A Survey of App Store Analysis for Software Engineering

William Martin, Federica Sarro, Yue Jia, Yuanyuan Zhang, and Mark Harman

Abstract—App Store Analysis studies information about applications obtained 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 combines 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 software for app stores, and have led to techniques for requirements engineering, release planning, software design, security and testing. This survey describes and compares the areas of research that have been explored thus far, drawing out common aspects, trends and directions future research should take to address open problems and challenges.

Index Terms—App store, analysis, mining, API, feature, release planning, requirements engineering, reviews, security, ecosystem

1 Introduction

App stores are a recent phenomenon: Apple's App Store and Google Play were launched in 2008, and since then both have accumulated in excess of 1 million downloadable and rateable apps. Google announced that there were 1.4 billion activated Android devices in September 2015 [32]. Mobile app stores are also extremely lucrative: the set of online mobile app stores were projected to be worth a combined 25 billion USD in 2015 [152]. The success of app stores has coincided with the mass consumer adoption of smartphone devices. 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.

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).

In this paper we provide a survey of literature that performs "App Store Analysis for Software Engineering" between 2000 and November 27, 2015.[1] Our contributions are as follows: i) We provide formal definitions of apps, stores, and technical and non-technical attributes, which are used for App Store Analysis research. ii) We study the growth patterns of App Store Analysis literature both overall, and in each emergent subcategory. iii) We analyse the scale of app samples used, and discuss how this is likely to progress in the future. iv) We identify 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.

1.1 Definitions

The following definitions help to clarify key components of App Store Analysis literature. We used them to find all 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 Fig. 1, based on the data we collected in previous studies [92], [154], [202]. As our 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,

• *The authors are with the Department of Computer Science, University College London, London WC1E 6B, United Kingdom. E-mail: {w.martin, f.sarro, yue.jia, yuanyuan.zhang, mark.harman}@ucl.ac.uk.*

Manuscript received 28 Jan. 2016; revised 5 Aug. 2016; accepted 4 Oct. 2016. Date of publication 1 Dec. 2016; date of current version 25 Sept. 2017. Recommended for acceptance by W. Martin.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2016.2630689

1. This paper is an updated version of an earlier technical report [157].

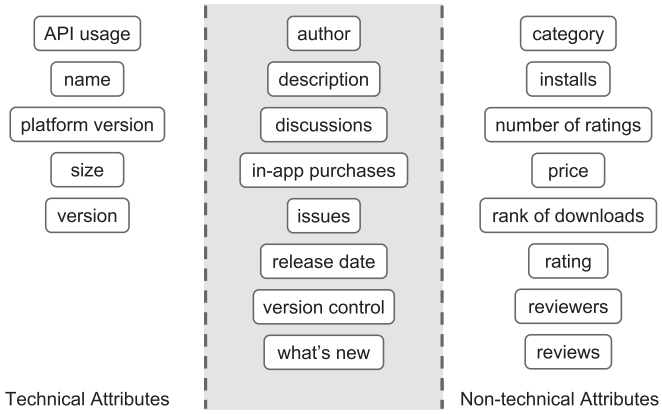


Fig. 1. *Example attributes* showing mined attributes that are strictly technical (left) or non-technical (right), and attributes that may be in either category (centre in box).

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 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. We can obtain the size of the `C` binary, and so this attribute is technical; we cannot obtain the price from either of these example files, and so this is a non-technical attribute.

Our definition of App Store may seem simplistic. However, at the time of writing, app stores serve as more than just collections of apps, but enable more developers than ever to produce and distribute content, and enable a communication channel between users and developer via reviewing systems. Therefore, our 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 has shown, 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 we can 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.

1.2 Overview

This survey is structured as follows: Section 2 describes the process used to find the included literature; Section 3 breaks down the growth trends in non-technical research compared with technical-only research, and Section 4 breaks down the growth of scale of apps used; key ideas in each subfield of app store analysis are identified in Section 5.

We define the following App Store Analysis subfields, based on the literature gathered through the process explained in Section 2: "API Analysis", which is discussed in Section 6; "Feature Analysis", which is discussed in Section 7; "Release Engineering", which is discussed in Section 8; "Review Analysis", which is discussed in Section 9; "Security", which is discussed in Section 10; "Store Ecosystem", which is discussed in Section 11; and "Size and Effort Prediction", which is discussed in Section 12.

Closely related work is discussed in Section 13; guidelines and recommendations for future app store analysis authors are outlines in Section 14; we identify potential future directions in Section 15, and conclude our findings in Section 16.

2 Literature Search

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

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, we also include studies that 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.

Our survey is not a Systematic Literature Review (SLR). The area of App Store Analysis is still developing, but has not reached a level of maturity at which research questions can be chosen and asked of a well-defined body of literature. Our study 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.

We apply the following *inclusion criteria*:

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.

We apply the following *exclusion criteria*:

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 it was not mined from an app store, and the study does not extend to app stores.

2.2 Search Methodology

In order to collect all relevant literature to date that meets the scope defined in Section 2.1, we perform a systematic search 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 criteria in three stages:

Title: We remove publications that are clearly irrelevant from the title.

Abstract: We inspect the abstract and remove publications which are clearly irrelevant according to the scope defined in Section 2.1.

TABLE 1
Search Query Results Indicating 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

Specific Queries	"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
General Queries	"app store" AND analysis AND API	"app store" AND analysis 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	1040	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

The top boxes indicate more specific queries run in multiple paper repositories, and the lower boxes indicate 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.

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 1. All of the references for papers discussed in this survey are available in an online repository [201].

2.2.1 Search Repositories

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

2.2.2 Terms

We searched for the following terms and phrases, to encompass the sub-fields of App Store Analysis that we identify: "App Store", mining, API, feature, release, requirements, reviews, security, and ecosystem. We performed searches 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

We performed 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

We mitigate the threat of missing papers 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 our definition, app store analysis research uses collections of apps, this should encompass much of the field. We also performed snowballing, which further helps to mitigate the threat of potentially missing papers. However, the threat of missing papers is a threat to the validity of any survey, including this one.

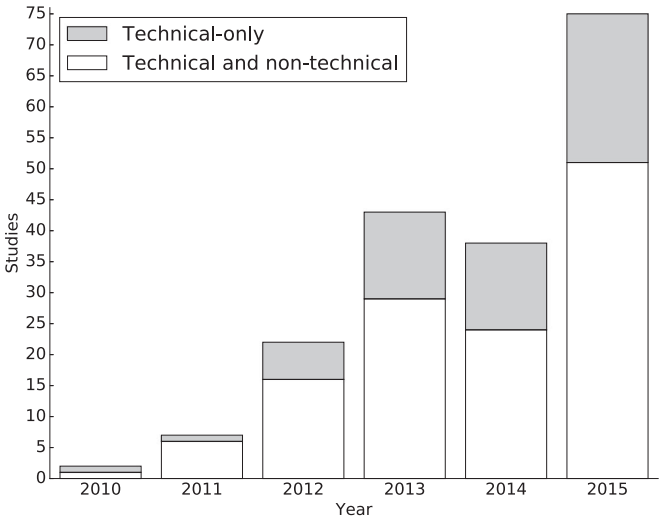


Fig. 2. *Histogram showing number of research papers incorporating non-technical information and technical-only research papers* showing the period from 2010 to November 27, 2015.

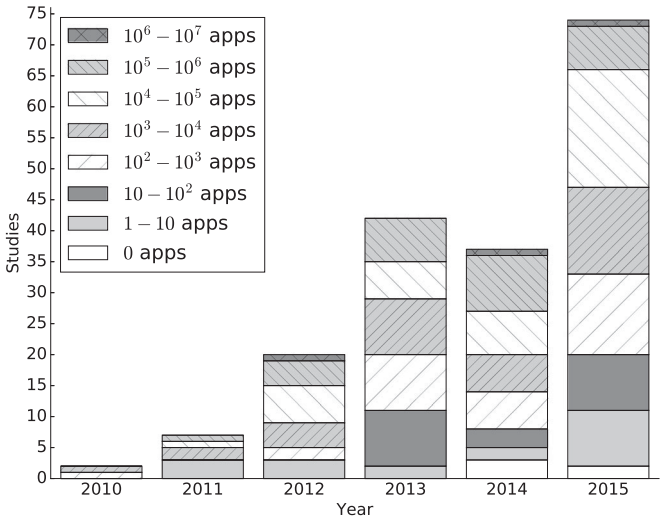


Fig. 4. *Histogram showing number of research papers grouped into app quantity ranges* each year, showing the period from 2010 to November 27, 2015. Each histogram depicts a range such as 10^2-10^3 apps, which means that the studies included used between 10^2 and 10^3 apps.

2.3 Snowballing

In addition to the repository searches specified in Section 2.2, we also perform snowballing [244] on many of the included studies. To do this we inspect the studies cited by the study, and the publications that subsequently cited the 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.

2.4 Search Results

Search results can be found in Table 1.

We set the time window to start with the year 2000, yet 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 information did not emerge until two years later. Papers were collected until November 27, 2015.

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 1. Many papers were discovered through snowballing, which do not appear in the table.

We present a summary of the included literature in Tables 3, 4, 5, 6, 7, 8, and 9. Histograms depicting the growth of publications studied on App Store Analysis for software engineering can be found in Figs. 2, 3, and 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. A breakdown of these studies in each sub-field that we identify is also presented in Fig. 5.

2.5 Lessons Learned

As can be seen from Table 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

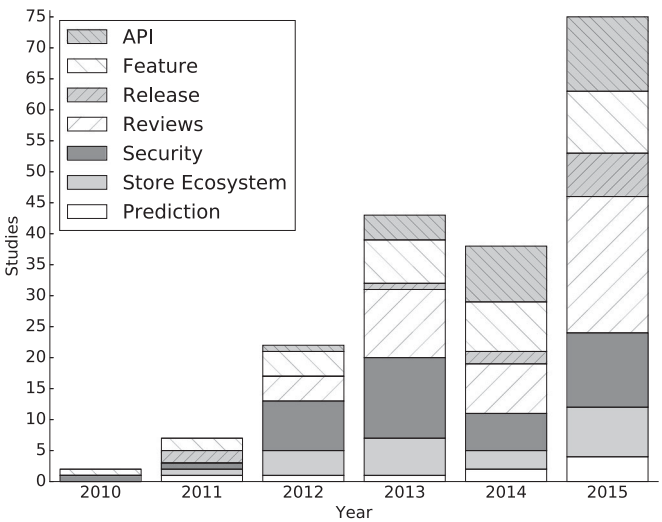


Fig. 3. *Histogram of sub-field trends* showing the period from 2010 to November 27, 2015.

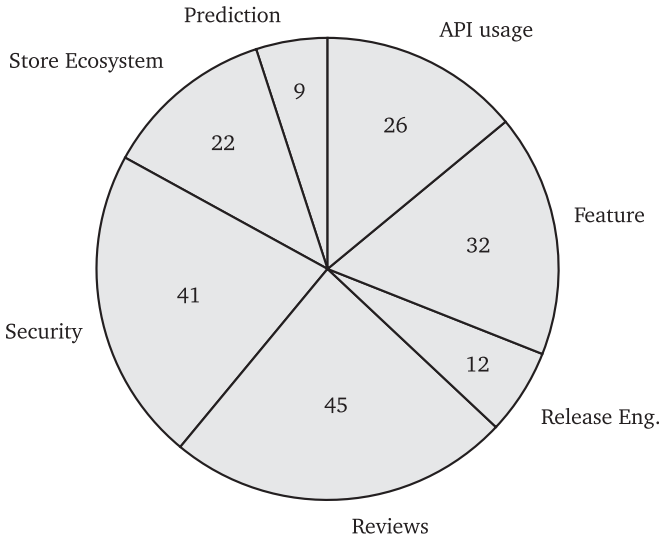


Fig. 5. *Pie chart showing overall sub-field distribution* showing the period from 2010 to November 27, 2015.

TABLE 2
Number of Research Papers Studying Each App Quantity Range from 2010 to November 27, 2015

No. Apps Range	Papers	No. Apps Range	Papers
0	5	$[10^3, 10^4)$	36
$[1, 10)$	19	$[10^4, 10^5)$	39
$[10, 10^2)$	21	$[10^5, 10^6)$	28
$[10^2, 10^3)$	31	$\geq 10^6$	3

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 our literature search, because paper discovery through many of the paper repositories we used could not be replied upon to find all relevant papers; in a growing field terms of reference are not fully stabilized. We therefore encourage future surveyors to visit the App Store Analysis paper repository [201], which can assist in the discovery of app store analysis literature.

3 Non-Technical Research

While software engineering deals primarily with code, it is not confined to deal with strictly technical sources of information. We can combine data from multiple (technical and non-technical) sources, and app stores provide a wealth of such information. There are 127 of 187 (68 percent) papers included in this study that incorporate non-technical information mined from app stores in order to either infer technical attributes (such as features), or to extract useful information such as bug reports and feature requests from users.

The histogram in Fig. 2 shows that the number of studies incorporating non-technical information is growing year-on-year. We can see from Fig. 2 that even including the boom in technical-only research, there is growth year-on-year (with the exception of 2014). Using linear regression, we are able to fit the growth trend with high accuracy ($R^2 = 0.9067$, $p = 0.003373$), which indicates that we can draw a straight line and predict (with 90 percent accuracy) the publications for a given year.

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. We assign the papers studied 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.

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 Fig. 4, where the range $[10^4, 10^5)$ is shown to grow and in 2015 represents almost half of the app usage literature.

The histogram for the studies using between 10^4 and 10^5 apps shows growth from 2011 to 2015, and this result is reflected in the histogram for 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.

5 Key Ideas Timeline

A timeline depicting the key ideas is shown in Fig. 6. This highlights the launch of major app stores studied, as well as the first studies in each subsection. We include studies into the timeline that have advanced the field of App Store Analysis in some way, or introduced influential ideas into their respective section.

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 3. 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. 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 iOS9) to developer-submitted LLVM IR (Intermediate Representation) binaries, which are then compiled for specific platforms by Apple.

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 section and Section 10. Nevertheless, the literature discussed in this Section is collected together and discussed here because it directly analyses API usage. Several papers included in this section relate to energy usage [139], [236], 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 [96].

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 1,000,000.

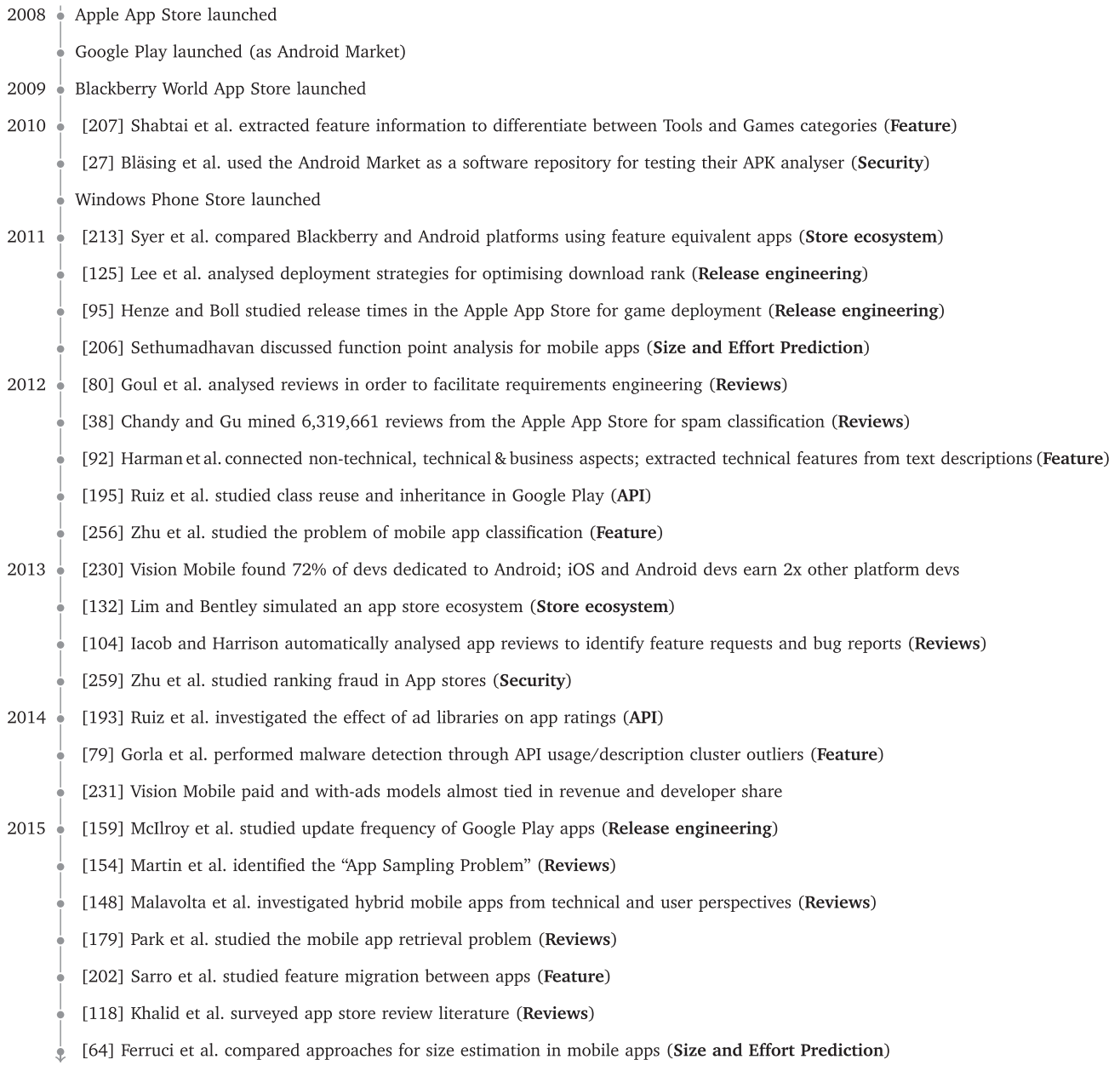


Fig. 6. *Key ideas timeline* for App Store Analysis literature. The primary area of study is suffixed in parentheses.

6.1 API Usage

Borges and Valente [30] used association rule mining to infer API usage patterns, using a dataset of 396 open source Android apps. For their study, the authors extended `API-Miner` [167] 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. [196] 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. [236] 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 [15] 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. [219] 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.

> API usage can be extracted from Android `APK` files, making analysis on the Android platform relatively straightforward. The extracted information has been used to analyse energy usage, detect malware, analyse graphical elements and to detect license violations.

TABLE 3
Chronological Summary of API-Related App Store Analysis Literature Showing the Authors, Publication Year, Publication Venue, and the Number of Apps Used in the Study

Authors [Ref], Year	Venue	No. apps
Ruiz et al. [195], 2012	ICPC	4,323
Linares-Vásquez et al. [138], 2013	FSE	7,097
Shirazi et al. [196], 2013	EICS	400
Minelli and Lanza [163], 2013	ICSM	20
Minelli and Lanza [164], 2013	CSMR	20
Ruiz et al. [193], 2014	IEEE Soft.	236,245
Hao et al. [91], 2014	MobiSys	3,600
Dering and McDaniel [57], 2014	MILCOM	450,000
Linares-Vásquez et al. [140], 2014	MSR	24,379
Ruiz et al. [192], 2014	IEEE Soft.	208,601
Linares-Vásquez [137], 2014	ICSE comp.	0
Viennot et al. [226], 2014	SIGMETRICS	1,107,476
Bartel et al. [18], 2014	IEEE Soft. Eng.	1,421
Zhang et al. [250], 2014	WiSec	10,311
Borges and Valente [30], 2015	PeerJ C. S.	396
Bavota et al. [21], 2015	IEEE Soft. Eng.	5,848
Kim et al. [121], 2015	ASE	350
Khalid et al. [114], 2015	IEEE Soft.	10,000
Watanabe et al. [242], 2015	SOUPS	200,000
Zhou et al. [254], 2015	WiSec	36,561
Wan et al. [236], 2015	ICST	398
Wang et al. [237], 2015	ISSTA	105,299
Syer et al. [214], 2015	Soft. Qual.	5
Azad [15], 2015	Masters thesis	950
Wang et al. [238], 2015	UbiComp	7,923
Seneviratne et al. [204], 2015	WiSec	4,114
	Mean	93,298
	Median	5,086

6.2 Class Reuse and Inheritance

In 2012 Ruiz et al. [195] studied class reuse and inheritance in 4,323 Android apps mined from five 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. [192] 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 [163], [164]. 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. [226] 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 percent of downloads, and the top 10 percent of most popular apps account for 96 percent of total downloads as of June 23, 2013.

Linares-Vásquez et al. [140] decompiled and analysed 24,379 APKs from Google Play and found that the 82 percent of detected clones replicate 3rd party libraries. Zhang et al. [250] 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. [121] 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. [237] 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.

> Code reuse is common in the Google Play store, but inheritance use is comparatively rare. Most apps are found to have small code bases, often replicating third party code instead of including compiled `jar` files. Clone and plagiarism detection tools are a widely discussed topic in the Class Reuse and Inheritance literature.

6.3 Faults

Linares-Vásquez et al. analysed the effect of fault and change-prone core Google APIs on app ratings [138]. 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. [138], Linares-Vásquez also presented an approach for a recommendation system for Android app developers [137], 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 [139], but this study did not combine non-technical app store information.

Bavota et al. [21] 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. [214] 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. [114] performed static analysis on 10,000 free Google Play apps, and found that three 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.

Fault and change-prone APIs have been used more frequently by poorly rated apps. Fault prone apps often depend more heavily on the platform than non fault prone apps.

6.4 Permissions and Security

In 2013 Peiravian and Xingquan [180] 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. [91] 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 [57] 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. [29], 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 [193]. 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. [79] trained a one-class support vector machine [149] on API usage information in order to identify outliers in trained clusters for security purposes. Bartel et al. [18] 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. [242] 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. [254] 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. [238] 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. [204] 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 percent of the paid apps collected personal information, compared to 85 percent in free apps. The authors subsequently showed that 20 percent of 3,605 collected Android apps were connected to more than three trackers.

There is a strong correlation between libraries used and permissions requested. Advertisement libraries sometimes have intrusive behaviours such as recording entered passwords. The treatment of personal data is a topic of interest in Permissions and Security API analysis.

6.5 Future Work

The biggest available avenue for future API analysis literature is to consider alternative platforms: all studies thus far have extracted API usage from Android apps. It remains to be seem what effect the move to intermediate representation will have on potential API analysis in the Apple App Store, but it may hinder efforts. 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.

The scale of API analysis studies is large, but future work may seek to study how usage varies over long time periods. Literature has looked at how API usage differs between apps of varying popularity or rating, but there is potential to look at differences between categories.

7 Feature Analysis

Papers that extract feature information from either technical or non-technical sources of information are discussed in this section, and are summarised in Table 4. We can observe that 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

TABLE 4
Chronological Summary of Feature-Related App Store Analysis Literature Showing the Authors, Publication Year, Store Used: g Signifies Google Play or Other Android Stores, a Signifies Apple App Store, n Signifies the Nokia (or Widsets) Platform, b Signifies Blackberry, s Signifies Samsung (Android) and w Signifies Windows Phone; Publication Venue, and the Number of Apps Used in the Study

Authors [Ref], Year	Store	Venue	No. apps
Shabtai et al. [207], 2010	g	CIS	2,285
Chen and Liu [40], 2011	a	iConference	102,337
Coulton & Bamford [49], '11	n	MobileHCI	3
Harman et al. [92], 2012	b	MSR	32,108
Sanz et al. [197], 2012	g	CCNC	820
Teufl et al. [218], 2012	g	MobiSec	130,211
Zhu et al. [256], 2012	n	CIKM	680
Mokarizadeh et al. [166],'13	g	WEBIST	21,065
Teufl et al. [217], 2013	g	Sec. & Com. Netw.	443
Lulu and Kuflik [22], 2013	g	IUI	120
Bhattacharya et al. [25],'13	g	CSMR	24
Yin et al. [249], 2013	a	WSDM	5,661
Lin et al. [135], 2013	a	SIGIR	7,116
Ihm et al. [107], 2013	g	CGC	10
Kim et al. [122], 2014	a	Service Business	100,830
Finkelstein et al. [65], 2014	b	Tech. report	42,092
Yang et al. [248], 2014	g	Tech report	26,703
Zhu et al. [257], 2014	n	TMC	680
Zhu et al. [260], 2014	g	KDD	170,753
Jiang et al. [109], 2014	g	INTERNETWARE	150
Zhu et al. [255], 2014	a	IEEE Cybernetics	15,045
Gorla et al. [79], 2014	g	ICSE	32,136
Vakulenko et al. [222], 2014	a	ICIS	600,000
Lin et al. [136], 2014	a	SIGIR	6,524
Sarro et al. [202], 2015	b,s	RE	54,983
Berardi et al. [24], 2015	a,g	SAC	5,993
Svedic [212], 2015	a	PhD thesis	60
Seneviratne et al. [205],'15	g	WWW	232,906
Tong et al. [220], 2015	g,w	JCST	10,000
Wang et al. [238], 2015	g	UbiComp	7,923
He et al. [93], 2015	g	Big Data	122,875
Tian et al. [219], 2015	g	ICSME	1,492
Nayebi and Ruhe [172],'15	g	PeerJ C.S.	241
Lulu and Kuflik [23], '15	g	MOB INF SYST	6,633
Al-Subaihin et al. [4], '16	bg	ESEM	17,877
		Mean	51,203
		Median	6,875

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.

Feature Analysis literature is broken down into "Classification", "Clustering", "Lifecycles", "Recommendation", "Success" and "Verification". This section has an overlap with Section 10, in the cases where features are used to help detect anomalies or verify app functionality.

7.1 Classification

Shabtai et al. [207] 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. [197] 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 [67].

Zhu et al. [256], [257] 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. [24] 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 [190] 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. [110] 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. [238] 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 percent and 94 percent accuracy, respectively.

> Features have been extracted for use with classifiers, in order to differentiate categories, rate app security and to predict how sensitive information is used.

7.2 Clustering

Teufl et al. [218] 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 [217] 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. [166] performed clustering on 21,065 apps, mined from Google Play, after applying topic modelling on app descriptions. They found that the resultant clusters were very different from the apps' 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 downloaded free apps more frequently, and that downloads correlated with the number of ratings an app had received.

Lulu and Kuflik [22] performed clustering on 120 apps mined from Google Play, comparing description-based with category-based clustering. They found that descriptions provided good clustering features, and presented the method as the basis of an app recommendation system. The authors later built on this work [23], 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 [74].

Kim et al. [122] 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. Al-Subaihin et al. [4] mined 17,877 apps from Google Play and Blackberry World app store. The authors clustered the apps using the similarity between features extracted from their descriptions. They scored the quality of the resulting clusters, and found them to be of higher quality than the existing categorisation of the mined apps. The authors conducted a human assessment of app similarity within clusters of varying granularity, and found a strong correlation between the similarity score of their technique and the human assessment.

Descriptions are often used for clustering apps based on their functionality. Clustering has been used to assign categories and to identify outliers. The clustering-assigned categories tend to differ from the store-assigned categories, and have also been shown to outperform them in feature classification quality.

7.3 Lifecycles

Sarro et al. [202] 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 [92], 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 migrated to a category with similar characteristics. However, there were also a few features that migrated 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 the intransitive features in a given category may support developers in identifying crucial ('must-have') requirements for their apps.

Features can migrate between apps and between categories. Intransitive features that do not migrate have been found to exhibit significantly different properties than migratory features.

7.4 Recommendation

Yin et al. [249] 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 found that the AT approach outperformed collaborative filtering and case-based reasoning in their initial experiments.

Lin et al. [135] 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 were 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. In 2014 Lin et al. [136] used topic modelling on app descriptions in order to produce a recommendation system. The model was semi-supervised and incorporated app version information using different weights corresponding to update types: so that newer app versions could be recommended when they add a certain feature to the description. Resultant topics were 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. [255] 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 stated that this can be used in a variety of ways, including app recommendation, review spam detection, and demonstrated its usefulness in ranking fraud detection. Zhu et al. [260] 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. [222] performed topic modelling on a set of 600,000 app descriptions mined from the 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. [93] 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 targeting approaches existing at the time of writing. Nayebi and Ruhe [172] 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 [2].

A variety of models have been trained on app feature data, incorporating hidden 'latent' factors, that are subsequently used to recommend apps to users, target users for advertising, and to suggest categories.

7.5 Success

In 2011, Coulton and Bamford [49] conducted a case study on games created for the WidSets platform, an earlier app store for Nokia phones (including non-smartphones). Their findings are transferable to modern app stores: high download numbers were required in order to gain active users, and popular features such as chat were able to increase the popularity and the proportion of active users. Chen and Liu [40] 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. [92] introduced app store mining as an MSR (Mining Software Repositories) problem. They mined app information and performed correlation analysis on price, downloads, and rating. Correlation 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 [65], 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. [25] 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. [107] 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. [109] conducted a user survey on 50 app descriptions in order to identify the attributes most important to the quality of a 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 were 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 [212] 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. [219] 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, which suggests that high rated apps were updated more frequently and used more modern features of the Android operating system.

> Ratings and reviews have been found to correlate with sales and download ranks. Features mined from app descriptions can be used as a basis for correlation analysis, and have been found meaningful from a developer survey.

7.6 Verification

Yang et al. [248] 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 an 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 percent category accuracy, and 76.5 percent permissions accuracy.

Watanabe et al. [242] found that apps often contain secondary functionality that is not mentioned in their descriptions. In a study of 232,906 apps, Seneviratne et al. [205] 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 establishing potential reasons for removal. Apps 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. [220] 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 (a topic model fitness function that measures the log-likelihood of generating a held-out test set), 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 [162]. Nevertheless, the study shows the importance of accurate feature discovery and representation, and can help lead to future studies using extracted features.

> Features have been used in a classifier for spam detection, and to validate whether an app makes correct claims about its functionality.

7.7 Future Work

There is potential for future work in tracking feature migration in alternative app stores: thus far the literature has studied apps mined from Blackberry and Samsung stores.

TABLE 5
Chronological Summary of Release Engineering-Related App Store Analysis Literature Showing the Authors, Publication Year, Store Used: g Signifies Google Play or Other Android Stores, a Signifies Apple App Store and w Signifies Windows Phone; the Type of Literature, and the Number of Apps Used in the Study

Authors [Ref], Year	Store	Venue	No. apps
Lee and Raghu [125], 2011	a	AMCIS	3,168
Henze and Boll [95], 2011	a	MobileHCI	24,647
Datta and Kajanan [54], 2013	a	CloudCom-Asia	3,535
Lee and Ragu [126], 2014	a	JMIS	7,579
Ruiz et al. [194], 2014	g	IEEE Soft.	120,981
Guerrouj et al. [84], 2015	g	SANER	154
Comino et al. [45], 2015	a,g	Tech report	1,000
McIlroy et al. [159], 2015	g	ESE	10,713
Gui et al. [85], 2015	g	ICSE	21
Carbunar and Potharaju [33],'15	g	ASONAM	160,000
Alharbi and Yeh [6], 2015	g	MobileHCI	24,436
Martin et al. [153], [155], 2016	g,w	ICSE comp.	1,033
Martin et al. [156], 2016	g	FSE	38,858
		Mean	29,772
		Median	5,557

Additionally, future work may seek to investigate the migration of features between different app stores or platforms.

Features have been used to classify and cluster apps, as well as recommend similar apps or categories. However, future work may apply recommendation in a different direction: it could be very useful for developers to receive recommendations on features they might implement, based on similar apps, highly desirable intransitive features, or other methods.

8 Release Engineering

This section discusses papers that focus on app releases or release strategies, which are summarised in Table 5. We can see from Table 5 that there were two papers published in 2011 that tackle this issue, one in 2013, and then a recent influx of five 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. The scale of 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.

8.1 Content

The 2014 study by Ruiz et al. investigated the updates made to update advertisement libraries [194]. They found that over 12 months, almost half of the 5,937 apps with multiple updates had an advertisement library update. Approximately 14 percent 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 percent of their releases contained ad-related changes [85].

The findings of Guerrouj et al. [84] 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 were sometimes adopted after they are depreciated, and that new pattern adoption rates were low. By tracking the app descriptions, they found that app developers sometimes updated the app descriptions to reflect changes in their applied design patterns. The authors 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. [202].

> Up to half of app updates over a 12 month period are advertisement library updates, which have been found to contain no other changes in 14 percent of cases. High code churn has been found to correlate with lower ratings.

8.2 Success

Moller et al. [170] studied the installation behaviour of users with recently updated apps, in a security related study. Lee and Raghu [126] 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 are more likely to 'survive' in the top charts, and that frequent feature updates are the most important factor in ensuring their survival, along with releasing in smaller categories. The authors also found that high volumes of positive reviews improve an app's likelihood of survival.

Carbunar and Potharaju [33] 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 percent 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.

Martin et al. [153], [155] conducted a longitudinal study on 1,033 apps mined from Google Play and Windows Phone Store over a 12 month time period. The authors used causal inference to identify the releases with most impact on ratings and downloads. They found that release text discussing features and not bug fixes may have led to more significant releases, and releases that improved rating. Martin et al. [156] later extended this work on a sample of 38,858 apps from Google Play, using their tool, `CIRA`. They found that paid apps that had significant positive effects on success were more expensive. The authors also contacted the developers of significant releases, finding that 78 percent agreed with the causal assessment and 33 percent

would consider changing their release strategy based on findings from the study.

> Positive reviews and frequent feature updates can help to keep free apps in the "most popular" charts. Causal inference has been used to identify releases with high impact on ratings and downloads.

8.3 Strategy

Lee et al. [125] published the earliest work that meets our definition of "app store analysis" in 2011 by incorporating technical with non-technical information for analysis of apps. The authors 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 sections, and found a positive relationship between download rank and app portfolio diversification. The study incorporated technical (download rank) with non-technical information (category, price) in order to identify actionable findings for app developers.

Henze and Boll [95] analysed release times and user activity in the Apple App Store, and concluded that Sunday evening was 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 [54] studied review counts from the Apple App Store, and found that apps received more reviews after deploying updates on Thursdays.

In 2014 Lin et al. [136] incorporated version information in their app recommendation system, in order to ensure that apps were recommended if they added new features to updated versions. Comino et al. [45] studied the top 1,000 apps in Apple App Store and Google Play. They found that for iTunes, increased numbers of app releases were more likely when the app was performing badly, and that releases could boost downloads. Neither finding held true for Google Play, however.

McIlroy et al. [159] studied update frequencies in the Google Play store, after mining data about 10,713 mobile apps. They found that only 1 percent of the studied apps received more than one update per week, and only 14 percent were updated in a two-week period. The authors also found that rating was not affected by update frequency. Nayebi and Ruhe [172] combined app features with values gained from crowd-sourcing as an approach to app service portfolio planning.

> App updates have been found to be more likely when an app is performing badly, and releases can boost downloads in the Apple App Store. Multiple studies suggest that day of release is a factor in the immediate success of app releases.

8.4 Future Work

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 5, the stores studied are split almost equally into Apple and Google, but there is potential future work involving release studies on Blackberry or Windows Phone Store.

TABLE 6
Chronological Summary of Reviews-Related App Store Analysis Literature Showing the Authors, Publication Year, Store Used: g Signifies Google Play or Other Android Stores, a Signifies Apple App Store, b Signifies Blackberry; the Type of Literature, and the Number of Apps Used in the Study

Authors [Ref], Year	Store	Venue	No. apps
Hoon et al. [101], 2012	a	OzCHI	17,330
Vasa et al. [224], 2012	a	OzCHI	17,330
Chandy and Gu [38], 2012	a	WebQuality	3,090
Goul et al. [80], 2012	a	HICSS	9
Ha et al. [89], 2013	g	CCNC	59
Oh et al. [174], 2013	g	CHI	24,000
Hoon et al. [100], 2013	a	Tech report	17,330
Iacob and Harrison [104], 2013	g	MSR	270
Galvis Carreño et al. [69], 2013	g	ICSE	3
Khalid [115], 2013	a	ICSE	20
Fu et al. [68], 2013	g	KDD	171,493
Chen et al. [42], 2013	a,g	WWW	5,059
Pagano and Maalej [175], 2013	a	RE	1,100
Hoon et al. [99], 2013	a	OzCHI	25
Iacob et al. [106], 2013	g	BCS-HCI	161
Iacob et al. [105], 2014	g	MobiCASE	270
Khalid [117], 2014	a	IEEE Soft.	20
Chen et al. [41], 2014	g	ICSE	4
Cen et al. [36], 2014	g	PIR	6,938
Guzman and Maalej [88], 2014	a,g	RE	7
Khalid et al. [116], 2014	g	FSE	99
Wano and Iio [241], 2014	a	NBIS	500
Erić et al. [61], 2014	a	QIP	968
Khalid et al. [118], 2015	g	IJITCS	0
Gao et al. [70], 2015	g	SOSE	4
McIlroy et al. [160], 2015	a,g	ESE	12,000
Cen et al. [35], 2015	g	SIAM	12,783
Vu et al. [234], 2015	g	ASE	3
Vu et al. [233], 2015	g	CoRR	95
Malavolta et al. [147], 2015	g	MS	11,917
Malavolta et al. [148], 2015	g	MOBILESoft	11,917
Park et al. [179], 2015	g	SIGIR	43,041
Panichella et al. [178], 2015	a,g	ICSME	7
Palomba et al. [176], 2015	g	ICSME	100
Moran et al. [168], 2015	g	FSE	14
Gomez et al. [78], 2015	g	MOBILESoft	46,644
Martin et al. [154], 2015	b	MSR	15,095
Maalej and Nabil [146], 2015	a,g	RE	1,140
Pérez [228], 2015	g	Masters thesis	4
Khalid et al. [119], 2015	-	IJIEEB	0
Gu and Kim [83], 2015	g	ASE	17
Guzman et al. [86], 2015	a,g	ESEM	7
Guzman et al. [87], 2015	a,g	ASE	7
McIlroy et al. [161], 2015	g	IEEE Soft.	10,713
Liang et al. [128], 2015	a	IJEC	139
		Mean	9,594
		Median	161

Several studied in this section have looked at developer strategy, and release behaviours that associate with positive results. A potential for future work could be individualised recommendation of strategy, which could be particularly useful for app developers who wish to break into higher app store ranks.

9 Review Analysis

Literature discussed in this section concerns the study of app reviews; a summary of discussed literature can be

found in Table 6. We can see from Table 6 that the majority of studies focused on the Google Play store, with a minority focusing on Apple App Store, and 1 paper studying Blackberry store. Review-centred literature was first published in 2012, and subsequently has gained significant and increasing interest and activity: we can see from Fig. 3 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.

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

Literature in this section 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.

9.1 Classification

Chandy and Gu [38] 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. [42] 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 percent of the Android apps were underrated and 18.1 percent 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 and user reviews, and iOS maturity ratings, to automatically verify app maturity ratings. Ha et al. [89] 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 concerned the quality of the app, and not security or privacy concerns.

Cen et al. [36] 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-sourced approach [35]. Using `AndroGuard` [8] 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. [78] 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. [87] developed an ensemble of machine learning classifiers in order to classify user reviews. They tested this system on 4,550 reviews mined from seven 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.

Reviews have been classified for spam, maturity ratings, and privacy and security risks. Research in 2015 has also used reviews to help detect erroneous apps.

9.2 Content

Hoon et al. [101] and Vasa et al. [224] 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 [100]. They found that the majority of mobile app reviews were short in length, and that rating and category both influenced the length of reviews. The majority of studied apps received under 50 reviews in their first year. Half of the apps analysed decreased in the user assessment of quality, denoted by rating over time. The authors suggested that user expectations were changing rapidly towards apps, and that developers must keep up with demand to remain competitive.

Iacob et al. [106] 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 nine 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. [116] studied the devices used to submit app reviews, in order to determine the optimal devices for testing. Palomba et al. [176] studied the Google Play reviews from 100 open source Android apps, and linked the reviews to code changes. They found that a mean of 49 percent of review requests were implemented in new releases, and that the apps with changes more directly implementing the content of user reviews improved their ratings with new releases. In order to bridge the gap between software attributes and user reviews, Hoon et al. [98] developed an ontology of words used to describe software quality attributes in app reviews.

McIlroy et al. [161] 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 percent of users were found to subsequently change their ratings, resulting in a median increase in individual user ratings of 20 percent. A summary of mobile app user feedback classification can be found in the study by Maalej et al. [235].

Review content literature has investigated the vocabulary and ontology of reviews, the factors affecting feedback, and devices most used for review submission.

9.3 Requirements Engineering

Oh et al. [174] developed a review digest system, that 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 [104] 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. One sixty-one apps and 3,279 reviews were used for manually training linguistic rules. 136,998 reviews were used for the evaluation, which found that 23.3 percent of reviews contained feature requests.

As an extension to the MARA system they had previously introduced [106], Iacob et al. [105] introduced a set of linguistic rules for identifying feature requests and bug reports in order to help facilitate app development. Wano and Iio [241] 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. [61] studied the star ratings of 48 million reviews mined from 968 popular free and paid Apple apps. They found that the reviews were mostly positive, and that there were significant differences in the distributions between categories, and also between free and paid sections. Free apps had more reviews but a lower mean rating, and higher standard deviation. Due to the higher numbers of reviews for free apps, which might give an app credibility, the authors argued that in-app purchasing revenue models were a good way to make money for developers, especially if used as a 'teaser' for a paid version.

Park et al. [179] 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. [178] 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 a precision of 0.85 and recall of 0.85, when training the system on language structure, content and sentiment features. Maalej and Nabil [146] produced a classification method identifying bug reports and feature requests from user reviews. The authors found that upwards of 70 percent precision and 80 percent recall could 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, could negatively affect the performance of this classification task.

Moran et al. [168] proposed 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. [119] argued that app store reviews can be used for "crowdsourcing" [150]. They argued 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.

> The requirements engineering review literature has used reviews to extract bug reports and feature requests, in addition to prioritising critical reviews.

9.4 Sentiment

The works discussed in this section have incorporated sentiment in their study of reviews. Sentiment describes a user's views or opinions, typically as positive or negative in this content, 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. [80] 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 [69] extracted user requirements from comments using the ASUM model [111], a sentiment-aware topic model. Initial results showed that the method aided requirements summarisation with significantly less effort than manual identification, but did not return all possible requirements. Hoon et al. [99] gathered a set of 29,182 short reviews of up to five words from the top 25 Health & Fitness apps in the Apple App Store. They analysed the reviews and found that they are mostly made up of sentiment words, which matched the star rating of the review closely.

Khalid [115], [117] 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 over 250,000 reviews combined, and therefore 6,390 reviews is a statistically representative sample at the 95 percent confidence level. The authors carried out a manual analysis of the 6,390 reviews, finding that 11 percent 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 [175] 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 was provided after releases, that positive feedback was often associated with highly downloaded apps, and that negative feedback was often associated with less downloaded apps and often did not contain user experience or contextual information.

In 2014 Chen et al. [41] produced a system for extracting the most informative reviews, placing weight on negative sentiment reviews. Guzman and Maalej [88] studied user sentiments towards app features from a multi-store sample, and studied the differences between user sentiments in Google Play from the Apple App Store. Guzman et al. [86] 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 [88]. Liang et al. [128] performed MultiFacet Sentiment Analysis

(MFSA) on user reviews mined from 139 apps on the Apple App Store. They reported that opinions on product quality formed a large portion of reviews, but opinions on service quality had a bigger effect on sales.

Sentiment analysis has identified frequent causes of user complaints, and has helped to prioritise informative reviews.

9.5 Summarisation

A large sample was used in the 2013 study by Fu et al. [68], 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 so it must 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 had an associated 'complaint' type about price, that free apps did not.

In 2015 McIlroy et al. [160] 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. [70] proposed `AR-Tracker`, a similar tool to `AR-Miner` [41], 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 [228] 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 an 0.78 accuracy.

Malavolta et al. [147], [148] analysed 3 million reviews from 11,917 Google Play apps, and produced a summary of user perceptions about 445 hybrid apps [94] compared with native apps. The authors found that hybrid mobile apps received similar ratings to native apps, but native apps had been reviewed on average 6.5 times more. They planned (at the time of writing) 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. [233], [234] developed `MARK`, a system that identifies keywords in sets of reviews in order to assist with summarisation and search. The method is one of several summarisation approaches that are applied to reviews.

Gu and Kim [83] proposed `SUR-Miner`, a review summarisation and categorisation tool, that they evaluated on 2,000 sentences mined from reviews of 17 Google Play apps. The tool was 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.

Review summarisation helps developers to gather information from large numbers of reviews that would be infeasible to read individually. A number of tools have been produced, such as `WisCom`, `AR-Tracker`, `MARK` and `SUR-Miner`.

9.6 Surveys and Methodological Aspects of App Store Analysis

Martin et al. [154] identified the App Sampling Problem, finding that the majority of past work used partial subsets of biased data for app review analysis. The authors assessed the bias and identified techniques that can be used to ameliorate its effects, as well as defining a classification scheme that can be applied to app review analysis studies to describe dataset completeness. Khalid et al. [118] 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.

There have been two recent methodological analytical surveys of the review analysis literature, including suggestions for improving the reviewing process, and the highlighting of a prevalent methodological issue called the "App Sampling Problem".

9.7 Future Work

Many studies have produced tools which can aid in the summarisation and requirements extraction from reviews, but these tools have not been widely adopted as of yet by developers. Future work may seek to bridge this gap, by making tools available to developers in some form.

An 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. In particular, a comparison of the review taxonomy, system and culture between different platforms including Windows Phone store is a potential future work.

10 Security

Studies relating to app security are discussed in this section, and are summarised in Table 7. We can see from Table 7 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 use properties such as the validation that highly rated apps have received, through being downloaded, used, and highly rated by many users. Much of the security-related literature uses the property that popular apps can generally be assumed to be non-malware, since they are scanned prior to being hosted in the store, and have large user bases.

There are many studies on mobile app security that use app stores in a less direct way than those discussed in this section, some of which are mentioned in Section 13.2.

Additionally, literature in Section 6.4 has identified potential risks associated with permissions, beyond the more direct security threats discussed in this section.

Many studies in this section use 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).

Security literature is broken down into "Faults", "Malware", "Permissions", "Plagiarism", "Privacy" and "Vulnerability" sections.

10.1 Faults

Ravindranath et al. [188] 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. [143] presented their DECAF system for detecting advertisement placement and layout violations, that can be used to 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. [52] 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 percent of apps made advertisement requests while running in the background, and 27 apps simulated user clicks on advertisements. Deng et al. [56] introduced their iRiS system, that 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.

Faults in mobile applications can point to potential security risks. Work in fault analysis has detected layout rule violations, terms of service violations and advertisement library issues.

10.2 Malware

In 2010 Bläsing et al. [27] used the top 150 free Google Play apps to test their static and dynamic APK analyser. They tested these apps against one known malware app, which was shown to be an outlier, establishing that their approach has the potential for Malware detection. Peng et al. [181] 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. [262] 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 had 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.

TABLE 7
Chronological Summary of Security-Related App Store Analysis Literature Showing the Authors, Publication Year, Store Used: g Signifies Google Play or Other Android Stores, or the Android Platform in General, and a Signifies Apple App Store; the Type of Literature, and the Number of Apps Used in the Study

Authors [Ref], Year	Store	Venue	No. apps
Blasing et al. [27], 2010	g	MALWARE	150
Batyuk et al. [20], 2011	g	MALWARE	1,865
Potharaju et al. [183], 2012	g	ESSoS	158,000
Moller et al. [170], 2012	g	LARGE	1
Chia et al. [43], 2012	g	WWW	19,344
Gibler et al. [72], 2012	g	TRUST	24,350
Grace et al. [81], 2012	g	WiSec	100,000
Crussell et al. [50], 2012	g	ESORICS	75,000
Peng et al. [181], 2012	g	CCS	500,000
Zhu et al. [262], 2015	g	ICICS	5,685
Awang Abu Bakar and Mahmud [14], 2013	g	ACSAT	5,000
Stevens et al. [211], 2013	g	MSR	10,300
Book et al. [29], 2013	g	CoRR	114,000
Sanz et al. [198], 2013	g	Cybernet. Syst.	333
Sanz et al. [200], 2013	g	SECRYPT	333
Sanz et al. [199], 2013	g	NSS	333
Wang et al. [240], 2013	g	DBSec	272,774
Crussell et al. [51], 2013	g	ESORICS	265,359
Gibler et al. [73], 2013	g	MobiSys	265,359
Peiravian and Xingquan [180],'13	g	ICTAI	1,250
Chakradeo et al. [37], 2013	g	WiSec	14,888
Pandita et al. [177], 2013	g	SEC	581
Zhu et al. [259], 2013	a	CIKM	15,045
Liu et al. [143], 2014	w	NSDI	51,150
Crussell et al. [52], 2014	g	MobiSys	165,426
Gorla et al. [79], 2014	g	ICSE	32,136
Zhang et al. [250], 2014	g	WiSec	10,311
Dering and McDaniel [57], 2014	g	MILCOM	450,000
Ham and Lee [90], 2014	g	IJCCE	10
Bhoraskar et al. [26], 2014	g	SEC	1,010
Qu et al. [186], 2014	g	CCS	45,811
Zhu et al. [261], 2015	a	TKDE	15,045
Kim et al. [121], 2015	g	ASE	350
Wang et al. [237], 2015	g	ISSTA	105,299
Schütte et al. [203], 2015	g	ConDroid	10,000
Mutchler et al. [169], 2015	g	MoST	998,286
Avdiienko et al. [13], 2015	g	ICSE	2,866
Ma et al. [145], 2015	g	COMPSAC	22,555
Vigneri et al. [227], 2015	g	CoRR	5,000
Yang et al. [247], 2015	g	ICSE	633
Lageman et al. [124], 2015	g	MILCOM	417
Deng et al. [56], 2015	a	CCS	2,019
Zhang et al. [251], 2015	g	CCS	100
Huang et al. [102], 2015	g	SEC	16,000
Chen et al. [39], 2015	g	SEC	1,165,847
		Mean	106,929
		Median	14,888

Chakradeo et al. [37] 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 [180] trained a malware classifier using 1,250 samples of known malware, and 1,250 samples of (assumed 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. [200] 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 [198], [199]. Similarly, Wang et al. [240] 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 are 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. [79] 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) [149] on each cluster to detect outliers in terms of API usage. This method indicated apps that exhibit sufficiently different behaviour from the norm to suggest the presence of malware. This approach was later extended by Ma et al. [145] who used known malware and benignware to train their model, and reported improvements on the resultant precision, recall and F-measure.

Avdiienko et al. [13] 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 were to be assumed benign as their download ranks were 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 [253] and VirusShare database [229]. In a similar way, the 2013 study by Sanz et al. [198] 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 [90], also used apps from the Google Play Games category as a benign set, against which to compare.

Lageman et al. [124] 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 percent with a Random Forest classifier [76]. In the largest app study to date, Chen et al. [39] 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.

There has been much work on malware detection through the use of static and dynamic analysis, and also alternative sources of information such as descriptions, API usage and data flow.

10.3 Permissions

In 2013 Awang Abu Bakar and Mahmud [14] mined 5,000 apps from the Google Play store and analysed their permissions. They found significant correlations of rho 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 highlighted the top permissions requested by apps, and found that 40 percent of the apps requested the phone's status and identity, a source of sensitive information. Stevens et al. [211] mined 10,300 apps from several Android stores including Google Play, and applied the permissions analysis tool Stowaway [12], that can detect declared and used permissions. The authors found that 44 percent of apps in their sample contained at least one unnecessary permission, and computed a Spearman's correlation coefficient of 0.72 between the 'popularity' of permissions on Stackoverflow and their misuse. Book et al. [29] studied library permissions on 114,000 apps mined from the Google Play store, and showed that libraries bundled with apps lead to old versions being included. Increasingly, advertisement libraries were taking advantage of app permissions, presenting a potential security risk that the authors argue should be solved by the app store or privacy legislation.

Pandita et al. [177] 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, that were manually labelled by the authors. The authors tested the system on the permissions for address book, calendar and audio recording, and achieved an average precision of 82.2 percent and recall of 81.5 percent. In a related study, Qu et al. [186] 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 percent and a recall of 92.0 percent when detecting 11 permissions.

The findings by Dering and McDaniel [57] suggest that library usage presents a security risk due to permissions usage. This is discussed in more detail in Section 6.

Library usage can present a security risk by taking advantage of requested app permissions. Tools such as Stowaway, WHYPER and AutoCog check permission usage, comparing it against permissions requested.

10.4 Plagiarism

In 2012 Potharaju et al. [183] 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 percent of apps with the most permissive rights were the most likely to spread malware, and that non-technical information such as category, number of downloads and publishing day could increase the initial spread of the malware. Crussell et al. [50] 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, that decompiles apps and compares them to detect clones [51]. They detected 4,295 cloned apps using this approach from a mined set of 265,239 apps. This dataset

was used in the study by Gibler et al. [73], who investigated the effects of application plagiarism on developers.

Zhu et al. [259], [261] 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 [121], [237], [250], whose studies are discussed in more detail in Section 6.

App plagiarism is a common approach to spread malware; a number of authors have detected cloned apps in the Google Play store. Potentially fraudulent apps have also been detected in the Apple App Store.

10.5 Privacy

In 2011 Batyuk et al. [20] used the top 1,865 free Google Play apps to test their static APK analyser, which detected that 167 apps accessed private identifiers, thereby presenting a security risk. Of these apps, 114 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` [53]. Chia et al. [43] 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 had received, but no correlations between permissions sought and privacy risk, nor rating. This result shows that ratings were not an effective indicator of the privacy of apps, and new suspicious apps were not likely to receive many ratings which could have served as a warning for future users.

Gibler et al. [72] 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. [81] 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 used advertisement libraries, of which 31 percent used more than one. The authors remarked that the majority of the 100 studied advertisement libraries were found to collect personal information.

Vigneri et al. [227] 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 that 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 determine apps with suspiciously high activity. The authors noted that a large proportion of apps, even those with high ratings and download ranks, downloaded a large number of advertisements. Huang et al. [102] presented their `SUPOR` system, which 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, and obtained a precision of 0.973 and a recall of 0.973, with a false positive rate of 0.087. The cases found included fields for national ID, username, password, credit card and health data.

Ratings were not found to correlate with permissions or privacy risk, but suspicious apps did not receive many ratings. Many advertisement libraries have been found to collect personal information, presenting a potent privacy risk.

10.6 Vulnerability

Moller et al. [170] 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 argued that this could have lead to users continuing to run vulnerable software even after a fix was available.

In 2015, Zhang et al. [251] 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, and asked whether the generated descriptions were readable and whether they could reduce the rate at which users download malware. The generated descriptions achieved a 4 percent lower readability rating than the original human-written descriptions, but decreased the malware download rate by 39 percent. Yang et al. [247] 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 was more related to whether an app was malicious than the type of security-sensitive resources that it accessed.

Schütte et al. [203] 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. [169] took a snapshot of 1,172,610 Google Play apps. They found that 998,286 of these apps used the `WebView` API, indicating that the apps used an embedded `WebView` in some way. The authors searched for several known vulnerabilities and found that 28 percent of the studied apps had at least one of these vulnerabilities. As a result, the authors proposed a set of API changes to mitigate such threats. In a related study, Bhoraskar et al. [26] 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 200 apps using the Facebook SDK were vulnerable to known attacks, and that 175 of 220 children's apps potentially collected information in violation of the US Children's Online Privacy Protection Act [46].

Reading app descriptions before downloading them can help to protect against vulnerability, since machine-generated descriptions are less readable, thereby service as potential warnings. Several authors have identified prevalent vulnerabilities, such as logic bombs and embedded `WebViews`.

TABLE 8
Chronological Summary of Store Ecosystem-Related App Store Analysis Literature Showing the Authors, Publication Year, Store Used: g Signifies Google Play or Other Android Stores, a Signifies Apple App Store, b Signifies the Blackberry Store and w Signifies Windows Phone; Publication Venue, and the Number of Apps Used in the Study

Authors [Ref], Year	Store	Venue	No. apps
Syer et al. [213], 2011	b,g	SCAM	3
d'Heureuse et al. [58], 2012	a,b,g,w	MCCR	1,164,489
Jung et al. [113], 2012	a	Market Lett	1,189
Lim and Bentley [131], 2012	a	GECCO	*
Lim and Bentley [130], 2012	a	ALIFE	*
Lim and Bentley [132], 2013	a	CEC	*
Garg and Telang [71], 2013	a	MIS	1,223
Ihm et al. [107], 2013	g	CGC	10
Zhong & Michahelles [252],'13	g	SAC	191,301
Petsas et al. [182], 2013	g	IMC	316,143
Syer et al. [215], 2013	g	CASCON	15
McDonnell et al. [158], 2013	g	ICSM	10
Cocco et al. [44], 2014	a	MWIS	***
Wenxuan and Airu [243],'14	a,g,w	ICDMW	736,377
Ng et al. [173], 2014	g	COMPSAC	506
Liu et al. [141], 2015	g	WSDM	6,157
Ruiz et al. [165], 2015	g	IEE Soft.	10,150
Syer et al. [214], 2015	g	Soft. Qual.	5
Joorabchi et al. [112], 2015	a,g	ISSRE	14
Gómez et al. [77], 2015	g	ICSE NIER	1
Askalidis [11], 2015	a	CoRR	162
Xie and Zhu [245], 2015	a	WiSec	179,353
Corral and Fronza [47], 2015	g	MOBILESoft	100
Lim et al. [133], 2015	a	TEVC	**
		Mean	144,845
		Median	848

Numbers in the table indicate empirical app data mined from stores, (*) signifies 500,000 simulated apps, (**) signifies 1,250,000 simulated apps, and (***) signifies over 500 simulated apps (final values were not specified by the paper's authors); only empirical data is considered for mean and median.

10.7 Future Work

There is potential future work in App Store Security Analysis in augmenting approaches with the non-technical information made available by app stores. Additionally, there is a great deal of literature on app security, perhaps warranting a standalone survey that can bring together elements that do not meet the scope for app store analysis.

The majority of studies in the section use datasets from Google Play, three study Apple datasets and just one studies Windows. Potential future work may therefore seek to study security on collections mined from Apple, Windows and Blackberry, and may extend to cross-store security analysis or comparison.

11 Store Ecosystem

In this section we discuss literature that focuses on a store's ecosystem, or the differences between stores. This literature is summarised in Table 8.

> The scale of store ecosystem studies ranges from 1 to 1,164,489 apps, and simulations have included up to 1,250,000 apps.

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

11.1 Inter-Store

In 2011, Syer et al. [213] 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 the Android apps were generally smaller but rely heavily on the platform. Conversely, Blackberry apps were larger and relied 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. [215] later compared development practices between 15 Android apps and five traditional desktop and server applications. They found that mobile apps were most similar to `Unix` utilities, in terms of smaller code bases and small development teams. However, they also reported that mobile apps suffered from greater numbers of defects and slower fix times than the studied traditional applications.

In 2012 d'Heureuse et al. [58] mined 1,164,489 apps in total from Apple, Blackberry, Google and Windows app stores. The apps were mined 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 percent. 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 three other stores.

Petsas et al. [182] analysed the downloads of 316,143 apps from 4 third-party Android app stores. They found that 10 percent of the apps accounted for at least 70 percent of the total downloads in the stores, and that user downloads followed a clustering type behaviour, where their 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. [173] 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 were the same as the official releases, and ranked the severity level of differences. The authors concluded that the third party app stores studied could not be trusted, as the proportion of apps which did not match their official releases was high, as were the corresponding difference severity levels.

In 2015 Ruiz et al. [165] 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 the 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. [112] introduced CheckCAMP, a tool that checks for inconsistencies between Android and iOS versions of the same app. The authors tested the tool on seven open source apps and seven 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.

Inter-store analysis literature has compared code practices, growth, user download behaviour and consistency between app stores such as Google, Blackberry, Apple and Windows. Third party Android app stores have been found to host apps that *appear* identical yet contain significant differences to the official versions, a sign that they may contain malware.

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. [214] uses information on the number of downloads from AppBrain, a replication of the number of installs bracket on Google Play (e.g., 1,000,000-5,000,000 installs appears on AppBrain as 1,000,000+). Ihm et al. [107] combined download information on 10 social networking apps from Google Play with the number of registered users on their respective websites. They found a strong correlation between the two metrics.

11.2 Intra-Store

Jung et al. [113] 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 is also benefit to getting an early entrant in markets. In 2013 McDonnell et al. [158] studied 10 apps using source code from github [75]. The Android platform was shown to be evolving, with an average of 115 API updates per month, due to which 28 percent 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 (at the time of writing) have accessible information on their number of downloads, other than 'ranges', such as the range 50-100. Zhong and Michahelles [252] analysed the distributions of download buckets and ratings of 191,301 apps from Google Play. They found that a small number of popular 'blockbuster' apps accounted for the majority of app downloads, and also had high ratings indicating customer satisfaction. Paid apps achieved more success when they were cheaper, but expensive professional apps had disproportionally high numbers of downloads. The authors concluded that developers could break into the higher download ranking positions by fulfilling a niche market. Garg and Telang [71] compared paid app demand in the Apple App Store, using download ranks. They found that the top ranked paid app is downloaded 120-150 times more than the 200th ranked app.

Askalidis [11] 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 was cheaper than their competition. They authors also found that sales where apps became free, or had easily redeemable digital discounts, were the most successful. Sales were shown to have mixed effects on the ratings of apps. Gómez et al. [77] 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 [245] investigated the practice of promoting apps through buying positive reviews, via illicit "underground" services. The authors registered on eight 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 [47] 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 hypotheses that source code quality plays a role in app success.

Source code quality has been shown to have no strong correlation to app success for open source apps. However, factors such as price, sales promotions and catering to a niche market may all play a factor in app success.

11.3 Recommendation

In 2014 Wenxuan and Airu [243] used information on the number of downloads and numbers of reviews, as well as the numbers 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. [141] 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 used the rating distribution over their dataset as the motivation for modelling user preference with a Poisson distribution.

The Interoperability-Enriched Recommendation (IER) system uses cross-store data to make app recommendations. Privacy allowances have also been used to enhance app recommendation for Google Play.

11.4 Simulations

Lim and Bentley simulated the app store ecosystem using an agent-based evolutionary model, in order to experiment with different publicity strategies [130], [131], modelling apps with infectious properties, so that they can spread after being downloaded by a user. They found that an 'app epidemic' is most likely to occur when the app appears on the

TABLE 9
Chronological Summary of App Store Analysis Literature Related to Size and Effort Prediction Showing the Authors, Publication Year, Publication Venue, and the Number of Apps Used in the Study

Authors [Ref], Year	Venue	No. apps
Sethumadhavan [206],'11	ISMA	6
Preuss [185], 2012	The IFPUG Guide to IT & Software Measurement	1
Preuss [184], 2013	ICEAA	1
van Heeringen and van Gorp [223], 2014	IWSM-MENSURA	0
Abdullah et al. [1],'14	ICOS	0
D'Avanzo et al. [55],'15	SAC	8
Francese et al. [66],'15	SEAA	23
Ferrucci et al. [63],'15	SEAA	13
Ferrucci et al. [64], 2015	PROFES	13
	Mean	7
	Median	6

'new apps' chart. The authors then used the model to explore different ranking algorithms [132]. The study 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 performed best in terms of overall downloads by incorporating fresh apps, and for this to work it needed to incorporate less historical performance data (also found later by Ruiz et al. [165]).

Lim et al. later simulated the ecosystem from a user's perspective [133], using collected usage information from over 10,000 participants [129]. They modelled developer strategies such as 'innovator' (who produces apps with random features) and 'copycat' (who copies the app) [133]. They found that 'optimiser' (who improves on the original 'innovator' apps) and 'copycat' working together led to the best overall fitness, provided that they represented a low proportion of the overall modelled developer population.

An agent-based evolutionary model has been used by multiple authors to simulate an app store. Authors have studied such factors as ranking algorithms and the optimal interaction between simulated developers.

Cocco et al. [44] 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 percent chance of a new app appearing in the top charts leads to the highest downloads-to-browse ratio.

11.5 Future Work

There are potential research opportunities to be found comparing stores, especially comparing the Windows Phone Store against more well-studied stores such as Apple, Blackberry and Google. Future studies may continue to build on the store simulation work by Lim et al., and may extend analysis to the success or evolution of less widely used stores such as Windows or Blackberry.

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 9. 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 [206] 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 [184], [185] 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 [223] discussed how to use COSMIC [48], a second generation functional size measurement method, in order to measure the functional size of mobile 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. [55] 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 could accurately estimate code size based on functionality alone, once trained. Francese et al. [66] 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. [63] applied the COSMIC approach to 13 Android applications, and showed that functional size was strongly correlated with app size, and that it could be used to accurately estimate the bytecode size of an app. Ferrucci et al. [64] later compared the related approaches by D'Avanzo et al. [55] and van Heeringen and van Gorp [223] on their dataset of 13 Android apps. They found that functional size results were correlated with multiple app size measures, but that the approach presented by D'Avanzo et al. [55] was more accurate.

Function Point Analysis and the COSMIC method of measuring the functional size of apps, have been used to predict the size and effort of apps.

12.1 Future Work

Size and effort prediction is a relatively small section of app store analysis, that we expect to continue to grow. Future studies may extend to Apple, Windows and Blackberry stores, and may seek to incorporate mined feature or API information to increase their prediction accuracy. We also expect predictive modelling to be used for estimating other properties of mobile apps, such as faults and crashes.

13 Closely Related Work

The following literature is important to the field of App Store Analysis, yet does not quite fully meet our exact definition of App Store Analysis. Nevertheless, we recognise that it would be restrictive to apply our definition too rigorously at this early stage in App Store Analysis. Furthermore, since this work meets aspects of our 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 thus far in the survey.

13.1 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.

In 2011 Böhmer et al. [28] 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 exceptions to this rule were communication apps, which were used throughout the day. In 2012, Ferreira et al. [62] surveyed 4,035 Android user charging habits, using an app to record their behaviour. Lin et al. [134] conducted a survey on 179 Android users, that asked about their expectations of app purpose and sensitive data handling. 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.

Shi et al. [209] developed a mobile app recommendation system that works by learning user preferences. Similarly, Zhu et al. [258] mined context-aware user preferences using log information. Rein and Münch [189] 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. [174] surveyed 100 app users and found that users were more likely to take a passive approach and delete apps rather than reviewing or contacting developers, but when they took an active approach, reviewing was the most popular approach. In 2014 Tan et al. [216] 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. [129] surveyed app users from 15 countries to understand how usage of apps and app stores differed by region. They found that behaviour did differ 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 was higher than average in Brazil and the UK, and lower than average in Japan and France, indicating that differences were influenced by more than global region.

13.2 Related Security

We 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 10.

Enck et al. [60] introduced `Kirin`, an Android app certification tool for flagging potential malware using a set of rules. In 2010 Enck et al. introduced `TaintDroid` [59], 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. [10] in 2014, called `FlowDroid`. This tool statically analyses information flow to find all possible flows. Mao et al. [151] introduced `Sapienz`, an Android testing tool that the authors applied to the top 1,000 Google Play apps, revealing 558 previously undetected crashing test causes.

Some authors have used sets mined from Google Play as benign app sets to test against known malware: Xu et al. [246], Rastogi et al. [187], Jing et al. [110], Arp et al. [9], Wang et al. [239], Liu and Liu [144], Roy et al. [191] and Khanmohammadi et al. [120]. Ho et al. [97] 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. [17], Jeon et al. [108], Grace et al. [82], Crussell et al. [50], [51], Ravindranath et al. [188], von Rhein et al. [232], Li et al. [127], Huang et al. [103], Cen et al. [34], Liu et al. [142] and Bastani et al. [19].

13.3 Reports

Initial studies, such as the 2010 work by Sharma et al. [208], evaluated the size and growth of the apps market up to the time. In 2011 Butler [31] 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 [210] published a report on the Apple App Store Education category, comparing it with their previous study in 2009. They found that over 72 percent of the top-selling apps in this category targeted children under 11 years of age, a number that had significantly increased from 47 percent 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 [230] on app industry monetary value and growth found that 72 percent of developers are 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.

13.4 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. [16] published `OSSGrab`, which mines `HTML` pages from Google Play. The tool was built in order to facilitate their app permissions study [14]. In 2014, Viennot et al. introduced the `PlayDrone` Google Play crawler [225], to facilitate their large scale API study [226].

The Android Malware Genome Project [253] is a popular source of malware applications for testing security tools. In 2015 Krutz et al. [123] made available a dataset containing 1,179 open source applications. The `AndroZoo` project provides a large collection of Android `APKs` [221].

14 Guidelines for Future App Store Analysis Authors

In this survey we have reported on the general content of studies, as well as the scale of apps used, and the store used. Our previous overview of review analysis [154] synthesised the number of reviews used, and the type of reviews dataset. 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. We therefore present the following guidelines for data to include, as suggestions for future App Store Analysis work, 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.
Breakdown of sampling dataset used [154] where applicable, particularly when reviews are used.
Description of ratings and user feedback categories, including trends and response ranges.
Details of statical analysis techniques that were used in the study.

15 Future Work

Here we discuss potential future avenues of research for app store analysis. Other such discussions can be found in the works by Al-Subaihin et al. [3], and Nagappan and Shihab [171].

Expectations. We expect to see the scale of app samples used increase in the years to come, as app stores increase in scale. Google Play and Apple App Store have both exceeded 1.5 million apps, and already there are studies featuring over 1 million apps. We also expect to see more longitudinal studies: the sub-fields for prediction and release engineering studies lend themselves particularly well to longitudinal data, and both of these fields grew in 2015.

As many studies in Section 10 have shown, app cloning and replication is a common problem in app stores. It may be the case that app stores will not grow indefinitely, and may even shrink in size following a consolidation of unique apps present, possibly using the some of the techniques discussed in this survey. In the meantime, as app stores continue to grow, app discovery presents a crucial problem to newer apps or developers, and so we may expect to see an effort to improve discovery, such as a greater-tiered category system.

Opportunities. An avenue for future research concerns the extraction of non-technical information from app stores, and extracting samples of apps (cognisant of the App Sampling Problem). Cross-store studies are also an avenue for future research. Few studies have compared multiple app stores, yet there is potential to learn the differences between dominant stores, and lesser known or fledgling stores.

App stores provide us with with the unique opportunity to leverage customer, business and technical aspects of applications in the same place. Future app store analysis studies may seek to further combine all of these aspects to provide greater insights into the socio-technical business of developing for app stores.

Problems. Restricted data availability in app stores presents issues for researchers. We encourage app store owners not to impose such restrictions as: limiting the ranked list of apps to the top few hundred; limiting reviews to the most recent only. Such restrictions could reduce the scope and accuracy of future App Store Analysis studies. In addition, more detailed breakdowns of the prices attached to free apps that utilise in-app-purchases could lead to valuable research findings for app developers.

A concept that could be extremely valuable to researchers is that of a centralised repository of app store data that can be freely accessed, consisting of apps that are not just "free and open source". However, legal and copyright issues present potent barriers to the construction of such a repository from being created at present, and so this remains an open problem.

16 Conclusion

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.

Overall, we find a surprisingly wide and diverse set of techniques and applications in App Store Analysis, highlighting the health and future potential of the field. App Store Analysis opens up an exciting new vista for software engineering research which can connect and deeply understand relationships between social, technical and business facing aspects of software development, deployment and uptake in ways previously impossible due to paucity of data.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their very helpful feedback, and our many colleagues who responded with their valuable comments on an earlier version of this survey. This research was supported by EPRSC (DAASE grant no. EP/J017515).

REFERENCES

[1] N. A. S. Abdullah, N. I. A. Rusli, and M. F. Ibrahim, "Mobile game size estimation: Cosmic FSM rules, UML mapping model and unity 3d game engine," in *Proc. IEEE Conf. Open Syst.*, 2014, pp. 42–47.

[2] N. Agarwal, R. Karimpour, and G. Ruhe, "Theme-based product release planning: An analytical approach," in *Proc. 47th Hawaii Int. Conf. Syst. Sci.*, 2014, pp. 4739–4748.

[3] A. Al-Subaihin, et al., "App store mining and analysis," in *Proc. 3rd Int. Workshop Softw. Develop. Lifecycle Mobile*, 2015, pp. 1–2.

[4] A. Al-Subaihin, et al., "Clustering mobile apps based on mined textual features," in *Proc. 10th Int. Symp. Empirical Softw. Eng. Meas.*, 2016, pp. 38:1–38:10.

[5] C. Albanesius, "Mobile app reviews: Google 'Bouncer' now scanning Android Market for malware," 2012. [Online]. Available: http://uk.pcmag.com/apps/66697/news/google-bouncer-now-scanning-android-market-for-mal

[6] K. Alharbi and T. Yeh, "Collect, decompile, extract, stats, and diff: Mining design pattern changes in Android apps," in *Proc. 17th Int. Conf. Human-Comput. Interaction Mobile Devices Services*, 2015, pp. 515–524.

[7] Amazon.com, "Amazon Mechanical Turk," 2013. [Online]. Available: https://www.mturk.com/mturk/welcome

[8] androguard, "androguard," 2015. [Online]. Available: https://github.com/androguard/androguard

[9] 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 *Proc. Annu. Symp. Netw. Distrib. Syst. Secur.*, 2014.

[10] S. Arzt, et al., "FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," in *Proc. 35th ACM SIGPLAN Conf. Program. Language Des. Implementation*, 2014, pp. 259–269.

[11] G. Askalidis, "The impact of large scale promotions on the sales and ratings of mobile apps: Evidence from Apple's App Store," *Computing Research Repository CoRR*, vol. abs/1506.06857, pp. 1–32, 2015.

[12] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "PScout: Analyzing the Android permission specification," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 217–228.

[13] V. Avdiienko, et al., "Mining apps for abnormal usage of sensitive data," in *Proc. 37th Int. Conf. Softw. Eng.*, 2015, pp. 426–436.

[14] N. S. Awang Abu Bakar and I. Mahmud, "Empirical analysis of Android apps permissions," in *Proc. Int. Conf. Advanced Comput. Sci. Appl. Technol.*, 2013, pp. 406–411.

[15] S. A. Azad, "Empirical studies of Android API usage: Suggesting related API calls and detecting license violations," Master's thesis, Faculty Eng. Comput. Sci., Concordia Univ., Montréal, Canada, 2015.

[16] A. A. Bakar, N. S. Mahmud, and I. Mahmud, "OSSGrab: Software repositories and app store mining tool," *Lecture Notes Softw. Eng.*, vol. 1, no. 3, pp. 219–223, 2013.

[17] 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 *Proc. 17th ACM Conf. Comput. Commun. Secur.*, 2010, pp. 73–84.

[18] 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," *IEEE Trans. Softw. Eng.*, vol. 40, no. 6, pp. 617–632, Jun. 2014.

[19] O. Bastani, S. Anand, and A. Aiken, "Interactively verifying absence of explicit information flows in Android apps," in *Proc. ACM SIGPLAN Int. Conf. Object-Oriented Program. Syst. Languages Appl.*, 2015, pp. 299–315.

[20] 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 *Proc. 6th Int. Conf. Malicious Unwanted Softw.*, 2011, pp. 66–72.

[21] 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 Trans. Softw. Eng.*, vol. 41, no. 4, pp. 384–407, Apr. 2015.

[22] 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 *Proc. Int. Conf. Intell. User Interfaces*, 2013, pp. 297–306.

[23] D. L. Ben Lulu and T. Kuflik, "Wise mobile icons organization: Apps taxonomy classification using functionality mining to ease apps finding," *Mobile Inf. Syst.*, vol. 2016, 2015, article ID: 3083450.

[24] G. Berardi, A. Esuli, T. Fagni, and F. Sebastiani, "Multi-store metadata-based supervised mobile app classification," in *Proc. 30th Annu. ACM Symp. Appl. Comput.*, 2015, pp. 585–588.

[25] 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 *Proc. 17th Eur. Conf. Softw. Maintenance Reengineering*, 2013, pp. 133–143.

[26] R. Bhoraskar, et al., "Brahmastra: Driving apps to test the security of third-party components," in *Proc. 23rd USENIX Conf. Secur. Symp.*, 2014, pp. 1021–1036.

[27] T. Bläsing, L. Batyuk, A. Schmidt, S. A. Çamtepe, and S. Albayrak, "An Android application sandbox system for suspicious software detection," in *Proc. 5th Int. Conf. Malicious Unwanted Softw.*, 2010, pp. 55–62.

[28] 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 *Proc. 13th Int. Conf. Human Comput. Interaction Mobile Devices Services*, 2011, pp. 47–56.

[29] T. Book, A. Pridgen, and D. S. Wallach, "Longitudinal analysis of Android ad library permissions," *Computing Research Repository CoRR*, vol. abs/1303.0857, pp. 1–9, 2013.

[30] H. S. Borges and M. T. Valente, "Mining usage patterns for the Android API," *Peer J. Comput. Sci.*, vol. 1, 2015, Art. no. e12.

[31] M. Butler, "Android: Changing the mobile landscape," *IEEE Pervasive Comput.*, vol. 10, no. 1, pp. 4–7, Jan. 2011.

[32] J. Callaham, "Google says there are now 1.4 billion active Android devices worldwide," 2015. [Online]. Available: http://www.androidcentral.com/google-says-there-are-now-14-billion-active-android-devices-worldwide

[33] B. Carbunar and R. Potharaju, "A longitudinal study of the Google app market," in *Proc. IEEE/ACM Int. Conf. Advances Social Netw. Anal. Mining*, 2015, pp. 242–249.

[34] 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, Jul.–Aug. 2015.

[35] L. Cen, D. Kong, H. Jin, and L. Si, "Mobile app security risk assessment: A crowdsourcing ranking approach from user comments," in *Proc. SIAM Int. Conf. Data Mining*, 2015, pp. 658–666.

[36] L. Cen, L. Si, N. Li, and H. Jin, "User comment analysis for Android apps and CSPI detection with comment expansion," in *Proc. 1st Int. Workshop on Privacy-Preserving IR*, 2014, pp. 25–30.

[37] S. Chakradeo, B. Reaves, P. Traynor, and W. Enck, "Mast: Triage for market-scale mobile Malware analysis," in *Proc. 6th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2013, pp. 13–24.

[38] R. Chandy and H. Gu, "Identifying spam in the iOS App Store," in *Proc. 2nd Joint WICOW/AIRWeb Workshop Web Quality*, 2012, pp. 56–59.

[39] K. Chen, et al., "Finding unknown malice in 10 seconds: Mass vetting for new threats at the Google-Play scale," in *Proc. 24th USENIX Conf. Secur. Symp.*, 2015, pp. 659–674.

[40] M. Chen and X. Liu, "Predicting popularity of online distributed applications: iTunes App Store case analysis," in *Proc. iConf.*, 2011, pp. 661–663.

[41] 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 *Proc. 36th Int. Conf. Softw. Eng.*, 2014, pp. 767–778.

[42] 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 *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 201–212.

[43] P. H. Chia, Y. Yamamoto, and N. Asokan, "Is this app safe?: A large scale study on application permissions and risk signals," in *Proc. 21st Int. Conf. World Wide Web*, 2012, pp. 311–320.

[44] L. Cocco, K. Mannaro, G. Concas, and M. Marchesi, "Simulation of the best ranking algorithms for an app store," in *Mobile Web Information Systems*. Berlin, Germany: Springer, 2014, vol. 8640, pp. 233–247.

[45] S. Comino, F. M. Manenti, and F. Mariuzzo, "Updates management in mobile applications. iTunes vs Google Play," Centre for Competition Policy, Univ. East Anglia, Norwich, U.K., 2015.

[46] F. T. Commission, "Complying with COPPA: Frequently Asked Questions," 2015. [Online]. Available: https://www.ftc.gov/tips-advice/business-center/guidance/complying-coppa-frequently-asked-questions

[47] L. Corral and I. Fronza, "Better code for better apps: A study on source code quality and market success of Android applications," in *Proc. 2nd ACM Int. Conf. Mobile Softw. Eng. Syst.*, 2015, pp. 22–32.

[48] COSMIC, "Common software measurement international consortium," 2015. [Online]. Available: http://cosmic-sizing.org/

[49] P. Coulton and W. Bamford, "Experimenting through mobile 'apps' and 'app stores'," *Int. J. Mobile Human Comput. Interact.*, vol. 3, no. 4, pp. 55–70, 2011.

[50] J. Crussell, C. Gibler, and H. Chen, "Attack of the clones: Detecting cloned applications on Android markets," in *Proc. 17th Eur. Symp. Res. Comput. Secur.*, 2012, pp. 37–54.

[51] J. Crussell, C. Gibler, and H. Chen, "Andarwin: Scalable detection of semantically similar Android applications," in *Proc. 18th Eur. Symp. Res. Comput. Secur.*, 2013, pp. 182–199.

[52] J. Crussell, R. Stevens, and H. Chen, "MadFraud: Investigating ad fraud in Android applications," in *Proc. 12th Annu. Int. Conf. Mobile Syst. Appl. Services*, 2014, pp. 123–134.

[53] DAI-Labor, "Androlyzer," 2015. [Online]. Available: https://androlyzer.com/

[54] D. Datta and S. Kajanan, "Do app launch times impact their subsequent commercial success? an analytical approach," in *Proc. Int. Conf. Cloud Comput. Big Data*, 2013, pp. 205–210.

[55] L. D'Avanzo, F. Ferrucci, C. Gravino, and P. Salza, "Cosmic functional measurement of mobile applications and code size estimation," in *Proc. 30th Annu. ACM Symp. Appl. Comput.*, 2015, pp. 1631–1636.

[56] Z. Deng, B. Saltaformaggio, X. Zhang, and D. Xu, "iRis: Vetting private API abuse in iOS applications," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 44–56.

[57] M. L. Dering and P. McDaniel, "Android Market reconstruction and analysis," in *Proc. IEEE Military Commun. Conf.*, 2014, pp. 300–305.

[58] 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 Comput. Commun. Rev.*, vol. 16, no. 2, pp. 16–27, 2012.

[59] W. Enck, et al., "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *Proc. 9th USENIX Conf. Operating Syst. Des. Implementation*, 2010, pp. 1–6.

[60] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *Proc. 16th ACM Conf. Comput. Commun. Secur.*, 2009, pp. 235–245.

[61] 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.

[62] 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. Mobile Human Comput. Interact.*, vol. 4, no. 3, pp. 28–43, 2012.

[63] F. Ferrucci, C. Gravino, P. Salza, and F. Sarro, "Investigating functional and code size measures for mobile applications," in *Proc. 41st Euromicro Conf. Series Softw. Eng. Advanced Appl.*, 2015, pp. 365–368.

[64] F. Ferrucci, C. Gravino, P. Salza, and F. Sarro, "Investigating functional and code size measures for mobile applications: A replicated study," in *Proc. 16th Int. Conf. Product-Focused Softw. Process Improvement*, 2015, pp. 271–287.

[65] 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," Univ. College London, London, U.K., Tech. Rep. rN/14/10, 2014.

[66] 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 *Proc. 41st Euromicro Conf. Series Softw. Eng. Advanced Appl.*, 2015, pp. 357–364.

[67] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Mach. Learn.*, vol. 29, no. 2/3, pp. 131–163, 1997.

[68] 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 *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 1276–1284.

[69] L. V. Galvis Carreño and K. Winbladh, "Analysis of user comments: An approach for software requirements evolution," in *Proc. Int. Conf. Softw. Eng.*, 2013, pp. 582–591.

[70] C. Gao, H. Xu, J. Hu, and Y. Zhou, "Ar-tracker: Track the dynamics of mobile apps via user review mining," in *Proc. IEEE Symp. Service-Oriented Syst. Eng.*, 2015, pp. 284–290.

[71] R. Garg and R. Telang, "Inferring app demand from publicly available data," *MIS Quart.*, vol. 37, no. 4, pp. 1253–1264, 2013.

[72] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "AndroidLeaks: Automatically detecting potential privacy leaks in Android applications on a large scale," in *Proc. 5th Int. Conf. Trust Trustworthy Comput.*, 2012, pp. 291–307.

[73] C. Gibler, R. Stevens, J. Crussell, H. Chen, H. Zang, and H. Choi, "Adrob: Examining the landscape and impact of Android application plagiarism," in *Proc. 11th Annu. Int. Conf. Mobile Syst. Appl. Services*, 2013, pp. 431–444.

[74] GinLemon, "Smart launcher 3—simple. light. fast," 2011. [Online]. Available: http://www.smartlauncher.net/

[75] GitHub, Inc., "Github," 2014. [Online]. Available: https://github.com/, 2014.

[76] W. Glodek and R. Harang, "Rapid permissions-based detection and analysis of mobile malware using random decision forests," in *Proc. IEEE Military Commun. Conf.*, 2013, pp. 980–985.

[77] M. Gómez, M. Martinez, M. Monperrus, and R. Rouvoy, "When app stores listen to the crowd to fight bugs in the wild," in *Proc. 37th Int. Conf. Softw. Eng.*, 2015, vol. 2, pp. 567–570.

[78] M. Gomez, R. Rouvoy, M. Monperrus, and L. Seinturier, "A recommender system of buggy app checkers for app store moderators," in *Proc. 2nd ACM Int. Conf. Mobile Softw. Eng. Syst.*, 2015, pp. 1–11.

[79] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proc. Int. Conf. Softw. Eng.*, 2014, pp. 292–302.

[80] M. Goul, O. Marjanovic, S. Baxley, and K. Vizecky, "Managing the enterprise business intelligence app store: Sentiment analysis supported requirements engineering," in *Proc. 45th Hawaii Int. Conf. System Sci.*, 2012, pp. 4168–4177.

[81] M. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements," in *Proc. 5th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2012, pp. 101–112.

[82] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker: Scalable and accurate zero-day Android malware detection," in *Proc. 10th Int. Conf. Mobile Syst. Appl. Services*, 2012, pp. 281–294.

[83] X. Gu and S. Kim, "What parts of your apps are loved by users?" in *Proc. 30th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2015, pp. 760–770.

[84] L. Guerrouj, S. Azad, and P. C. Rigby, "The influence of App churn on App success and StackOverflow discussions," in *Proc. 22nd Int. Conf. Softw. Anal. Evol. Reengineering*, 2015, pp. 321–330.

[85] J. Gui, S. Mcilroy, M. Nagappan, and W. G. J. Halfond, "Truth in advertising: The hidden cost of mobile ads for software developers," in *Proc. 37th Int. Conf. Softw. Eng.*, 2015, vol. 1, pp. 100–110.

[86] E. Guzman, O. Aly, and B. Bruegge, "Retrieving diverse opinions from app reviews," in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2015, pp. 1–10.

[87] E. Guzman, M. El-Haliby, and B. Bruegge, "Ensemble methods for app review classification: An approach for software evolution (N)," in *Proc. 30th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2015, pp. 771–776.

[88] E. Guzman and W. Maalej, "How do users like this feature? a fine grained sentiment analysis of app reviews," in *Proc. 22nd IEEE Int. Requirements Eng. Conf.*, 2014, pp. 153–162.
[89] E. Ha and D. Wagner, "Do Android users write about electric sheep? examining consumer reviews in Google Play," in *Proc. 10th IEEE Consumer Commun. Netw. Conf.*, 2013, pp. 149–157.
[90] Y. J. Ham and H.-W. Lee, "Detection of malicious Android mobile applications based on aggregated system call events," *Int. J. Comput. Commun. Eng.*, vol. 3, no. 2, pp. 149–154, 2014.
[91] 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 *Proc. 12th Int. Conf. Mobile Syst. Appl. Services*, 2014, pp. 204–217.
[92] M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: MSR for app stores," in *Proc. 9th IEEE Working Conf. Mining Softw. Repositories*, 2012, pp. 108–111.
[93] 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 *Proc. IEEE Int. Conf. Big Data*, 2015, pp. 1043–1052.
[94] H. Heitkötter, S. Hanschke, and T. A. Majchrzak, "Evaluating cross-platform development approaches for mobile applications," in *Proc. 8th Int. Conf. Web Inf. Syst. Technol.*, 2012, pp. 120–138.
[95] N. Henze and S. Boll, "Release your app on Sunday eve: Finding the best time to deploy apps," in *Proc. 13th Int. Conf. Human Comput. Interaction Mobile Devices Services*, 2011, pp. 581–586.
[96] A. Hindle, "Green software engineering: The curse of methodology," in *Proc. IEEE 23rd Int. Conf. Softw. Anal. Evol. Reengineering*, 2016, vol. 5, pp. 46–55.
[97] T.-H. Ho, D. Dean, X. Gu, and W. Enck, "PREC: Practical root exploit containment for Android devices," in *Proc. 4th ACM Conf. Data Appl. Secur. Privacy*, 2014, pp. 187–198.
[98] L. Hoon, M. Rodriguez-Garca, R. Vasa, R. Valencia-Garca, and J.-G. Schneider, "App reviews: Breaking the user and developer language barrier," in *Proc. 4th Int. Conf. Trends Appl. Softw. Eng.*, 2016, vol. 405, pp. 223–233.
[99] L. Hoon, R. Vasa, G. Y. Martino, J.-G. Schneider, and K. Mouzakis, "Awesome!: Conveying satisfaction on the app store," in *Proc. 25th Australian Comput.-Human Interaction Conf.: Augmentation Appl. Innovation, Collaboration*, 2013, pp. 229–232.
[100] L. Hoon, R. Vasa, J.-G. Schneider, and J. Grundy, "An analysis of the mobile app review landscape: Trends and implications," Faculty Inf. Commun. Technol., Swinburne Univ. Technol., Hawthorn Vic, Australia, 2013.
[101] L. Hoon, R. Vasa, J.-G. Schneider, and K. Mouzakis, "A preliminary analysis of vocabulary in mobile app user reviews," in *Proc. 24th Australian Comput.-Human Interaction Conf.*, 2012, pp. 245–248.
[102] J. Huang, et al., "SUPOR: Precise and scalable sensitive user input detection for Android apps," in *Proc. 24th USENIX Conf. Secur. Symp.*, 2015, pp. 977–992.
[103] W. Huang, Y. Dong, A. Milanova, and J. Dolby, "Scalable and precise taint analysis for Android," in *Proc. Int. Symp. Softw. Testing Anal.*, 2015, pp. 106–117.
[104] C. Iacob and R. Harrison, "Retrieving and analyzing mobile apps feature requests from online reviews," in *Proc. 10th Working Conf. Mining Softw. Repositories*, 2013, pp. 41–44.
[105] C. Iacob, R. Harrison, and S. Faily, "Online reviews as first class artifacts in mobile app development," in *Proc. 5th Int. Conf. Mobile Comput. Appl. Services*, 2014, pp. 47–53.
[106] C. Iacob, V. Veerappa, and R. Harrison, "What are you complaining about?: A study of online reviews of mobile applications," in *Proc. 27th Int. BCS Human Comput. Interaction Conf.*, 2013, pp. 29:1–29:6.
[107] S.-Y. Ihm, W.-K. Loh, and Y.-H. Park, "App analytic: A study on correlation analysis of app ranking data," in *Proc. Int. Conf. Cloud Green Comput.*, 2013, pp. 561–563.
[108] J. Jeon, et al., "Dr. Android and Mr. Hide: Fine-grained permissions in Android applications," in *Proc. 2nd ACM Workshop Secur. Privacy Smartphones Mobile Devices*, 2012, pp. 3–14.
[109] H. Jiang, H. Ma, Z. Ren, J. Zhang, and X. Li, "What makes a good app description?" in *Proc. 6th Asia-Pacific Symp. Internetware Internetware*, 2014, pp. 45–53.
[110] Y. Jing, G.-J. Ahn, Z. Zhao, and H. Hu, "RiskMon: Continuous and automated risk assessment of mobile applications," in *Proc. 4th ACM Conf. Data Appl. Secur. Privacy*, 2014, pp. 99–110.
[111] Y. Jo and A. H. Oh, "Aspect and sentiment unification model for online review analysis," in *Proc. 4th ACM Int. Conf. Web Search Data Mining*, 2011, pp. 815–824.
[112] M. E. Joorabchi, M. Ali, and A. Mesbah, "Detecting inconsistencies in multi-platform mobile apps," in *Proc. 26th IEEE Int. Symp. Softw. Rel. Eng.*, 2015, pp. 450–460.
[113] E.-Y. Jung, C. Baek, and J.-D. Lee, "Product survival analysis for the App Store," *Marketing Lett.*, vol. 23, no. 4, pp. 929–941, 2012.
[114] 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 Trans. Softw. Eng.*, vol. 33, no. 4, pp. 34–39, 2016.
[115] H. Khalid, "On identifying user complaints of iOS apps," in *Proc. Int. Conf. Softw. Eng.*, 2013, pp. 1474–1476.
[116] 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 *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2014, pp. 610–620.
[117] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan, "What do mobile app users complain about?" *IEEE Softw.*, vol. 32, no. 3, pp. 70–77, May/Jun. 2015.
[118] M. Khalid, M. Asif, and U. Shehzaib, "Towards improving the quality of mobile app reviews," *Int. J. Inf. Technol. Comput. Sci.*, vol. 7, no. 10, 2015, Art. no. 35.
[119] M. Khalid, U. Shehzaib, and M. Asif, "A case of mobile app reviews as a crowdsource," *Int. J. Inf. Eng. Electron. Business*, vol. 7, no. 5, 2015, Art. no. 39.
[120] K. Khanmohammadi, M. R. Rejali, and A. Hamou-Lhadj, "Understanding the service life cycle of Android apps: An exploratory study," in *Proc. 5th Annu. ACM CCS Workshop Secur. Privacy Smartphones Mobile Devices*, 2015, pp. 81–86.
[121] D. Kim, A. Gokhale, V. Ganapathy, and A. Srivastava, "Detecting plagiarized mobile apps using API birthmarks," *Autom. Softw. Eng.*, vol. 23, no. 4, pp. 591–618, Dec. 2016.
[122] 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.
[123] D. E. Krutz, et al., "A dataset of open-source Android applications," in *Proc. 12th Working Conf. Mining Softw. Repositories*, 2015, pp. 522–525.
[124] N. Lageman, M. Lindsey, and W. Glodek, "Detecting malicious Android applications from runtime behavior," in *Proc. IEEE Military Commun. Conf.*, 2015, pp. 324–329.
[125] G. Lee and T. S. Raghu, "Product portfolio and mobile apps success: Evidence from App Store market," in *Proc. 17th Americas Conf. Inf. Syst.*, 2011, pp. 3912–3921.
[126] G. Lee and T. Raghu, "Determinants of mobile apps' success: Evidence from the app store market," *J. Manage. Inf. Syst.*, vol. 31, no. 2, pp. 133–170, 2014.
[127] L. Li, et al., "ICCTA: Detecting inter-component privacy leaks in Android apps," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, 2015, pp. 280–291.
[128] 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," *Int. J. Electron. Commerce*, vol. 20, no. 2, pp. 236–260, 2015.
[129] 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 Trans. Softw. Eng.*, vol. 41, no. 1, pp. 40–64, Jan. 2015.
[130] S. L. Lim and P. J. Bentley, "App epidemics: Modelling the effects of publicity in a mobile app ecosystem," in *Proc. 13th Int. Conf. Simul. Synthesis Living Syst.*, 2012, pp. 202–209.
[131] 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.
[132] S. L. Lim and P. J. Bentley, "Investigating app store ranking algorithms using a simulation of mobile app ecosystems," in *Proc. IEEE Congress Evol. Comput.*, 2013, pp. 2672–2679.
[133] 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 Trans. Evol. Comput.*, vol. 20, no. 4, pp. 529–545, 2016.
[134] 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 *Proc. ACM Conf. Ubiquitous Comput.*, 2012, pp. 501–510.
[135] 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 *Proc. 36th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2013, pp. 283–292.

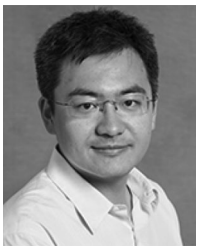
[136] J. Lin, K. Sugiyama, M.-Y. Kan, and T.-S. Chua, "New and improved: Modeling versions to improve app recommendation," in *Proc. 37th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2014, pp. 647–656.
[137] M. Linares-Vásquez, "Supporting evolution and maintenance of Android apps," in *Proc. 36th Int. Conf. Softw. Eng.*, 2014, pp. 714–717.
[138] 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 *Proc. 9th Joint Meeting Found. Softw. Eng.*, 2013, pp. 477–487.
[139] 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 *Proc. 11th Working Conf. Mining Softw. Repositories*, 2014, pp. 2–11.
[140] 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 *Proc. 11th Working Conf. Mining Softw. Repositories*, 2014, pp. 242–251.
[141] 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 *Proc. 8th ACM Int. Conf. Web Search Data Mining*, 2015, pp. 315–324.
[142] B. Liu, B. Liu, H. Jin, and R. Govindan, "Efficient privilege de-escalation for ad libraries in mobile apps," in *Proc. 13th Annu. Int. Conf. Mobile Syst. Appl. Services*, 2015, pp. 89–103.
[143] B. Liu, S. Nath, R. Govindan, and J. Liu, "Decaf: Detecting and characterizing ad fraud in mobile apps," in *Proc. 11th USENIX Conf. Netw. Syst. Des. Implementation*, 2014, pp. 57–70.
[144] X. Liu and J. Liu, "A two-layered permission-based Android Malware detection scheme," in *Proc. 2nd IEEE Int. Conf. Mobile Cloud Comput. Services Eng.*, 2014, pp. 142–148.
[145] S. Ma, S. Wang, D. Lo, R. H. Deng, and C. Sun, "Active semi-supervised approach for checking app behavior against its description," in *Proc. 39th IEEE Annu. Comput. Softw. Appl. Conf.*, 2015, vol. 2, pp. 179–184.
[146] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? on automatically classifying app reviews," in *Proc. IEEE 23rd Int. Requirements Eng. Conf.*, 2015, pp. 116–125.
[147] I. Malavolta, S. Ruberto, T. Soru, and V. Terragni, "End users' perception of hybrid mobile apps in the Google Play store," in *Proc. 4th Int. Conf. Mobile Services*, 2015, pp. 25–32.
[148] I. Malavolta, S. Ruberto, V. Terragni, and T. Soru, "Hybrid mobile apps in the Google Play store: An exploratory investigation," in *Proc. 2nd ACM Int. Conf. Mobile Softw. Eng. Syst.. ACM*, 2015, pp. 56–59.
[149] L. M. Manevitz and M. Yousef, "One-class SVMs for document classification," *J. Mach. Learn. Res.*, vol. 2, pp. 139–154, 2002.
[150] K. Mao, L. Capra, M. Harman, and Y. Jia, "A survey of the use of crowdsourcing in software engineering," Univ. College London, London, U.K., Tech. Rep. rN/15/01, 2016.
[151] K. Mao, M. Harman, and Y. Jia, "Sapienz: Multi-objective automated testing for Android applications," in *Proc. 25th Int. Symp. Softw. Testing Anal.*, 2016, pp. 94–105.
[152] marketsandmarkets.com, "World mobile applications market-advanced technologies, global forecast," 2010. [Online]. Available: http://www.marketsandmarkets.com/Market-Reports/mobile-applications-228.html
[153] W. Martin, "Causal impact for app store analysis," in *Proc. 38th Int. Conf. Softw. Eng.*, 2016, pp. 659–661.
[154] W. Martin, M. Harman, Y. Jia, F. Sarro, and Y. Zhang, "The app sampling problem for app store mining," in *Proc. 12th IEEE Working Conf. Mining Softw. Repositories*, 2015, pp. 123–133.
[155] W. Martin, F. Sarro, and M. Harman, "Causal impact analysis applied to app releases in Google Play and Windows Phone Store," Univ. College London, London, U.K., Tech. Rep. rN/15/07, 2015.
[156] W. Martin, F. Sarro, and M. Harman, "Causal impact analysis for app releases in Google Play," in *Proc. 24th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2016, pp. 435–446.
[157] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A survey of app store analysis for software engineering," Univ. College London, London, U.K., Tech. Rep. rN/16/02, 2016.
[158] T. McDonnell, B. Ray, and M. Kim, "An empirical study of API stability and adoption in the Android ecosystem," in *Proc. IEEE Int. Conf. Softw. Maintenance*, 2013, pp. 70–79.
[159] S. McIlroy, N. Ali, and A. E. Hassan, "Fresh apps: An empirical study of frequently-updated mobile apps in the google play store," *Empirical Softw. Eng.*, vol. 21, no. 3, pp. 1346–3270, Jun. 2016.
[160] 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 Softw. Eng.*, vol. 21, no. 3, pp. 1067–1106, Jun. 2016.
[161] 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 Softw.*, vol. PP, no. 99, 2015.
[162] D. Mimno, H. Wallach, E. Talley, M. Leenders, and A. McCallum, "Optimizing semantic coherence in topic models," in *Proc. Conf. Empirical Methods Natural Language Process.*, 2011, pp. 262–272.
[163] R. Minelli and M. Lanza, "Samoa—a visual software analytics platform for mobile applications," in *Proc. 29th Int. Conf. Softw. Maintenance*, 2013, pp. 476–479.
[164] R. Minelli and M. Lanza, "Software analytics for mobile applications–insights & lessons learned," in *Proc. 15th Eur. Conf. Softw. Maintenance Reengineering*, 2013, pp. 144–153.
[165] 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 Softw.*, vol. 33, no. 6, pp. 86–92, Nov./Dec. 2016.
[166] S. Mokarizadeh, M. T. Rahman, and M. Matskin, "Mining and analysis of apps in Google Play," in *Proc. 9th Int. Conf. Web Inf. Syst. Technol.*, 2013, pp. 527–535.
[167] J. E. Montandon, H. Borges, D. Felix, and M. T. Valente, "Documenting APIs with examples: Lessons learned with the APIMiner platform," in *Proc. IEEE 20th Working Conf. Reverse Eng.*, 2013, pp. 401–408.
[168] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, and D. Poshyvanyk, "Auto-completing bug reports for Android applications," in *Proc. 10th Joint Meeting Found. Softw. Eng.*, 2015, pp. 673–686.
[169] P. Mutchler, A. Doupé, J. Mitchell, C. Kruegel, and G. Vigna, "A large-scale study of mobile web app security," in *Proc. Mobile Secur. Technol. Workshop*, 2015, pp. 1–11.
[170] 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 *Proc. 3rd Int. Workshop Res. Large*, 2012, pp. 3–6.
[171] M. Nagappan and E. Shihab, "Future trends in software engineering research for mobile apps," in *Proc. 23rd IEEE Int. Conf. Softw. Anal. Evol. Reengineering*, 2016, pp. 21–32.
[172] M. Nayebi and G. Ruhe, "Trade-off service portfolio planning—a case study on mining the Android app market," *PeerJ PrePrints*, vol. 3, 2015, Art. no. e1671.
[173] Y. Y. Ng, H. Zhou, Z. Ji, H. Luo, and Y. Dong, "Which Android app store can be trusted in China?" in *Proc. IEEE 38th Annu. Comput. Softw. Appl. Conf.*, 2014, pp. 509–518.
[174] J. Oh, D. Kim, U. Lee, J.-G. Lee, and J. Song, "Facilitating developer-user interactions with mobile app review digests," in *Proc. Extended Abstracts Human Factors Comput. Syst.*, 2013, pp. 1809–1814.
[175] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study," in *Proc. 21st. IEEE Int. Requirements Eng. Conf.*, 2013, pp. 125–134.
[176] F. Palomba, et al., "User reviews matter! tracking crowdsourced reviews to support evolution of successful apps," in *Proc. 31st Int. Conf. Softw. Maintenance Evol.*, 2015, pp. 291–300.
[177] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "WhyPer: Towards automating risk assessment of mobile applications," in *Proc. 22nd USENIX Conf. Secur.*, 2013, pp. 527–542.
[178] 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," in *Proc. 31st IEEE Int. Conf. Softw. Maintenance Evol.*, 2015, pp. 281–290.
[179] D. H. Park, M. Liu, C. Zhai, and H. Wang, "Leveraging user reviews to improve accuracy for mobile app retrieval," in *Proc. 38th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2015, pp. 533–542.
[180] N. Peiravian and X. Zhu, "Machine learning for Android Malware detection using permission and API calls," in *Proc. IEEE 25th Int. Conf. Tools Artif. Intell.*, 2013, pp. 300–305.

[181] H. Peng, et al., "Using probabilistic generative models for ranking risks of Android apps," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 241–252.
[182] 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 *Proc. Conf. Internet Meas. Conf.*, 2013, pp. 277–290.
[183] R. Potharaju, A. Newell, C. Nita-Rotaru, and X. Zhang, "Plagiarizing smartphone applications: Attack strategies and defense techniques," in *Proc. 4th Int. Conf. Eng. Secure Softw. Syst.*, 2012, pp. 106–120.
[184] T. Preuss, "Mobile applications, function points and cost estimating," in *Proc. Int. Cost Estimation Anal. Assoc. Conf.*, 2013.
[185] T. Preuss, "Mobile applications, functional analysis, and the customer experience," in *The IFPUG Guide to IT and Software Measurement, IFPUG*, Ed. Boca Raton FL, USA: Auerbach Publications, 2012, pp. 408–433.
[186] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen, "AutoCog: Measuring the description-to-permission fidelity in Android applications," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 1354–1365.
[187] V. Rastogi, Y. Chen, and W. Enck, "AppsPlayground: Automatic security analysis of smartphone applications," in *Proc. 3rd ACM Conf. Data Appl. Secur. Privacy*, 2013, pp. 209–220.
[188] L. Ravindranath, S. Nath, J. Padhye, and H. Balakrishnan, "Automatic and scalable fault detection for mobile applications," in *Proc. 12th Annu. Int. Conf. Mobile Syst. Appl. Services*, 2014, pp. 190–203.
[189] A.-D. Rein and J. Münch, "Feature prioritization based on mock-purchase: A mobile case study," in *Proc. Lean Enterprise Softw. Syst. Conf.*, 2013, pp. 165–179.
[190] S. Robertson, "Understanding inverse document frequency: On theoretical arguments for idf," *J. Documentation*, vol. 60, pp. 503–520, 2004.
[191] S. Roy, et al., "Experimental study with real-world data for Android app security analysis using machine learning," in *Proc. 31st Annu. Comput. Secur. Appl. Conf.*, 2015, pp. 81–90.
[192] 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 Softw.*, vol. 31, no. 2, pp. 78–86, Mar./Apr. 2014.
[193] 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 Softw.*, vol. 31, no. 6, pp. 86–92, Nov./Dec. 2014.
[194] I. J. M. Ruiz, M. Nagappan, B. Adams, T. Berger, S. Dienst, and A. E. Hassan, "Analyzing ad library updates in android apps," *IEEE Softw.*, vol. 33, no. 2, pp. 74–80, Mar./Apr. 2016.
[195] I. J. M. Ruiz, M. Nagappan, B. Adams, and A. E. Hassan, "Understanding reuse in the Android market," in *Proc. 20th IEEE Int. Conf. Program Comprehension*, 2012, pp. 113–122.
[196] 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 *Proc. 5th ACM SIGCHI Symp. Eng. Interactive Comput. Syst.*, 2013, pp. 275–284.
[197] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, and P. G. Bringas, "On the automatic categorisation of Android applications," in *Proc. IEEE Consum. Commun. Netw. Conf.*, 2012, pp. 149–153.
[198] B. Sanz, et al., "MAMA: Manifest analysis for malware detection in Android," *Cybern. Syst. Intell. Netw. Secur. Survivability*, vol. 44, no. 6/7, pp. 469–488, 2013.
[199] B. Sanz, I. Santos, J. Nieves, C. Laorden, I. Alonso-Gonzalez, and P. G. Bringas, "Mads: Malicious Android applications detection through string analysis," in *Proc. 7th Int. Conf. Netw. Syst. Secur.*, 2013, pp. 178–191.
[200] B. Sanz, I. Santos, X. Ugarte-Pedrero, C. Laorden, J. Nieves, and P. G. Bringas, "Instance-based anomaly method for Android malware detection," in *Proc. 10th Int. Conf. Secur. Cryptography*, 2013, pp. 387–394.
[201] F. Sarro, "The UCLAppA repository: A repository of research articles on mobile software engineering and app store analysis," (2013). [Online]. Available: http://www0.cs.ucl.ac.uk/staff/F.Sarro/projects/UCLappA/UCLappArepository.html
[202] F. Sarro, A. 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 *Proc. 23rd IEEE Int. Requirements Eng. Conf.*, 2015, pp. 76–85.
[203] J. Schütte, R. Fedler, and D. Titze, "ConDroid: Targeted dynamic analysis of Android applications," in *Proc. 29th IEEE Int. Conf. Advanced Inf. Netw. Appl.*, 2015, pp. 571–578.
[204] S. Seneviratne, H. Kolamunna, and A. Seneviratne, "A measurement study of tracking in paid mobile applications," in *Proc. 8th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2015, pp. 7:1–7:6.
[205] S. Seneviratne, A. Seneviratne, M. A. Kaafar, A. Mahanti, and P. Mohapatra, "Early detection of spam mobile apps," in *Proc. 24th Int. Conf. World Wide Web*, 2015, pp. 949–959.
[206] G. Sethumadhavan, "Sizing Android mobile applications," presented at 6th IFPUG Int. Softw. Meas. Anal. Conf, Sao Paulo, Brazil, 2011.
[207] A. Shabtai, Y. Fledel, and Y. Elovici, "Automated static code analysis for classifying Android applications using machine learning," in *Proc. Int. Conf. Comput. Intell. Secur.*, 2010, pp. 329–333.
[208] C. Sharma, "Sizing up the global mobile apps market," Chetan Sharma Consulting, Issaquah, WA, USA, vol. 659, 2010.
[209] K. Shi and K. Ali, "Getjar mobile application recommendations with very sparse datasets," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2012, pp. 204–212.
[210] C. Shuler, "iLearnII; An analysis of the education category of the iTunes App Store," Joan Ganz Cooney Center, Sesame Workshop, New York, NY, USA, 2012.
[211] R. Stevens, J. Ganz, V. Filkov, P. Devanbu, and H. Chen, "Asking for (and about) permissions used by Android apps," in *Proc. 10th Working Conf. Mining Softw. Repositories*, 2013, pp. 31–40.
[212] Z. Svedic, "The effect of informational signals on mobile apps sales ranks across the globe," Ph.D. dissertation, Beedie School Business Faculty, Simon Fraser Univ., Burnaby, BC, USA, 2015.
[213] 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 *Proc. IEEE 11th Int. Working Conf. Source Code Anal. Manipulation*, 2011, pp. 55–64.
[214] M. D. Syer, M. Nagappan, B. Adams, and A. E. Hassan, "Studying the relationship between source code quality and mobile platform dependence," *Softw. Quality J.*, vol. 23, no. 3, pp. 485–508, 2015.
[215] 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 *Proc. Conf. Center Advanced Studies Collaborative Res.*, 2013, pp. 283–297.
[216] J. Tan, et al., "The effect of developer-specified explanations for permission requests on smartphone user behavior," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2014, pp. 91–100.
[217] 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)," *Secur. Commun. Netw.*, vol. 9, pp. 389–419, 2013.
[218] P. Teufl, et al., "Android Market analysis with activation patterns," in *Security and Privacy in Mobile Information and Communication Systems*, vol. 94. Berlin, Germany: Springer, 2012, pp. 1–12.
[219] 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 *Proc. 31st Int. Conf. Softw. Maintenance Evol.*, 2015, pp. 1–10.
[220] Y.-X. Tong, J. She, and L. Chen, "Towards better understanding of app functions," *J. Comput. Sci. Technol.*, vol. 30, no. 5, pp. 1130–1140, 2015.
[221] Université du Luxembourg, "Androzoo," 2016. [Online]. Available: https://androzoo.uni.lu/
[222] S. Vakulenko, O. Müller, and J. V. Brocke, "Enriching iTunes App Store categories via topic modeling," in *Proc. Int. Conf. Inf. Syst.*, 2014, pp. 1–11.
[223] H. van Heeringen and E. Van Gorp, "Measure the functional size of a mobile app: Using the cosmic functional size measurement method," in *Proc. Conf. Int. Workshop Softw. Meas. Int. Conf. Softw. Process Product Meas. Joint Conf. Int. Workshop*, 2014, pp. 11–16.
[224] R. Vasa, L. Hoon, K. Mouzakis, and A. Noguchi, "A preliminary analysis of mobile app user reviews," in *Proc. 24th Australian Comput.-Human Interaction Conf.*, 2012, pp. 241–244.
[225] N. Viennot, "GitHub - nviennot/playdrone: Google Play Crawler," 2014. [Online]. Available: https://github.com/nviennot/playdrone

[226] N. Viennot, E. Garcia, and J. Nieh, "A measurement study of Google Play," in *Proc. ACM Int. Conf. Meas. Model. Comput. Syst.*, 2014, pp. 221–233.
[227] L. Vigneri, J. Chandrashekar, I. Pefkianakis, and O. Heen, "Taming the Android appstore: Lightweight characterization of Android applications," *Computing Research Repository CoRR*, vol. abs/1504.06093, no. 26, pp. 1–26, 2015.
[228] L. Villarroel Pérez, "Mining mobile apps reviews to support release planning," Master's thesis, Lenguajes y Sistemas Informáticos e Ingeniería del Software, ETSI_Informatica, Madrid, Spain, 2015.
[229] VirusShare, "Virusshare.com," 2011. [Online]. Available: http://virusshare.com/
[230] Vision Mobile, "Developer Economics 2013: The tools report," 2013. [Online]. Available: http://www.visionmobile.com/product/developer-economics-2013-the-tools-report/
[231] Vision Mobile, "Developer Economics Q1 2015: State of the Developer Nation," 2015. [Online]. Available: http://www.visionmobile.com/product/developer-economics-q1-2015-state-developer-nation/
[232] 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, MIP-1504, 2015.
[233] 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 *Proc. 30th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2015, pp. 749–759.
[234] P. M. Vu, H. V. Pham, T. T. Nguyen, and T. T. Nguyen, "Tool support for analyzing mobile app reviews," in *Proc. 30th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2015, pp. 789–794.
[235] T. J. Walid Maalej, M. Nayebi, and G. Ruhe, "Toward data-driven requirements engineering," *IEEE Softw.* vol. 33, no. 1, pp. 48–54, Jan./Feb. 2016.
[236] M. Wan, Y. Jin, D. Li, and W. G. Halfond, "Detecting display energy hotspots in Android apps," in *Proc. IEEE 8th Int. Conf. Softw. Testing Verification Validation*, 2015, pp. 1–10.
[237] H. Wang, Y. Guo, Z. Ma, and X. Chen, "WuKong: A scalable and accurate two-phase approach to Android app clone detection," in *Proc. Int. Symp. Softw. Testing Anal.*, 2015, pp. 71–82.
[238] H. Wang, J. Hong, and Y. Guo, "Using text mining to infer the purpose of permission use in mobile apps," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, 2015, pp. 1107–1118.
[239] 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 Trans. Inf. Forensics Secur.*, vol. 9, no. 11, pp. 1869–1882, Nov. 2014.
[240] Y. Wang, J. Zheng, C. Sun, and S. Mukkamala, "Quantitative security risk assessment of Android permissions and applications," in *Proc. 27th Int. Conf. Data Appl. Secur. Privacy XXVII*, 2013, pp. 226–241.
[241] M. Wano and J. Iio, "Relationship between reviews at app store and the categories for software," in *Proc. 17th Int. Conf. Netw.-Based Inf. Syst.*, 2014, pp. 580–583.
[242] 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 *Proc. 11th Symp. Usable Privacy Secur.*, 2015, pp. 241–255.
[243] S. Wenxuan and Y. Airu, "Interoperability-enriched app recommendation," in *Proc. IEEE Int. Conf. Data Mining Workshop*, 2014, pp. 1242–1245.
[244] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proc. 18th Int. Conf. Eval. Assessment Softw. Eng.*, 2014, pp. 38:1–38:10.
[245] Z. Xie and S. Zhu, "AppWatcher: Unveiling the underground market of trading mobile app reviews," in *Proc. 8th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2015, pp. 10:1–10:11.
[246] W. Xu, F. Zhang, and S. Zhu, "Permlyzer: Analyzing permission usage in Android applications," in *Proc. 24th IEEE Int. Symp. Softw. Rel. Eng.*, 2013, pp. 400–410.
[247] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, "AppContext: Differentiating malicious and benign mobile app behaviors using context," in *Proc. 37th Int. Conf. Softw. Eng.*, 2015, vol. 1, pp. 303–313.
[248] Y. Yang, J. Stella Sun, and M. W. Berry, "APPIC: Finding the hidden scene behind description files for Android apps," Dept. Electrical Eng. Comput. Sci., Univ. Tennessee, Knoxville, TN, USA, Tech. Rep. id:lda, 2014.
[249] P. Yin, P. Luo, W.-C. Lee, and M. Wang, "App recommendation: A contest between satisfaction and temptation," in *Proc. 6th ACM Int. Conf. Web Search Data Mining*, 2013, pp. 395–404.
[250] F. Zhang, H. Huang, S. Zhu, D. Wu, and P. Liu, "Viewdroid: Towards obfuscation-resilient mobile application repackaging detection," in *Proc. 7th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2014, pp. 25–36.
[251] M. Zhang, Y. Duan, Q. Feng, and H. Yin, "Towards automatic generation of security-centric descriptions for Android apps," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 518–529.
[252] 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 *Proc. 28th Annu. ACM Symp. Appl. Comput.*, 2013, pp. 499–504.
[253] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and evolution," in *Proc. IEEE Symp. Secur. Privacy*, 2012, pp. 95–109.
[254] Y. Zhou, L. Wu, Z. Wang, and X. Jiang, "Harvesting developer credentials in Android apps," in *Proc. 8th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2015, pp. 23:1–23:12.
[255] H. Zhu, C. Liu, Y. Ge, H. Xiong, and E. Chen, "Popularity modeling for mobile apps: A sequential approach," *IEEE Trans. Cybern.*, vol. 45, no. 7, pp. 1303–1314, Jul. 2014.
[256] H. Zhu, H. Cao, E. Chen, H. Xiong, and J. Tian, "Exploiting enriched contextual information for mobile app classification," in *Proc. 21st ACM Int. Conf. Inf. Knowl. Manage.*, 2012, pp. 1617–1621.
[257] H. Zhu, E. Chen, H. Xiong, H. Cao, and J. Tian, "Mobile app classification with enriched contextual information," *IEEE Trans. Mobile Comput.*, vol. 13, no. 7, pp. 1550–1563, Jul. 2014.
[258] H. Zhu, E. Chen, H. Xiong, K. Yu, H. Cao, and J. Tian, "Mining mobile user preferences for personalized context-aware recommendation," *ACM Trans. Intell. Syst. Technol.*, vol. 5, no. 4, 2015, Art. no. 58.
[259] H. Zhu, H. Xiong, Y. Ge, and E. Chen, "Ranking fraud detection for mobile apps: A holistic view," in *Proc. 22nd ACM Int. Conf. Inf. Knowl. Manage.*, 2013, pp. 619–628.
[260] H. Zhu, H. Xiong, Y. Ge, and E. Chen, "Mobile app recommendations with security and privacy awareness," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2014, pp. 951–960.
[261] H. Zhu, H. Xiong, Y. Ge, and E. Chen, "Discovery of ranking fraud for mobile apps," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 1, pp. 74–87, Jan. 2015.
[262] J. Zhu, Z. Guan, Y. Yang, L. Yu, H. Sun, and Z. Chen, "Permission-based abnormal application detection for Android," in *Proc. 14th Int. Conf. Inf. Commun. Secur.*, 2012, pp. 228–239.

William Martin is working toward the PhD degree in computer science at University College London, where he is part of the CREST centre and the UCLappA research group. He is supervised by Mark Harman, Yue Jia and Federica Sarro. His research interests include temporal and causal analyses, machine learning and of course app store analysis. He is a member of Appredict, an app store analytics company that has spun out from UCLappA.

Federica Sarro is a senior research associate in the CREST centre, Department of Computer Science, University College London. She is currently member of the UCLs App Store Analysis Group (UCLappA) and of Appredict, an app store analytics company spun out from the UCLappA group. Her main research areas include empirical and search based software engineering, with a particular interest in predictive modelling for cost and quality estimations, and app stores mining and analysis. She has published more than 50 papers and served as program committee member of more than 40 international events in her area of expertise; in 2015 she has been elected in the steering committee of SSBSE. She has also been program co-chair of SSBSE 2016 and GECCO 2017, co-organiser of WAMA 2016, and track/publicity chair for other international events.



Yue Jia is a lecturer in the Department of Computer Science, University College London. He is currently leading the App Store Analysis Group (UCLappA), which analyses mobile app store ecosystems to understand their key mechanisms. His research interests cover app store analysis, software testing and search-based software engineering. He is also co-director of Appredict, an app store analytics company, spun out from UCL's UCLappA group, and director of MaJiCKe, an automated test data generation start up.

Yuanyuan Zhang received the PhD degree in software engineering from Kings College London, in 2010. She is currently a principal research associate in the CREST centre, University College London. Her research interests include search-based requirements optimisation, app store mining and analysis and evolutionary computation. She has published more than 20 papers including the *IEEE Transactions on Software Engineering*, the *Religious Education,* and the *Religious Education* journal. She is the co-author of several invited keynote papers at leading international conferences, including SPLC 2014 and ICST 2015. She has served on program committees including REFSQ, GECCO, SSBSE, AIRE, MOBS, RELENG, RET, review committee for RE and as the program co-chair for SSBSE 2013 and been elected onto the steering committee (2013-2016) for SSBSE.

Mark Harman is professor of software engineering in the Department of Computer Science, University College London, where he directs the CREST centre and is head of software systems engineering. He is known for work on source code analysis, software testing, app store analysis and Search Based Software Engineering (SBSE), a field he co-founded and which has grown rapidly to include more than 1,600 authors spread more than 40 countries. His work has been used by many organisations including Daimler, Ericsson, Google, Huawei, Microsoft and Visa. He is co-director of Appredict, an app store analytics company, spun out from UCL's UCLappA group, and chief scientific advisor to MaJiCKe, and automated test data generation start up.