold bonus
0.00	1.26	play google service achievement leaderboards	0.00	4.31	hero dungeon equipment skill item
0.00	1.26	added physical global turned improved	-2.33	1.72	increased level max cap limit
0.32	-0.63	help hero email need igg	2.33	-1.72	hero battle tower enemy skill
0.16	-1.26	challenge reward win daily earn	4.65	0.00	new enjoy bundle vehicle tank
1.90	0.00	guild battle raid event hero	4.65	0.00	added physical global turned improved

Role Playing

NI	I	Topic	N	P	Topic
-2.54	-1.13	graphic control environment realistic driving	-4.62	-0.89	fixed improved crash updated performance
-0.32	0.56	hero dungeon equipment skill item	-3.08	0.00	graphic control environment realistic driving
-0.32	0.56	mode immersive night preview endless	-1.54	0.89	optimization performance memory various measure
0.16	-0.56	level episode new pig pop	1.54	0.00	tt text network speech based
0.32	-0.56	scenario upgrade aplikace open knihoven	1.54	-0.89	thanks playing scene weve game
0.64	-0.56	adventure level island explore come	3.08	0.00	weapon new mission gun sniper

Simulation

NI	I	Topic	N	P	Topic
-0.09	0.84	tournament player online win ranking	0.00	1.51	live stream watch video broadcast
0.00	0.63	notification sound setting alert vibration	-0.94	0.38	version doubled displacement app randomization
0.00	0.63	news feed article section read	-0.94	0.38	language spanish french german portuguese
0.36	0.00	team match cheer vault dream	0.47	-0.75	round course shot golf hole
0.18	-0.21	fixed bug crash opening navigationdrawer	0.47	-0.75	player stats single profile starting
0.18	-0.21	notification user showing event loading	1.88	0.00	alarm time mode timer sound

Sports

NI	I	Topic	N	P	Topic
-1.30	1.96	guild battle raid event hero	0.00	2.56	building build city quest decoration
-0.87	1.96	reward arena class skill season	0.00	2.56	language spanish french german portuguese
-0.43	1.96	chinese language simplified traditional japanese	0.00	2.56	chinese language simplified traditional japanese
0.43	-1.96	christmas holiday winter gift santa	0.00	2.56	guild battle raid event hero
0.43	-1.96	thanks exciting issue visit like	8.33	0.00	fixed bug uploading crashing nowvideo
0.87	-1.96	hero dungeon equipment skill item	8.33	0.00	reward arena class skill season

Strategy

NI	I	Topic	N	P	Topic
0.00	2.44	play google service achievement leaderboards	0.00	3.70	play google service achievement leaderboards
-0.70	1.22	friend invite facebook join send	-3.57	0.00	world new tour map travel
-0.35	1.22	device crash specific friendly numerous	0.00	1.85	bug fixed minor improved hint
0.35	-1.22	weve easier making squashed feedback	3.57	0.00	update october april september november
0.35	-1.22	ad purchase remove app restore	3.57	0.00	para versin la que mejoras
1.05	-1.22	de le pour correction vous	3.57	0.00	friend invite facebook join send

Trivia

NI	I	Topic	N	P	Topic
-0.24	1.34	updated library internal party component	-1.61	2.30	friend invite facebook join send
-0.71	0.67	facebook twitter share sharing instagram	-1.61	1.15	ad purchase remove app restore
-0.71	0.67	friend invite facebook join send	0.00	2.30	word dictionary letter day offline
0.48	-0.67	play google service achievement leaderboards	1.61	0.00	app time pebble display multiple
0.48	-0.67	coin free earn gold bonus	1.61	-1.15	language spanish french german portuguese
0.71	-0.67	game play conquer additional biggest	3.23	0.00	updated library internal party component

Word

Appendix D

CIRA Usage Manual

D.1 Servlet Tool

Available at `http://www0.cs.ucl.ac.uk/staff/W.Martin/cira`. Enter the control, target, release id, confidence interval and graph fields, and the Cira server will process your request. To perform a single experiment, check graph mode and enter a release id. Targets, controls and confidence interval are required fields.

D.2 Tool Data Specifications

The format for target files is:

```
line: targetID(integer),priorStartWeek(count from 1),releaseWeek(
start of posterior),posteriorEndWeek,[comma separated list of
floats for the vector]
```

Each target goes on a new line. The format for control files is:

```
line: [comma separated list of floats - week 1 for each control]
line: [comma separated list of floats - week 2 for each control]
and so on
```

Each control spans every line; each week goes on a new line.

Appendix E

Developer Questionnaire

The following questionnaire was sent to developers who expressed an interest in receiving a report on their significant release.

Answering these questions will take only 12 keystrokes, including 6 return key presses. However, if you want to elaborate on your answer, please do feel free to do so in free text. We are very interested to hear your views on whether this research could be useful to you.
1. Did you find the time series report useful? 0 - no, 1 - yes
2. Would you be interested in further time series reports on your app(s)? 0 - no, 1 - yes
3. Do you agree that the release impacted your app's performance? 0 - no, 1 - yes
4. Are you aware of any external influences, such as advertising campaigns, that could have led to the observed effect? 0 - no, 1 - yes
5. Would you be interested in learning about potential contributing factors to significant releases? 0 - no, 1 - yes
6. Would you make any changes to your app releasing strategy based on these findings? 0 - no, 1 - yes