UCL DEPARTMENT OF COMPUTER SCIENCE

Research Note
RN/16/02

A Survey of App Store Analysis for Software Engineering

January 29, 2016

William Martin

Federica Sarro

Yue Jia

Yuanyuan Zhang

Mark Harman

Abstract

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

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. Dediu stated that there were more than 200 million Android devices as of April 2012 [54]. 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 [148]. 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 key differences: availability, compatibility, ease of use, variety, and user-submitted content.

It is the user-submitted content which really distinguishes app stores from the ad-hoc commercially available applications that existed beforehand. 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 globally 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 an initial survey into literature that performs "App Store Analysis for Software Engineering" between 2000 and November 27, 2015. 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, 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 [91, 149, 195]. 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 box can be considered non-technical in the

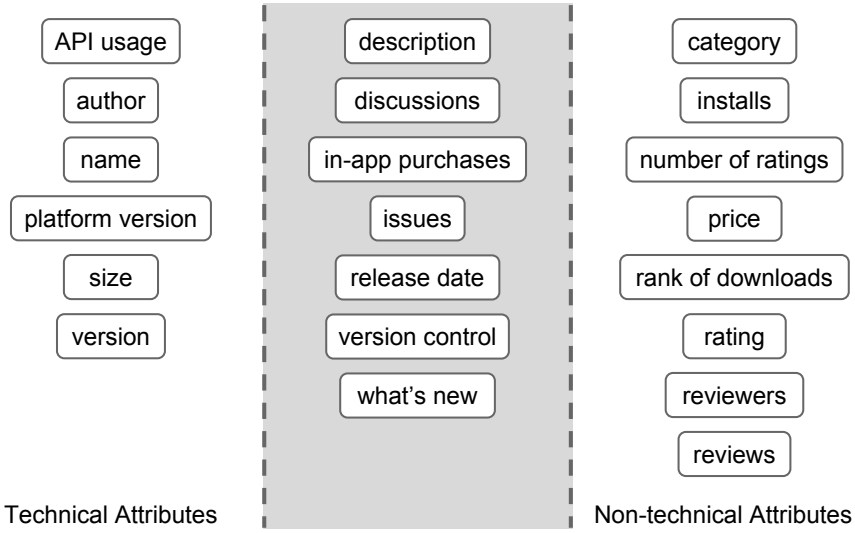

Figure 1: **Example attributes** showing mined attributes which are strictly technical (left) or non-technical (right), and attributes which may be in either category (centre in box).

strictest sense of the definition; they cannot be guaranteed to be obtainable solely from the software in all cases. That is, those in the grey box can be non-technical, but can also be technical. 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.

1.2 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 or 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.

1.3 Search Methodology

In order to collect all relevant literature to date that meets the scope defined in Section 1.2, 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 1.2.
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. We make many of the papers discussed in this survey available in an online repository [194].

1.3.1 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.

1.3.2 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" AND mining AND API
"app store analysis" AND mining AND feature
"app store analysis" AND mining AND release
"app store analysis" AND mining AND requirements
"app store analysis" AND mining AND reviews
"app store analysis" AND mining AND security
"app store analysis" AND mining AND ecosystem

We performed the following more general searches to ensure that no relevant literature was missed from the survey:

"app store analysis"
"app store" AND analysis

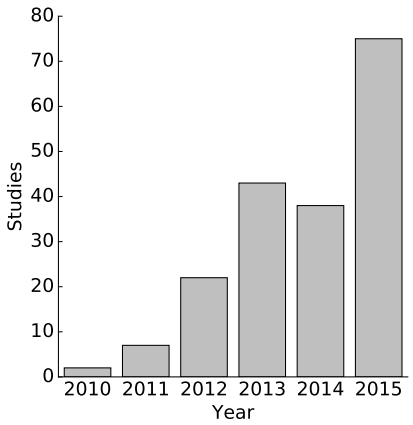

Figure 2: **Histogram of App Store Analysis literature** showing the period from 2010 to November 27, 2015.

"app store" AND mining
"app store" AND mine

1.4 Snowballing

In addition to the repository searches specified in Section 1.3, we also perform snowballing [235] on many of the included studies. This is where 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.

1.5 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.

1.6 Overview

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

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. The top boxes indicate queries run only in Google Scholar, and the lower boxes indicate the queries run to multiple paper repositories. In the case of Google Scholar, only the top 1,000 results were accessible to inspect at the time of search.

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

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

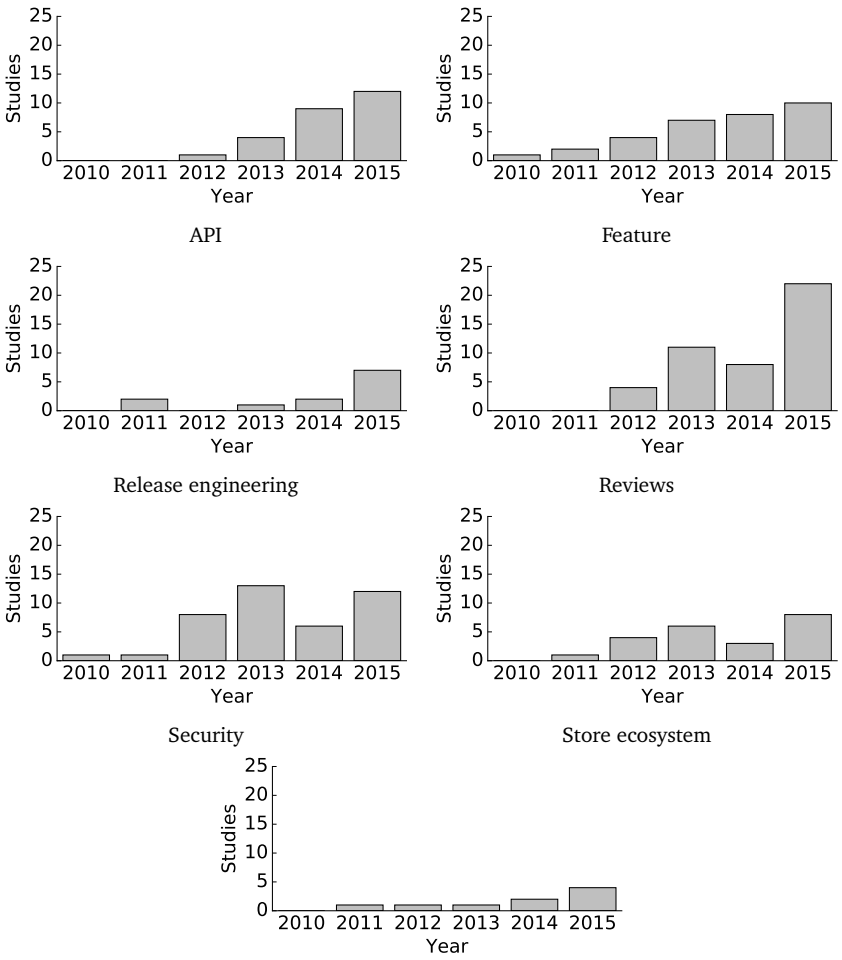

Figure 3: **Histogram of sub-field trends** showing the period from 2010 to November 27, 2015.

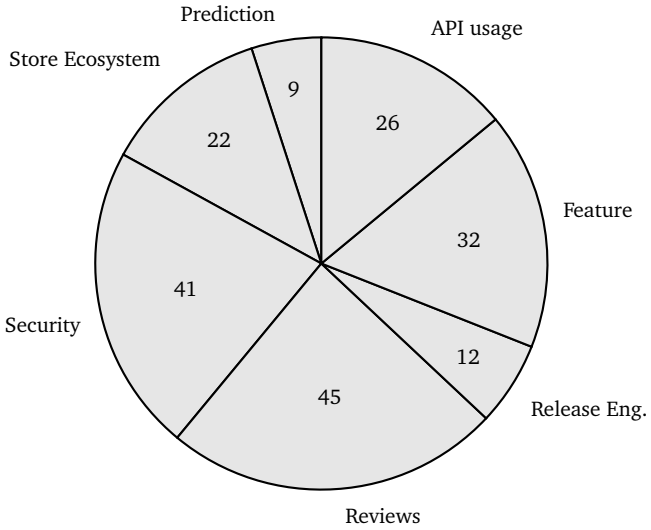

Figure 4: **Pie chart showing overall sub-field distribution** showing the period from 2010 to November 27, 2015.

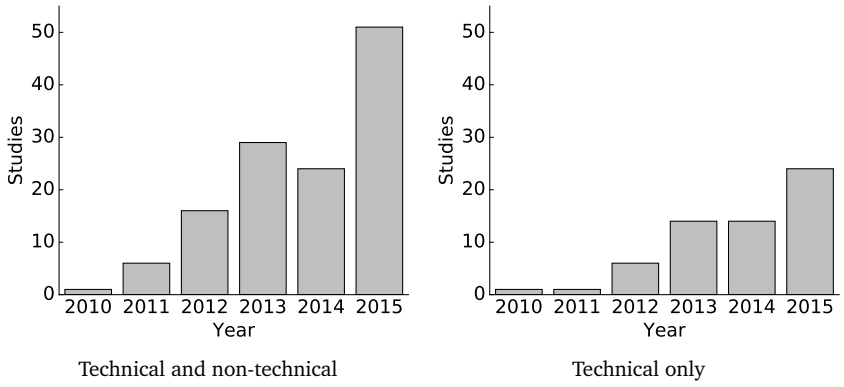

Figure 5: **Histogram showing number of research papers incorporating non-technical information and technical-only research papers** showing the period from 2010 to November 27, 2015.

2 Non-Technical Research

It is important to note that while software engineering deals primarily with code, 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%) 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. 5 shows that the number of studies incorporating non-

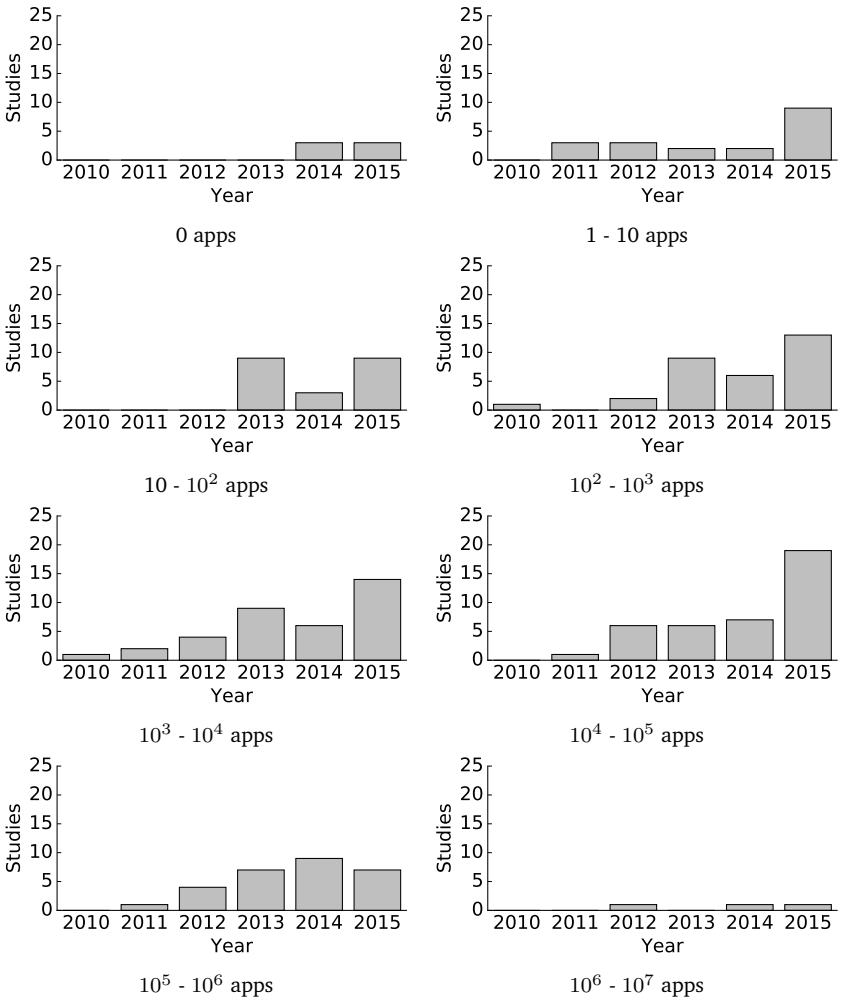

Figure 6: **Histograms showing number of research papers grouped into app quantity ranges** each year, showing the period from 2010 to November 27, 2015. Each histogram depicts a range such as 10^2 - 10^3 apps, which means that the studies included used between 10^2 and 10^3 apps.

technical information is growing year-on-year. This graph also shows that in 2013 there was not only a large spike in App Store Analysis research, but that much of this growth came from technical-only papers mining data from app stores to support related research, treating them as a type of software repository. We can see from Fig. 2 that even including the boom in technical-only research, there is a linear growth trend to App Store Analysis research ($R^2 = 0.90$).

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	6	$[10^3, 10^4)$	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

3 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 3,000 apps, but the other half study a quantity of apps several orders of magnitude higher. This is reflected in Fig. 6, 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 2015 is not a complete year, and 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.

4 Key Ideas Timeline

A timeline depicting the key ideas is shown in Fig. 7. The first App Store Analysis literature emerged in 2010, with the study by Shabtai et al. [200] that extracted feature information for training a classifier to distinguish tools and games. The authors proposed that the approach could later be used for detecting malware. This was the first app store feature analysis paper. In the same year, Bläsing et al. [26] used the Android Market as a form of software repository by downloading a set of apps to test their static and dynamic analysis tool. This was the first app store security analysis paper.

Lee et al. [123] published the earliest work that incorporated technical with non-technical information in 2011 by analysing developer sales strategies in the Apple App Store. In the same month, Henze and Boll [94] studied release times in the Apple App Store, finding that Sunday evening is the best time for deploying games. These were the first app store release engineering papers. Syer et al. [205] compared feature equivalent apps from Android and Blackberry. This

2008 Apple App Store launched

Google Play launched (as Android Market)

2009 Blackberry World App Store launched

2010 [200] Shabtai et al. extracted feature information to differentiate between Tools and Games categories (**Feature**)

[26] Bläsing et al. used the Android Market as a software repository for testing their APK analyser (**Security**)

Windows Phone Store launched

2011 [205] Syer et al. compared Blackberry and Android platforms using feature equivalent apps (**Store ecosystem**)

[123] Lee et al. analysed deployment strategies for optimising download rank (**Release engineering**)

[94] Henze and Boll studied release times in the Apple App Store for game deployment (**Release engineering**)

[199] Sethumadhavan discussed function point analysis for mobile apps (**Size and Effort Prediction**)

2012 [79] Goul et al. analysed reviews in order to facilitate requirements engineering (**Reviews**)

[36] Chandy and Gu mined 6,319,661 reviews from the Apple App Store for spam classification (**Reviews**)

[91] Harman et al. connected non-technical, technical & business aspects; extracted technical features from text descriptions (**Feature**)

[188] Ruiz et al. studied class reuse and inheritance in Google Play (**API**)

[247] Zhu et al. studied the problem of mobile app classification (**Feature**)

2013 [221] Vision Mobile found 72% of devs dedicated to Android; iOS and Android devs earn 2x other platform devs

[130] Lim and Bentley simulated an app store ecosystem (**Store ecosystem**)

[102] Iacob and Harrison automatically analysed app reviews to identify feature requests and bug reports (**Reviews**)

[249] Zhu et al. studied ranking fraud in App stores (**Security**)

2014 [187] Ruiz et al. investigated the effect of ad libraries on app ratings (**API**)

[78] Gorla et al. performed malware detection through API usage/description cluster outliers (**Feature**)

[222] Vision Mobile paid and with-ads models almost tied in revenue and developer share

2015 [152] McIlroy et al. studied update frequency of Google Play apps (**Release engineering**)

[149] Martin et al. identified the "App Sampling Problem" (**Reviews**)

[146] Malavolta et al. investigated hybrid mobile apps from technical and user perspectives (**Reviews**)

[173] Park et al. studied the mobile app retrieval problem (**Reviews**)

[195] Sarro et al. studied feature migration between apps (**Feature**)

[116] Khalid et al. surveyed app store review literature (**Reviews**)

[63] Ferruci et al. compared approaches for size and effort prediction in mobile apps (**Size and Effort Prediction**)

Figure 7: **Key ideas timeline** for App Store Analysis literature.

was the first app store ecosystem analysis paper. In the first study on size and effort prediction for mobile apps, Sethumadhavan [199] discussed the application of FPA (Function Point Analysis) to Android applications.

In 2012 Goul et al. [79] analysed sentiment in 5,000 reviews in order to facilitate requirements engineering. Then in a larger study Chandy and Gu [36] mined 6,319,661 reviews from the Apple App Store for spam classification. These were the first app store review analysis papers. Harman et al. [91] identified App Store Analysis as a "Mining Software Repositories" problem, and performed analysis on the trends between technical, non-technical and business attributes extracted from the Blackberry store. Later in 2012, Ruiz et al. published the first study on app store API usage, by studying class reuse and inheritance in Google Play. Zhu et al. [247, 248] published the first study on the problem of mobile app classification, using features extracted from the web and device logs.

In 2013 Lim and Bentley performed a simulation of the app store ecosystem [130], and Iacob and Harrison produced a system for automatically extracting feature requests and bug reports from app reviews [102]. In this year Vision Mobile identified a clear lead in the number of developers dedicated to Android, over other platforms [221]. Zhu et al. published the first study on problem of detecting ranking fraud in app stores [249].

In 2014 Ruiz et al. identified a link between ad libraries on app ratings [187], and Gorla et al. performed malware detection by clustering app descriptions and identifying API usage outliers [78]. With-ads revenue models have seen increased usage since the inception of apps stores, and at the end of 2014 had almost tied paid apps for revenue and developer share [222].

In 2015 McIlroy et al. performed the first study on app update times in Google Play [152], and Martin et al. identified the "App Sampling Problem" that affects many App Store Analysis papers [149]. In addition, Malavolta et al. analysed hybrid apps [93] from technical and user perspectives [146], and Park et al. utilised user reviews to improve mobile app retrieval [173]. Later in 2015, Sarro et al. performed the first study on feature migration between apps [195], and Khalid et al. published the first survey of app store review literature [116]. In an empirical study, Ferrucci et al. [63] compared the approaches for size and effort prediction of mobile apps.

5 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. This is 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 this has not been possible for the Apple platform iOS, since the store was launched in 2008, but this is likely due to the complexity and security inherent in iOS binaries; 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) copy-

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. [188], 2012	ICPC	4,323
Linares-Vásquez et al. [136], 2013	FSE	7,097
Shirazi et al. [189], 2013	EICS	400
Minelli and Lanza [156], 2013	ICSM	20
Minelli and Lanza [157], 2013	CSMR	20
Ruiz et al. [187], 2014	IEEE Soft.	236,245
Hao et al. [90], 2014	MobiSys	3,600
Dering and McDaniel [56], 2014	MILCOM	450,000
Linares-Vásquez et al. [138], 2014	MSR	24,379
Ruiz et al. [186], 2014	IEEE Soft.	208,601
Linares-Vásquez [135], 2014	ICSE comp.	0
Viennot et al. [217], 2014	SIGMETRICS	1,107,476
Bartel et al. [17], 2014	IEEE Soft. Eng.	1,421
Zhang et al. [241], 2014	WiSec	10,311
Borges and Valente [29], 2015	PeerJ C. S.	396
Bavota et al. [20], 2015	IEEE Soft. Eng.	5,848
Kim et al. [119], 2015	ASE	350
Khalid et al. [112], 2015	IEEE Soft.	10,000
Watanabe et al. [233], 2015	SOUPS	200,000
Zhou et al. [245], 2015	WiSec	36,561
Wan et al. [227], 2015	ICST	398
Wang et al. [228], 2015	ISSTA	105,299
Syer et al. [206], 2015	Soft. Qual.	5
Azad [14], 2015	Masters thesis	950
Wang et al. [229], 2015	UbiComp	7,923
Seneviratne et al. [197], 2015	WiSec	4,114
	Mean	93,298
	Median	5,086

right on binaries and b) many iOS apps are paid-for. 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. 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.

In 2012 Ruiz et al. [188] studied class reuse and inheritance in 4,323 Android apps mined from 5 categories in Google Play. 217 apps were found with 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. [186] in 2014. More evidence of substantial code reuse was found, and the authors concluded that the apps benefit from increased productivity but risk dependence on the quality of the code they reuse. Borges and Valente [29] used association rule mining to infer API usage patterns, using a dataset of 396 open source Android apps. For their study, the authors extended `APIMiner` [161] 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.

Linares-Vásquez et al. analysed the effect of fault and change-prone core Google APIs on app ratings [136]. 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. The authors found that the more fault-prone and change-prone APIs we used frequently by less well-rated apps, and that the more popular (successful) apps used APIs that were less susceptible to API 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. [136], Linares-Vásquez also presented an approach for a recommendation system for Android app developers [135], to help them to prepare for platform updates and avoid breaking changes and bugs. The authors extended their API analysis work to identify APIs that have a high energy usage [137], but this study did not combine non-technical app store information.

In 2013 Minelli and Lanza presented a visual analytics web tool for studying repositories of apps [157, 156]. 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 small code-bases. Additionally the authors found little use of inheritance in Android apps, and much duplication. Shirazi et al. [189] 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. Peiravian and Xingquan [174] used API calls to augment their malware classifier training.

Hao et al. [90] 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 [56] 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. [28], 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 ad libraries on app ratings [187]. They combined non-technical rating information with the extracted technical information showing ad library usage to perform the study. Ad libraries query their host server at regular intervals to fetch advertisements for display, and this interval determines the "ad 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 ad libraries. However, certain APIs were found to have low median ratings from apps that used them. The authors assessed that this is due to intrusive behaviours, such as recording entered passwords. Bavota et al. [20] later 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 the 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.

Linares-Vásquez et al. [138] decompiled and analysed 24,379 APKs from Google Play and found that the 82% of detected clones replicate 3rd party libraries. Gorla et al. [78] trained a one-class support vector machine [147] on API usage information in order to identify outliers in trained clusters for security purposes. Viennot et al. [217] 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 approximately a quarter of free apps are duplicated content. They found that paid apps account for just 0.05% of downloads, and the top 10% of most popular apps account for 96% of total downloads as of June 23, 2013. Bartel et al. [17] 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.

Zhang et al. [241] 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. [119] scan API invocations to identifying plagiarised applications, in a more sophisticated approach than similarity detectors that scan code, as it handles code obfuscation. Wang et al. [228] 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.

In 2015, Khalid et al. [112] performed static analysis on 10,000 free Google Play apps, and found that 3 categories of `FindBugs` warnings occur more frequently in lower rated apps. The categories 'bad practice', 'internationalisation' and 'performance' had more warnings in lower-rated apps, suggesting that these areas are the ones developers should focus on to achieve better rating performance. Watanabe et al. [233] 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. [245] 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.

Wan et al. [227] 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. Syer et al. [206] 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. Azad [14] 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. [211] 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. Wang et al. [229] 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. [197] studied 275 free and 234 paid Android apps, and found that paid apps collect personal information, in the same way as free apps do. 60% of the paid apps collected personal information, compared to 85% in free apps. The authors subsequently showed that 20% of 3,605 collected Android apps were connected to more than three trackers.

6 Feature Analysis

Papers that extract feature information from either technical or non-technical sources of information are discussed in this subsection, and are summarised in Table 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.

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. [200], 2010	g	CIS	2,285
Chen and Liu [38], 2011	a	iConference	102,337
Coulton & Bamford [47], '11	n	MobileHCI	3
Harman et al. [91], 2012	b	MSR	32,108
Sanz et al. [190], 2012	g	CCNC	820
Teufl et al. [210], 2012	g	MobiSec	130,211
Zhu et al. [247], 2012	n	CIKM	680
Mokarizadeh et al. [160],'13	g	WEBIST	21,065
Teufl et al. [209], 2013	g	Sec. & Com. Netw.	443
Lulu and Kuflik [21], 2013	g	IUI	120
Bhattacharya et al. [24],'13	g	CSMR	24
Yin et al. [240], 2013	a	WSDM	5,661
Lin et al. [133], 2013	a	SIGIR	7,116
Ihm et al. [105], 2013	g	CGC	10
Kim et al. [120], 2014	a	Service Business	100,830
Finkelstein et al. [64], 2014	b	Tech. report	42,092
Yang et al. [239], 2014	g	Tech report	26,703
Zhu et al. [248], 2014	n	TMC	680
Zhu et al. [250], 2014	g	KDD	170,753
Jiang et al. [107], 2014	g	INTERNETWARE	150
Zhu et al. [246], 2014	a	IEEE Cybernetics	15,045
Vakulenko et al. [213], 2014	a	ICIS	600,000
Lin et al. [134], 2014	a	SIGIR	6,524
Sarro et al. [195], 2015	b,s	RE	54,983
Berardi et al. [23], 2015	a,g	SAC	5,993
Svedic [204], 2015	a	PhD thesis	60
Seneviratne et al. [198],'15	g	WWW	232,906
Tong et al. [212], 2015	g,w	JCST	10,000
Wang et al. [229], 2015	g	UbiComp	7,923
He et al. [92], 2015	g	Big Data	122,875
Tian et al. [211], 2015	g	ICSME	1,492
Nayebi and Ruhe [166],'15	g	PeerJ C.S.	241
Lulu and Kuflik [22], '15	g	MOB INF SYST	6,633
		Mean	52,084
		Median	6,579

Papers in this section show that it is possible to extract feature information from sources other than source code or requirements lists: features are extracted from app descriptions, API usage, manifest files, decompiled source strings, categories and permissions. Many different methods are used for extraction and categorisation of features, including natural language processing, topic modelling and clustering. The work shows that analysis of multiple apps can be augmented with meaningful technically-oriented information, mined from freely-available app store pages.

Shabtai et al. [200] 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 2011, Coulton and Bamford [47] conducted a case study on games created for the WidSets platform, an earlier app store that targets Nokia phones (including non-smartphones). Their findings were transferable to modern app stores: high download numbers are required in order to gain active users, and popular features such as chat can increase popularity and the proportion of active users. Chen and Liu [38] 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.

In 2012 Sanz et al. [190] 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 [66].

Harman et al. [91] introduced app store mining as an MSR (Mining Software Repositories) problem. They mined app information and performed correlation analysis on price, downloads, and rating. Correlations were computed 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 [64], 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.

Teufl et al. [210] 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 [209] to propose a first-step malware detection method using links between description terms and security permissions to identify suspicious outliers. Zhu et al. [247, 248] 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. [23] 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 [184] 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.

In 2013 Mokarizadeh et al. [160] performed clustering on 21,065 apps, mined from Google Play, after applying topic modelling on app descriptions. They found that the resultant clustering was very different from the app's assigned categories, and apps in the same category often had dissimilar description topic distributions. Mokarizadeh et al. also performed correlation analysis and found unsurprisingly that users download free apps more frequently, and that downloads correlate with the number of ratings an app has received. Lulu and Kuflik [21] performed clustering on 120 apps mined from Google Play, comparing description-based with category-based clustering. They found that descriptions provide good clustering features, and present the method as the basis of an app recommendation system. The authors later built on this work [22], by extracting features from 6,633 app descriptions and enriching them with information on 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 [73].

Bhattacharya et al. [24] 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. Importantly, 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.

Yin et al. [240] proposed the Actual Tempting (AT) model to perform app recommendation for users. The model uses latent tempting parameters, and uses information such as the number of users who own an app "a" and later download app "b", and who do not download "b" after owning "a". The model also uses feature overlap information, measured by performing topic modelling on app descriptions and computing the topic overlap between each pair. The authors find that the AT approach outperforms collaborative filtering and case-based reasoning in their initial experiments. Lin et al. [133] used topic modelling on the Twitter messages of users that follow an app's Twitter feed, in order to generate latent groups related to the app. The groups are then used as part of a recommendation system, in order to help remove the problem of cold start in app recommendation based on other metadata. The system was tested on 7,116 apps mined from Apple App Store and the authors found that it outperformed recommendation using app descriptions. However, in 2014 Lin et al. [134] used topic modelling on app descriptions in order to produce a recommendation system. The model is semi-supervised and incorporates app version information using different weights corresponding to update types: so that newer app versions can be recommended when they add a certain feature to the description.

Resultant topics are weighted based on their category in the app store to provide a recommendation. The model was trained on 6,524 apps mined from the Apple App Store. Ihm et al. [105] 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.

Yang et al.[239] 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 the app's permissions file to establish whether the description makes correct claims about its functionality, and whether it resides in the correct 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 their method achieved (average) 88.1% category accuracy, and 76.5% permissions accuracy. Kim et al. [120] 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 recategorised them using the resulting clusters. Zhu et al. [246] mined the daily top 300 free and top 300 paid apps from Apple App Store charts from February 2, 2010 to September 17, 2012, collecting information on 15,045 apps in total. They used popularity information to construct a Popularity-based Hidden Markov Model (PHMM), to encode trend and other latent factors. The authors state that this can be used in a variety of ways, including app recommendation, review spam detection, and demonstrate its usefulness in ranking fraud detection.

Jiang et al. [107] conducted a user survey on 50 app descriptions in order to identify the attributes most important to the quality of the description. A support vector machine was trained on the resultant attributes and tested on a sample of 100 descriptions, finding an accuracy of 0.62. The findings showed that quick overviews are the most effective form of app description, and the study contains further heuristics on good description styles. Jinh et al. [108] 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. Zhu et al. [250] 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. [213] performed topic modelling on a set of 600,000 app descriptions mined from Apple App Store. They use the resultant topics to suggest categories, and to improve and augment existing categorisation approaches used in app stores.

In a longitudinal study on 60 paid iOS apps, Svedic [204] 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 as a result the app sales increase. Watanabe et al. [233] found that apps often contain secondary functionality that is not mentioned in their descriptions. In a study of 232,906 apps, Seneviratne et al. [198] 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.

Sarro et al. [195] 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 [91], and used the proposed theory to empirically analyse the migratory and non-migratory behaviours of 4,053 non-free features from Samsung and Blackberry stores. The results revealed that features generally migrate to a category that has similar characteristics, however there are also a few features that migrate to apparently non-related categories. The early identification of these features may allow developers to find undiscovered requirements. The authors also found that approximately one third of features were intransitive (they neither migrate nor do they die out over the period studied), and such features exhibited significantly different behaviours with regard to important properties, such as their price. Being aware of which are the intransitive features in a given category may support developers in identifying crucial ('must-have') requirements for their apps.

Tong et al. [212] 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 [155]. Nevertheless, the study shows the importance of accurate feature discovery and representation, and can help lead to future studies using extracted features. Wang et al. [229] extracted features from decompiled `Java` code, from their collection of 7,923 apps mined from Google Play. They used the extracted features to train classifiers for predicting how 'location' and 'contact' information is used, with 85% and 94% accuracy, respectively. He et al. [92] trained a system for targeting users for advertising, with a dataset containing app install data on a per-user basis, consisting of 122,875 apps from the Huawei App Store. The authors reported a higher click rate than current targeting approaches.

Tian et al. [211] 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 are updated more frequently and use more modern features of the Android operating system. Nayebi and Ruhe [166] extracted feature information from 241 Google Play apps, and used crowd-sourcing to assign user value to each of the features. The authors use the approach for service portfolio planning [2].

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 [123], 2011	a	AMCIS	3,168
Henze and Boll [94], 2011	a	MobileHCI	24,647
Datta and Kajanan [52], 2013	a	CloudCom-Asia	3,535
Lee and Ragu [124], 2014	a	JMIS	7,579
Ruiz et al. [159], 2014	g	IEEE Soft.	120,981
Guerrouj et al. [83], 2015	g	SANER	154
Comino et al. [43], 2015	a,g	Tech report	1,000
McIlroy et al. [152], 2015	g	ESE	10,713
Gui et al. [84], 2015	g	ICSE	21
Carbunar and Potharaju [31],'15	g	ASONAM	160,000
Alharbi and Yeh [5], 2015	g	MobileHCI	24,436
Martin et al. [150], 2015	g,w	Tech report	1,033
		Mean	29,772
		Median	5,557

7 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 5 prior to November 27, 2015. Release studies typically require time series data, in order that the changes made to apps in their releases can be recorded. Due to the recent 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 are no release studies in Blackberry or Windows Phone Store. The scale of the past studies in this section is relatively small, ranging from 21 to 24,647; this scale is not surprising, given the difficulty of mining longitudinal data for a large number of data points.

Lee et al. [123] 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, and found a positive relationship between download rank and app portfolio diversification. The study incorporates technical (download rank) with the non-technical information (category, price) in order to identify actionable findings for app developers. In 2011 Henze and Boll [94] analysed release times and user activity in the Apple App Store, and concluded that Sunday evening is the best time for deploying games. Their study also found that version updates were an effective strategy for raising an app's rank in the store.

In 2012 Moller et al. [164] studied the installation behaviour of users with recently updated apps, in a security related study. In 2013 Datta and Ka-

janan [52] studied review counts from the Apple App Store, and found that apps receive more reviews after deploying updates on Thursdays. In 2014 Lin et al. [134] incorporated version information in their app recommendation system, in order to ensure that apps are recommended if they add new features to new versions. Lee and Raghu [124] 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.

The 2014 study by Ruiz et al. investigated the updates made to update ad libraries [159]. They found that over 12 months, almost half of the 5,937 apps with multiple updates had an ad library update. Approximately 14% of ad updates contained no changes to the app's code, indicating the effort involved in keeping ad libraries updated. In 2015 there was a spike in studies focused on app release engineering. Gui et al. found, from 21 apps in Google Play with frequent releases, that 23% of their releases contained ad-related changes [84].

Comino et al. [43] studied the top 1,000 apps in Apple App Store and Google Play. They found that for iTunes, increased numbers of app releases are more likely when the app is performing badly, and releases can boost downloads. Neither finding held true for Google Play, however. Very recently, McIlroy et al. [152] studied update frequencies in the Google Play store, after mining data about 10,713 mobile apps. They found that only 1% of the studied apps received more than one update per week, and only 14% were updated in a two-week period. McIlroy et al. also found that rating was not affected by update frequency. However, the findings of Guerrouj et al. [83] indicate that high code churn in releases correlates with lower ratings. Carbunar and Potharaju [31] conducted a longitudinal study on 160,000 Google Play apps mined daily over a 6 month time period in 2012. They found that at most 50% of apps were updated in each category, and that there is an issue of "stale apps" affecting aggregated statistics on large populations. The authors also found that a few developers dominated the total download counts, that productive developers did not have many popular apps, and that there was no correlation between price and downloads.

Alharbi and Yeh [5] tracked the design patterns used by 24,436 Android apps over a period of 18 months. They found that depreciated patterns are sometimes adopted after they are depreciated, and that new pattern adoption rates are low. By tracking the app descriptions, they found that authors sometimes update the app descriptions to reflect changes to their design patterns. They believe that this shows that descriptions are used as a communication channel between developers and users. The authors report on apps that start and stop using certain design patterns. An interesting future research direction might be to record the migration of these "design features" using the app feature migration terminology of Sarro et al. [195]. Nayebi and Ruhe [166] combined app features with values gained from crowd-sourcing as an approach to app service portfolio planning. Martin et al. [150] conducted a longitudinal study on 1,033 apps mined from Google Play and Windows Phone Store in a 12 month time period. The authors used causal inference to identify the releases with most impact in ratings and downloads. They found that release text discussing features and not bug fixes may lead to more impactful releases, and releases

that improve rating.

8 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 focus 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 has been gaining momentum: 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. One avenue of research that has not been attempted is the study of reviews in the Windows Phone Store, which was launched in 2010 but has not achieved the widespread success of Google Play and Apple App Store, in the competitive market.

In 2012 Goul et al. [79] 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. In a larger study Chandy and Gu [36] 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.

Later in 2012, Hoon et al. [99] and Vasa et al. [215] 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 [98]. They found that the majority of mobile apps reviews are short in length, and that rating and category influences the length of reviews. The majority of studied apps received under 50 reviews in their first year. Half of the apps analysed decrease in rating quality over time, leading the authors to suggest that user expectations are changing rapidly towards apps, and that developers must keep up with demand to remain competitive.

Another large sample was used in the 2013 study by Fu et al. [67], 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 is a large difference between free and paid apps, and that paid apps have an associated 'complaint' type about price that free apps do not.

In 2013 Ha et al. [88] manually examined 556 reviews mined from 59 Google Play apps, in order to classify them into topics and sub-topics based on content. They found that most information in reviews concerns the quality

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. [99], 2012	a	OzCHI	17,330
Vasa et al. [215], 2012	a	OzCHI	17,330
Chandy and Gu [36], 2012	a	WebQuality	3,090
Goul et al. [79], 2012	a	HICSS	9
Ha et al. [88], 2013	g	CCNC	59
Oh et al. [168], 2013	g	CHI	24,000
Hoon et al. [98], 2013	a	Tech report	17,330
Iacob and Harrison [102], 2013	g	MSR	270
Galvis Carreño et al. [68]	g	ICSE	3
Khalid [113], 2013	a	ICSE	20
Fu et al. [67], 2013	g	KDD	171,493
Chen et al. [40], 2013	a,g	WWW	5,059
Pagano and Maalej [169], 2013	a	RE	1,100
Hoon et al. [97], 2013	a	OzCHI	25
Iacob et al. [104], 2013	g	BCS-HCI	161
Iacob et al. [103], 2014	g	MobiCASE	270
Khalid [115], 2014	a	IEEE Soft.	20
Chen et al. [39], 2014	g	ICSE	4
Cen et al. [34], 2014	g	PIR	6,938
Guzman and Maalej [87], 2014	a,g	RE	7
Khalid et al. [114], 2014	g	FSE	99
Wano and Iio [232], 2014	a	NBIS	500
Erić et al. [60], 2014	a	QIP	968
Khalid et al. [116], 2015	g	IJITCS	0
Gao et al. [69], 2015	g	SOSE	4
McIlroy et al. [153], 2015	a,g	ESE	12,000
Cen et al. [33], 2015	g	SIAM	12,783
Vu et al. [225], 2015	g	ASE	3
Vu et al. [224], 2015	g	CoRR	95
Malavolta et al. [145], 2015	g	MS	11,917
Malavolta et al. [146], 2015	g	MOBILESoft	11,917
Park et al. [173], 2015	g	SIGIR	43,041
Panichella et al. [172], 2015	a,g	ICSME	7
Palomba et al. [170], 2015	g	ICSME	100
Moran et al. [162], 2015	g	FSE	14
Gomez et al. [77], 2015	g	MOBILESoft	46,644
Martin et al. [149], 2015	b	MSR	15,095
Maalej and Nabil [144], 2015	a,g	RE	1,140
Pérez [219], 2015	g	Masters thesis	4
Khalid et al. [117], 2015	-	IJIEEB	0
Gu and Kim [82], 2015	g	ASE	17
Guzman et al. [85], 2015	a,g	ESEM	7
Guzman et al. [86], 2015	a,g	ASE	7
McIlroy et al. [154], 2015	g	IEEE Soft.	10,713
Liang et al. [126], 2015	a	IJEC	139
		Mean	9,594
		Median	161

of the app, and not security or privacy concerns. Oh et al. [168] developed a review digest system which they tested on 1,711,556 reviews mined from 24,000 Google Play apps. They automatically categorised reviews into bug reports, functional requests and non-functional requests, and produce a digest featuring the most informative reviews in each category.

Iacob and Harrison [102] 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. 161 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% of reviews contain feature requests. Iacob et al. [104] then studied how the price and rating of an app influence the type and amount of user feedback that it receives through reviews. The authors selected 3,279 reviews for the study, from which they identified 9 classes of feedback: positive, negative, comparative, price related, request for requirements, issue reporting, usability, customer support, versioning. From the selected apps, there was a roughly equal split of positive type reviews with feature/issue type reviews, with very few other types such as negative or price related. As an extension to the MARA system, Iacob et al. [103] introduced a set of linguistic rules for identifying feature requests and bug reports in order to help facilitate app development.

Khalid [113, 115] manually categorised 6,390 negative reviews from a sample of 20 free iOS apps, and reported the most frequent causes of complaints. The apps had combined over 250,000 reviews, and so 6,390 reviews is a statistically representative sample at the 95% confidence level. The authors carried out a manual analysis of the 6,390 reviews, finding that 11% of samples concerned complaints about a recent update. Users were most dissatisfied by issues relating to invasion of privacy and unethical behaviour, while hidden cost was the 2nd most negatively perceived complaint.

Chen et al. [40] compared the maturity ratings of 1,464 equivalent apps between the Apple App Store and Google Play, and taking the Apple store ratings as the accurate ratings, the authors found that 9.7% of the Android apps were underrated and 18.1% were overrated. The authors also studied a sample of 729,128 reviews from 5,059 Google Play game apps, and trained a classifier on the sets of app descriptions and user reviews, and iOS maturity ratings, to automatically verify app maturity ratings. Pagano and Maalej [169] gathered a sample of 1.1 million reviews from the Apple App Store in order to provide an empirical summary of user reviewing behaviour. They found that most feedback is provided after releases, that positive feedback is often associated with highly downloaded apps, and that negative feedback is often associated with poorly downloaded apps and often does not contain user experience or contextual information. Khalid et al. [114] studied the devices used to submit app reviews, in order to determine the optimal devices for testing.

Several authors have incorporated sentiment in their study of reviews. Galvis Carreño and Winbladh [68] extracted user requirements from comments using the ASUM model [109], a sentiment-aware topic model. Initial results showed that the method aids requirements summation with significantly less effort than manual identification, but do not return all possible requirements. Hoon et al. [97] gathered a set of 29,182 short reviews of up to 5 words, from the top 25 Health & Fitness apps in the Apple App Store. They analysed the reviews and found they are mostly made up of sentiment words, and match the star rating

of the review closely.

In 2014 Chen et al. [39] produced a system for extracting the most informative reviews, placing weight on negative sentiment reviews. Guzman and Maalej [87] studied user sentiments towards app features from a multi-store sample, which also distinguished differences of user sentiments in Google Play from Apple App Store. Wano and Iio [232] 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. [60] studied the star ratings of 48 million reviews from 968 popular free and paid Apple apps. They found that the reviews were mostly positive, and there were significant differences in the distributions between categories, and also between free and paid. Free apps had more reviews but a lower mean, and higher standard deviation. Due to the higher numbers of reviews for free apps, which give an app credibility, the authors argue that in-app purchasing revenue models are a good way to make money for developers, especially if used as a 'teaser' for a paid version.

Cen et al. [34] devised an approach to identify the Comments with Security / Privacy Issues (CSPI) from a set of mined Google Play app reviews. The authors later built upon this work, using reviews in order to rank the security risk of apps, by detecting security labels in a crowd-sourcing approach [33]. Using `AndroGuard` [7] 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.

In 2015 McIlroy et al. [153] 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. [69] proposed `AR-Tracker`, a similar tool to `AR-Miner` [39], 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. Gomez et al. [77] 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.

Martin et al. [149] 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 which 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. Pérez [219] mined and labelled 160 user reviews from 5 Google Play apps in order to train a review categorisation tool, that identifies feature requests and bug reports. The tool was evaluated on 400 labelled reviews and achieved 0.78 accuracy. Malavolta et al. [145, 146] analysed 3 million reviews from 11,917 Google Play apps, and produced a summary of user perceptions about 445 hybrid apps [93] compared with native apps. The authors found that hybrid mobile apps are receive similar ratings to native apps, but native apps have been reviewed on average 6.5 times more. They plan to replicate the work using multiple stores and a small set of cross-platform apps to compare their percep-

tion across different platforms. Vu et al. [224, 225] developed MARK, a system that identifies keywords in sets of reviews in order to assist summarisation and search. The method is one of several summarisation approaches that are applied to reviews.

Park et al. [173] 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. [172] presented a system for automatically classifying user reviews based on a predetermined taxonomy, in order to support software maintenance and requirements evolution. They verified the system on a manually labelled truth set of 1,421 sentences extracted from reviews, and achieved 0.85 precision and 0.85 recall when training the system on language structure, content and sentiment features. Maalej and Nabil [144] produced a classification method identifying bug reports and feature requests from user reviews. The authors found that upwards of 70% precision and 80% recall can be obtained using multiple binary classifiers, as an alternative to a single multiclass classifier. They also found that the commonly used NLP techniques, stopword removal and lemmatisation, can negatively affect the performance of this classification task.

Khalid et al. [116] partially 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 proposed assigning categories to reviews, as well as the ability to sort and filter reviews based on the assigned category, helpfulness and star rating. They suggest that adding a user reply feature would assist the developers to get the highest quality reviews.

Moran et al. [162] propose the FUSION system, that performs static and dynamic analysis on Android apps, in order to help users complete bug reports. The system focuses on the steps to reproduce a bug, using dynamic analysis to walk through Android system events. A separate set of authors, Khalid et al. [117], argue that app store reviews can be used for crowdsourcing. They argue that users are inadvertently performing crowdsourcing when they review apps, solving the following problems: requests for potential features, suggestions for developer action, recommendations for other users, and issue reporting.

Palomba et al. [170] study the Google Play reviews from 100 open source Android apps, and link the reviews to code changes. They find that a mean of 49% of reviews are implemented in new releases, and that the apps with changes more directly implementing the content of user reviews improve their ratings with new releases. Gu and Kim [82] proposed SUR-Miner, a review summarisation and categorisation tool, which they evaluated on 2,000 sentences from reviews of 17 Google Play apps. The tool is intended for use by developers, and produces a visualisation of the reviews. The authors surveys the developers of the studied apps, of whom 28 out of 32 agreed that the tool is useful. In the Google Play store it is possible for developers to respond to reviews, which can lead to users changing their rating.

Guzman et al. [85] developed a tool called DIVERSE, which 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 [87].

Guzman et al. [86] also developed an ensemble of machine learning classifiers in order to classify user reviews. They tested this system on 4,550 reviews mined from 7 apps in the Google and Apple app stores, and achieved a precision of 0.74 and recall of 0.59 on a manually labelled set of 1,820 reviews.

McIlroy et al. [154] studied responses to reviews from 10,713 Google Play apps, finding that most developers do not respond to reviews. However in the cases where a response occurred, 38.7% of users were found to subsequently change their ratings, resulting in a median increase in individual user ratings of 20%. Liang et al. [126] performed MultiFacet Sentiment Analysis (MFSA) on user reviews from 139 apps mined from Apple App Store. They reported that opinions on product quality form a larger portion of reviews, but opinions on service quality have a bigger effect on sales.

In order to bridge the gap between software attributes and user reviews, Hoon et al. [96] developed an ontology of words used to describe software quality attributes in app reviews. A summary of mobile app user feedback classification can be found in the study by Maalej et al. [226].

9 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, but rather use properties such as the validation that highly rated apps have received, through being downloaded, used, and highly rated by thousands of users. Much of the security-related literature uses the property that popular apps can generally be assumed non-malware, since they are scanned prior to being hosted in the store, and have large user bases. Almost all dedicated security research in this section that mines app stores as software repositories use the Google Play store; that is, there are no studies on Blackberry or Apple, and just one study on Windows. Many studies in this section use large collections (>10,000 apps) of benign apps for malware detection. The number of apps used ranges from 1 to 998,286 (the largest study in this survey).

In 2010 Bläsing et al. [26] used the top 150 free Google Play apps to test their static and dynamic APK analyser. They tested these apps against 1 known malware app which was shown to be an outlier, proving that their approach can work for malware detection. In 2011 Batyuk et al. [19] used the top 1,865 free Google Play apps to test their static APK analyser, which detected that 167 apps accessed private identifiers, presenting a security risk. 114 of these apps wrote the information after reading it, which might indicate that the apps contain spyware. The work has since been extended into a static analysis tool called `Androlyzer` [51].

In 2012 Potharaju et al. [177] conducted a study on 158,000 free Android apps, identifying apps that are likely to be plagiarised in order to spread malware. The authors found that the 29.4% of apps with the most permissive rights are most likely to spread malware, and that non-technical information such as category, number of downloads and publishing day can increase the initial spread of the malware. Chia et al. [41] evaluated the ratings of apps from Facebook, Chrome and Google Play, as a means of warning against privacy risks.

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. [26], 2010	g	MALWARE	150
Batyuk et al. [19], 2011	g	MALWARE	1,865
Potharaju et al. [177], 2012	g	ESSoS	158,000
Moller et al. [164], 2012	g	LARGE	1
Chia et al. [41], 2012	g	WWW	19,344
Gibler et al. [71], 2012	g	TRUST	24,350
Grace et al. [80], 2012	g	WiSec	100,000
Crussell et al. [48], 2012	g	ESORICS	75,000
Peng et al. [175], 2012	g	CCS	500,000
Zhu et al. [252], 2015	g	ICICS	5,685
Awang Abu Bakar and Mahmud [13], 2013	g	ACSAT	5,000
Stevens et al. [203], 2013	g	MSR	10,300
Book et al. [28], 2013	g	CoRR	114,000
Sanz et al. [191], 2013	g	Cybernet. Syst.	333
Sanz et al. [193], 2013	g	SECRYPT	333
Sanz et al. [192], 2013	g	NSS	333
Wang et al. [231], 2013	g	DBSec	272,774
Crussell et al. [49], 2013	g	ESORICS	265,359
Gibler et al. [72], 2013	g	MobiSys	265,359
Peiravian and Xingquan [174],'13	g	ICTAI	1,250
Chakradeo et al. [35], 2013	g	WiSec	14,888
Pandita et al. [171], 2013	g	SEC	581
Zhu et al. [249], 2013	a	CIKM	15,045
Liu et al. [141], 2014	w	NSDI	51,150
Crussell et al. [50], 2014	g	MobiSys	165,426
Gorla et al. [78], 2014	g	ICSE	32,136
Ham and Lee [89], 2014	g	IJCCE	10
Bhoraskar et al. [25], 2014	g	SEC	1,010
Qu et al. [180], 2014	g	CCS	45,811
Zhu et al. [251], 2015	a	TKDE	15,045
Schütte et al. [196], 2015	g	ConDroid	10,000
Mutchler et al. [163], 2015	g	MoST	998,286
Avdiienko et al. [12], 2015	g	ICSE	2,866
Ma et al. [143], 2015	g	COMPSAC	22,555
Vigneri et al. [218], 2015	g	CoRR	5,000
Yang et al. [238], 2015	g	ICSE	633
Lageman et al. [122], 2015	g	MILCOM	417
Deng et al. [55], 2015	a	CCS	2,019
Zhang et al. [242], 2015	g	CCS	100
Huang et al. [100], 2015	g	SEC	16,000
Chen et al. [37], 2015	g	SEC	1,165,847
		Mean	106,933
		Median	14,888

They found a strong correlation between popularity and the number of ratings apps receive, but no correlations between permissions asked and privacy risk, nor rating. This result shows that ratings are not an effective indicator of the privacy of apps, and new suspicious apps are not likely to receive many ratings which could serve as a warning for future users. Moller et al. [164] studied the update behaviour of users following recent updates, finding from a case study that approximately half of users did not update their app for at least a week after the update. The authors argue that this could lead to users continuing to run vulnerable software even after a fix is available.

Gibler et al. [71] 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. Crussell et al. [48] introduced the tool `DNADroid`, which they used to identify 141 cloned apps in the Google Play store, from a mined set of 75,000 apps. The authors then introduced the tool `AnDarwin`, which decompiles apps and compares their to detect clones [49]. They detected 4,295 cloned apps using this approach from a mined set of 265,239 apps. This dataset is used in the study by Gibler et al. [72], who investigated the effects of application plagiarism on developers.

Peng et al. [175] 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. [252] proposed an approach to malware detection by using permission and description information to detect abnormal permission sets. They evaluated the system on 5,685 apps mined from Google Play and found some words that have a large effect on permission validity; they also tested the system on known malware and found that it was able to successfully detect it as such.

In 2013 Awang Abu Bakar and Mahmud [13] mined 5,000 apps from the Google Play store and analysed their permissions. They found extremely weak correlations between (technical) the number of permissions asked for and (non-technical) the price, download rank and rating. They highlight the top permissions requested by apps, and found that 40% of the apps requested the phone's status and identity, a source of sensitive information. Stevens et al. [203] mined 10,300 apps from several Android stores including Google Play and applied the permissions analysis tool `Stowaway` [11] that can detect declared and used permissions. The authors found that 44% of apps in their sample contained at least one unnecessary permission, and computed a Spearman's correlation coefficient of 0.72 between the popularity of permissions on Stackoverflow and their misuse. Chakradeo et al. [35] created an app malware triaging tool call `MAST`, which they trained on known malware and a set of 14,888 apps mined from Google Play (that are assumed to be benign). Peiravian and Xingquan [174] trained a malware classifier using 1,250 samples of known malware, and 1,250 samples of benign apps mined from Google Play. They trained the classifier using information on the permissions requested and the API calls made by the apps.

Grace et al. [80] introduced `AdRisk`, a static analysis tool for identifying potential privacy risks associated with ad libraries. From their study on 100,000 apps mined from Google Play, the authors found that 52,067 apps use ad libraries, of which 31% use more than one. The authors remarked that the majority of 100 studied ad libraries were found to collect personal information. Book

et al. [28] studied library permissions on 114,000 apps mined from the Google Play store, showing libraries bundled with apps lead to old versions being included. Increasingly ad libraries take advantage of app permissions presenting a security risk, which the authors argue should be solved by the app store or privacy legislation.

Sanz et al. [193] used cosine similarity between the sets of features declared in Android manifest files, in order to detect anomalies that could be 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 [192, 191]. Similarly, Wang et al. [231] 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` [4] is run to detect malware and remove it from the store. Zhu et al. [249, 251] mined ranking, rating and reviewing data from 15,045 apps from the Apple App Store. They detected outliers using hypothesis tests in order to find potential 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'.

Pandita et al. [171] presented the `WHYPER` system for automatically extracting the reason a permission is used from the description. They evaluated the system using 581 apps mined from Google Play, which were manually labelled by the authors. The authors tested the system on the permissions address book, calendar and audio recording, and achieved an average precision of 82.2% and recall of 81.5%. In a related study, Qu et al. [180] introduced `AutoCog`, a tool for checking the fidelity between app descriptions and requested permissions. The authors tested the system on 45,811 Google Play apps, and achieved a precision of 92.6% and a recall of 92.0% when detecting 11 permissions. In 2015, Zhang et al. [242] argued that the descriptions given to apps contain insufficient security information. The authors presented the `DescribeMe` system, that generates security-centric descriptions using static analysis. They performed a user study using Amazon's Mechanical Turk [6], on a set of 100 apps, asking whether the generated descriptions are readable and whether they can reduce the rate at which users download malware. The generated descriptions achieved a 4% lower readability rating than the original human-written descriptions, but decreased the malware download rate by 39%.

Ravindranath et al. [182] 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. [141] presented their `DECAF` system for detecting ad placement and layout violations, which can indicate ad 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. [50] presented `MAdFraud`, a system that detects ad fraud in the form of requesting ads while the application is in the background, and in the form of simulating user clicks on ads. They tested the system on 165,426 apps gathered from Google Play and a separate security company, and found that 30% of apps made ad requests while running in the background, and 27

apps simulated user clicks on ads.

As a means of detecting potentially malicious apps, Gorla et al.[78] performed topic modelling on app descriptions, and then applied K-means to the results to form distinct clusters. Utilising API information from the app manifest, the authors trained a one-class support vector machine (SVM) [147] on each cluster to detect outliers in terms of API usage. This method indicates which apps are not doing what apps of a similar nature typically do, and could be used to detect malware. This approach was later extended by Ma et al. [143] who used known malware and benignware to train their model, and reported improvements on the resultant precision, recall and F-measure.

Avdiienko et al. [12] extracted information flow data from apps in order to train a benign-trained malware classifier. The classifier was trained on 2,950 of the most popular Google Play apps, which are assumed benign as their download rank is in the top 100 in each of 30 categories. In this way, the authors combined non-technical information (download rank) with extracted technical information (information flow) to detect malware. The system reported high precision on sets of known malware from the Genome project [244] and VirusShare database [220]. In a similar way, the 2013 study by Sanz et al. [191] 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 [89], also used apps from the Google Play Games category as a benign set.

In 2015 Schütte et al. [196] 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. [163] took a snapshot of 1,172,610 Google Play apps. They found that 998,286 of these apps used `WebView`, indicating that the apps use an embedded `WebView` in some way. The authors searched for several known vulnerabilities and found that 28% of the studied apps had at least one of these vulnerabilities. As a result, the authors propose a set of API changes to mitigate such threats. In a similar study Bhoraskar et al. [25] 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 13 of 200 apps using Facebook SDK are vulnerable to known attacks, and 175 of 220 children's apps potentially collect information in violation of the US Children's Online Privacy Protection Act [44].

Vigneri et al. [218] used a set of 5,000 apps mined from Google Play, on which they performed dynamic execution to determine network usage. They focused, in particular, on network activity to URLs which 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. Yang et al. [238] used 633 apps mined from Google Play as the benign set to test their tool for distinguishing between malicious and benign apps. They found that the intent of security accesses is more related to whether an app is malicious than the type of security-sensitive resources that it accesses. Lageman et al. [122] generated feature sets to be used for classification of malware and benignware, from runtime log datasets of 419 malware apps and 417 mined benign apps. They tested the feature set and

achieved a true positive rate of 90% with a Random Forest classifier [75].

Deng et al. [55] introduced their iRiS system, which performs static analysis on iOS apps in order to detect suspicious apps that may violate Apple's terms of service. The authors detected 146 apps from a sample of 2019, that accessed sensitive user information through use of private APIs. Huang et al. [100] 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, obtaining a precision of 0.973 and a recall of 0.973, with a false positive rate of 0.087. The cases of entry fields found include national ID, username, password, credit card and health data. In the largest app study to date, Chen et al. [37] 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, and when run on the entire dataset, detected 127,429 instances of malware and 20 likely instances of zero-day malware.

App security has been well studied in the literature, and perhaps warrants a survey of its own. There are a host of studies into 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. There is potential future work in this area in augmenting approaches with the non-technical information made available by app stores.

10 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. There are potential research opportunities to be found comparing stores, especially concerning the Windows Phone Store, which has taken off slowly but continues to grow. The scale of studies in this section is small, ranging between 0 and 10,150 apps, with a median of just 15. It is therefore also an opportunity to expand on the scale of the discussed studies, in order to negate any bias introduced by the small datasets used so far.

In 2011, Syer et al. [205] studied the different code practices between app stores, by selecting 3 pairs of feature-equivalent apps from Android and Blackberry. The authors analysed the source code, code dependencies and code churn of these apps, and found that Android apps are generally smaller but rely heavily on the platform. Conversely, Blackberry apps are larger and rely heavily on 3rd-party APIs. In order to reach the largest customer base developers need to cater for each platform, and so the authors remarked that it is therefore easier to develop for Blackberry and port to Android than the reverse.

In 2013 McDonnell et al. [151] studied 10 apps using source code from github [74]. The Android platform was shown to be evolving fast with an average of 115 API updates per month, due to which 28% of Android references were out of date, and the median lag time to update to support a new API was found to be 16 months. The APIs used most were the ones updated most frequently, yet interestingly API updates are more defect prone than other changes

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. 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.

Authors [Ref], Year	**Store**	**Venue**	**No. apps**
Syer et al. [205], 2011	b,g	SCAM	3
d'Heureuse et al. [57], 2012	a,b,g,w	MCCR	1,164,489
Jung et al. [111], 2012	a	Market Lett	1,189
Garg and Telang [70], 2013	a	MIS	1,223
Lim and Bentley [129], 2012	a	GECCO	*
Lim and Bentley [128], 2012	a	ALIFE	*
Lim and Bentley [130], 2013	a	CEC	*
Zhong & Michahelles [243],'13	g	SAC	191,301
Petsas et al. [176], 2013	g	IMC	316,143
Syer et al. [207], 2013	g	CASCON	15
McDonnell et al. [151], 2013	g	ICSM	10
Cocco et al. [42], 2014	a	MWIS	***
Wenxuan and Airu [234],'14	a,g,w	ICDMW	736,377
Ng et al. [167], 2014	g	COMPSAC	506
Liu et al. [139], 2015	g	WSDM	6,157
Ruiz et al. [158], 2015	g	IEE Soft.	10,150
Joorabchi et al. [110], 2015	a,g	ISSRE	14
Gómez et al. [76], 2015	g	ICSE NIER	1
Askalidis [10], 2015	a	CoRR	162
Xie and Zhu [236], 2015	a	WiSec	179,353
Corral and Fronza [45], 2015	g	MOBILESoft	100
Lim et al. [131], 2015	a	TEVC	**
		Mean	144,844
		Median	848

in client code. Syer et al. [207] later compared development practices between 15 Android apps and 5 traditional desktop and server applications. They found that mobile apps are most similar to `Unix` utilities, in terms of smaller code bases and small development teams. However, mobile apps suffer from greater numbers of defects and slower fix times than the studied traditional applications.

In 2012 d'Heureuse et al. [57] mined 1,164,489 total apps from Apple, Blackberry, Google and Windows app stores at regular intervals over a period of 3 months, in order to perform cross store comparison and also to study growth over time. The authors found that although Google and Apple are much larger stores, Blackberry and Windows show rates of new apps and removed apps, and Google has twice the rate of app deletion of other stores. Google also showed a higher growth rate, and so the authors predicted that it would overtake Apple in total apps in February 2013. 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 appear in multiple markets were on average 7.15 MB larger in the Apple store, and were a similar size in the 3 other stores. Jung et al. [111] assessed the differences between free and paid apps on Apple's Korean App Store. They found that customer ratings are more critical to the survival of free apps, and there is also more benefit to getting an early entrant in markets.

Lim and Bentley simulated the app store ecosystem using an agent-based evolutionary model, in order to experiment with different publicity strategies [129, 128], 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 if the app appears on the 'new apps' chart. The authors then used the model to explore different ranking algorithms [130]. 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 performs best in terms of overall downloads by incorporating fresh apps, and for this to work it needs to incorporate less historical performance data (also found later by Ruiz et al. [158]). This study is unique and provides useful findings for app store maintainers. Lim et al. later simulated the ecosystem from a user's perspective [131], using collected usage information from over 10,000 participants [127]. They modelled developer strategies such as 'innovator' (who produces apps with random features) and 'copycat' (who copies the app) [131]. They found that 'optimiser' (who improves on the original 'innovator' apps) and 'copycat' working together led to the best overall fitness, provided they represented a low proportion of the overall modelled developer population.

Cocco et al. [42] extended the model used by Lim and Bentley, and investigated additional ranking algorithms and user behaviour. They explored store ranking algorithms, and found that a 1% chance of a new app appearing in the top charts leads to the highest downloads-to-browse ratio. Apps in Google Play do not have accessible information on their number of downloads, other than 'buckets', such as the range 50-100. Zhong and Michahelles [243] 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 account for the majority of app downloads, and also have high ratings indicating customer satisfaction. Paid apps achieved more success if they were cheaper, but expen-

sive professional apps have disproportionally high numbers of downloads. The authors conclude that developers can break into the top downloads numbers by fulfilling a niche market.

Petsas et al. [176] analysed the downloads of 316,143 apps from 4 third-party Android app stores. They found that 10% of the apps account for at least 70% of the total downloads in the stores, and that user downloads follow a clustering type behaviour, where their subsequent app downloads are usually in the same category. The authors also found that popularity follows a power-law distribution against app price, for paid apps. Garg and Telang [70] 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. Ng et al. [167] 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 if they are the same as the official, and ranked the severity level of differences. The authors concluded that the third party app stores studied cannot be trusted, as the proportion of apps which do not match their official releases is high, as are the corresponding difference severity levels.

In 2014 Wenxuan and Airu [234] 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. [139] also studied app recommendation systems, by incorporating the level of privacy that the app needs as well as user interests. They evaluated their approach using 6,157 apps mined from Google Play, and found that their recommender performed better when treating each app function with different privacy allowances. They use the rating distribution over their dataset as the motivation for modelling user preference with a Poisson distribution.

In 2015 Ruiz et al. [158] conducted a longitudinal rating study on 10,150 apps over the period of 12 months. They argue 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. As it stands, the 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. Askalidis [10] studied the effects of sales promotions in the Apple App Store on 162 apps, finding that rival apps can benefit from the promotion if the apps are not reduced to become significantly cheaper than them, and that sales where apps become free, or have easily redeemable digital discounts are the most successful. Sales are shown to have mixed effects on the ratings of apps. Joorabchi et al. [110] introduced `CheckCAMP`, a tool that checks for inconsistencies between Android and iOS versions of the same app. The authors tested the tool on 7 open source apps and 7 industry apps, and validated their results with a user study, finding an F-measure of 1.0 on the open source apps and an F-measure of 0.92 on the industry apps.

Gómez et al. [76] proposed an app store feature of automatically patching defective apps, which they demonstrate by automatically fixing a defective app mined from Google Play. Xie and Zhu [236] investigated the practice of promoting apps through buying positive reviews, via illicit "underground" services. 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 [199],'11	ISMA	6
Preuss [178], 2012	The IFPUG Guide to IT & Software Measurement	1
Preuss [179], 2013	ICEAA	1
van Heeringen and van Gorp [214], 2014	IWSM-MENSURA	0
Abdullah et al. [1],'14	ICOS	0
D'Avanzo et al. [53],'15	SAC	8
Francese et al. [65],'15	SEAA	23
Ferrucci et al. [62],'15	SEAA	13
Ferrucci et al. [63], 2015	PROFES	13
	Mean	7
	Median	6

authors registered on 8 such app promotion sites and exposed approximately 30,000 promoted apps. Their tool, `AppWatcher` was used to collect information from 179,353 randomly selected iOS apps, from which they mined 9,399,014 reviews. The authors reported on differences between datasets of promoted and random apps. Corral and Fronza [45] 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.

11 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 studies up to November 27, 2015 is relatively small compared with other sections of App Store Analysis, but as the field has shown strong growth in 2015, it seems likely that the scale of studies will increase in the future.

In 2011 Sethumadhavan [199] 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 [178, 179] 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 [214] discussed how to use the `COSMIC` method [46] to measure the functional size of apps. Abdullah et al. [1] discussed using the `COSMIC` method to estimate game apps, using an intermediate representation of required assets and functionality in the `Unity3D` game engine.

2015 saw more much more work in this area. D'Avanzo et al. [53] applied the COSMIC approach to 8 Google Play apps, and applied linear regression to the functional point size in order to estimate the code size. By applying leave-one-out cross validation, the authors showed that the approach can accurately estimate code size based on functionality alone, once trained. Francese et al. [65] 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. [62] applied the COSMIC approach to 13 Android applications, showing that functional size is strongly correlated with app size, and that it can be used to accurately estimate the bytecode size of the app. Ferrucci et al. [63] later compared the related approaches by D'Avanzo et al. [53] and van Heeringen and van Gorp [214] on their dataset of 13 Android apps. They found that both functional size results were correlated with multiple app size measures, but that the approach presented by D'Avanzo et al. [53] was more accurate.

12 Other sources of non-technical information

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. [206] uses information on the number of downloads from AppBrain, a replication of the number of installs bracket on Google Play (eg. 1,000,000 - 5,000,000 installs appears on AppBrain as 1,000,000+). Ihm et al. [105] combined download information on 10 social networking apps from Google Play with the number of registered users on their respective websites. They unsurprisingly found a strong correlation between the two metrics.

13 Closely Related Work

The following literature is important to the field of App Store Analysis, yet itself does not meet our exact definition of App Store Analysis. Nevertheless, since this work meets aspects of 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. These studies are important to the field of App Store Analysis and so are included here.

In 2011 Böhmer et al. [27] 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 is less than 72 seconds long, and that smartphones are used for almost 60 minutes every day. The type of application was found to differ between times of day, such as news applications in the morning and games at night. The

exception to this rule was communication apps, which are used throughout the day.

In 2012 Ferreira et al. [61] surveyed 4,035 Android user charging habits, using an app to record their behaviour. Lin et al. [132] conducted a survey on 179 Android users, that asked about their expectations of app's 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. Rein and Münch [183] 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. [168] surveyed 100 app users and found that users are more likely to take a passive approach and delete apps rather than reviewing or contacting developers, but when they take an active approach, reviewing is the most popular approach. In 2014 Tan et al. [208] surveyed users and developers of the Apple App Store, regarding the iOS permission request explanation feature. The feature is 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. [127] surveyed app users from 15 countries to understand how usage of apps and app stores differs by region. They found that behaviour does differ significantly by region in many regards. In Eastern regions such as China and India a greater proportion of users participate in recommendation and rating of apps, almost 4 times the proportion of Western users. Additionally, the survey finds that app abandonment due to issues is higher than average in Brazil and the UK, and lower than average in Japan and France, indicating that differences are affected by more than global region. It is a unique study as it gathers user information regarding multiple app stores across a large number of global app users: the focus is on usage, not on apps, yet the authors identified actionable findings for app developers.

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 9.

Enck et al. [59] introduced `Kirin`, an Android app certification tool for flagging potential malware using a set of rules. In 2010 Enck et al. introduced `TaintDroid` [58], 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. [9] in 2014, called `FlowDroid`. This tool statically analyses information flow to find all possible flows.

Some authors have used sets mined from Google Play as benign app sets to test against known malware: Xu et al. [237], Rastogi et al. [181], Jing et al. [108], Arp et al. [8], Wang et al. [230], Liu and Liu [142], Roy et al. [185] and Khanmohammadi et al. [118]. Ho et al. [95] 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. [16], Jeon et al. [106], Grace et al. [81], Crussell et al. [48, 49], Ravindranath et al. [182], von Rhein et al. [223], Li et

al. [125], Huang et al. [101], Cen et al. [32], Liu et al. [140] and Bastani et al. [18].

13.3 Reports

Initial studies such as the 2010 work by Sharma et al. [201] evaluated the size and growth of the apps market up to the time. In 2011 Butler [30] conducted a study on the Android system, highlighting how it is changing mobile development by enabling people with no prior development experience to release an app. In 2012 Shuler [202] published a report on the Apple App Store Education category, comparing it with their previous study in 2009. They found that over 72% of the top-selling apps in this category target children aged 10 or below, a number that has significantly increased from 47% in 2009. Additionally, the average price of an app had risen by 1 USD since 2009, and the majority of top Education developers in 2012 had not been present in 2009.

The 2013 report by Vision Mobile [221] on app industry monetary value and growth found that 72% of developers are dedicated to Android. iOS and Android developers earn on average double that of developers of other platforms, and iOS is considered the highest priority platform. As of 2013, iOS, Android and Blackberry were the leading platforms, despite Blackberry's increasing 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, Awang Abu Bakar and Mahmud [15] published `OSSGrab` which mines `HTML` pages from Google Play. The tool was built in order to facilitate their app permissions study [13]. In 2014, Viennot et al. introduced the `PlayDrone` Google Play crawler [216], to facilitate their large scale API study [217].

The Android Malware Genome Project [244] is a popular source of malware applications for testing security tools. In 2015 Krutz et al. [121] made available a dataset containing 1,179 open source applications.

14 Future Work

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. An avenue for future research concerns the extraction of non-technical information from app stores, and extracting samples of apps whilst dealing with 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. For discussions on future avenues of research in software engineering for mobile apps, we refer the reader to the works by Al-Subaihin et al. [3], and Nagappan and Shihab [165].

15 Conclusions

We have surveyed the published literature in App Store Analysis for software engineering, and identified the key sub-fields of App Store Analysis to date: API analysis, feature analysis, release engineering, review analysis, security analysis, store ecosystem comparison, and analysis of size and effort prediction. The four largest fields are API, feature, review and security analysis, but release engineering and prediction have shown strong growth in 2015 and may outgrow other sub-fields in the future. 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.

References

[1] Nur Atiqah Sia Abdullah, Nur Ida Aniza Rusli, and Mohd Faisal Ibrahim. Mobile game size estimation: Cosmic fsm rules, uml mapping model and unity3d game engine. In *Open Systems (ICOS), 2014 IEEE Conference on*, pages 42–47. IEEE, 2014.

[2] Nishant Agarwal, Reza Karimpour, and Guenther Ruhe. Theme-based product release planning: An analytical approach. In *System Sciences (HICSS), 2014 47th Hawaii International Conference on*, pages 4739–4748. IEEE, 2014.

[3] Afnan Al-Subaihin, Anthony Finkelstein, Mark Harman, Yue Jia, William Martin, Federica Sarro, and Yuanyuan Zhang. App store mining and analysis. In *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*, DeMobile 2015, pages 1–2. ACM, 2015.

[4] Chloe Albanesius. Mobile app reviews: Google 'Bouncer' now scanning Android Market for malware, 2012.

[5] Khalid Alharbi and Tom Yeh. Collect, decompile, extract, stats, and diff: Mining design pattern changes in Android apps. In *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services*, MobileHCI '15, pages 515–524. ACM, 2015.

[6] Inc. Amazon.com. Amazon Mechanical Turk, 2013.

[7] androguard. androguard, 2015.

[8] Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of Android malware in your pocket. In *Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS)*, 2014.

[9] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '14)*, pages 259–269. ACM, 2014.

[10] Georgios Askalidis. The impact of large scale promotions on the sales and ratings of mobile apps: Evidence from Apple's App Store. *CoRR*, abs/1506.06857, 2015.

[11] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. Pscout: analyzing the Android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 217–228. ACM, 2012.

[12] Vitalii Avdiienko, Konstantin Kuznetsov, Alessandra Gorla, Andreas Zeller, Steven Arzt, Siegfried Rasthofer, and Eric Bodden. Mining apps for abnormal usage of sensitive data. In *2015 International Conference on Software Engineering (ICSE)*, 2015.

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

[14] Shams Abubakar Azad. Empirical studies of Android API usage: Suggesting related API calls and detecting license violations. Master's thesis, Concordia University, 2015.

[15] Awang Abu Bakar, Normi Sham Mahmud, and Iqram Mahmud. OSSGrab: Software repositories and app store mining tool. *Lecture Notes on Software Engineering*, 1(3):219–223, 2013.

[16] David Barrera, H. Güneş Kayacik, Paul C. van Oorschot, and Anil Somayaji. A methodology for empirical analysis of permission-based security models and its application to Android. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 73–84. ACM, 2010.

[17] Alexandre Bartel, John Klein, Martin Monperrus, and Yves Le Traon. Static analysis for extracting permission checks of a large scale framework: The challenges and solutions for analyzing Android. *Software Engineering, IEEE Transactions on*, 40(6):617–632, 2014.

[18] Osbert Bastani, Saswat Anand, and Alex Aiken. Interactively verifying absence of explicit information flows in Android apps. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA 2015, pages 299–315. ACM, 2015.

[19] Leonid Batyuk, Markus Herpich, Seyit Ahmet Camtepe, Karsten Raddatz, Aubrey-Derrick Schmidt, and Sahin Albayrak. Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications. In *Proceedings of the 2011 6th International Conference on Malicious and Unwanted Software*, MALWARE '11, pages 66–72. IEEE Computer Society, 2011.

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

[21] David Lavid Ben Lulu and Tsvi Kuflik. Functionality-based clustering using short textual description: Helping users to find apps installed on their mobile device. In *Proceedings of the 2013 International Conference on Intelligent User Interfaces*, IUI '13, pages 297–306. ACM, 2013.

[22] David Lavid Ben Lulu and Tsvi Kuflik. Wise mobile icons organization: Apps taxonomy classification using functionality mining to ease apps finding. *Mobile Information Systems*, 2016, 2015. Article ID: 3083450.

[23] Giacomo Berardi, Andrea Esuli, Tiziano Fagni, and Fabrizio Sebastiani. Multi-store metadata-based supervised mobile app classification. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, pages 585–588. ACM, 2015.

[24] Pamela Bhattacharya, Liudmila Ulanova, Iulian Neamtiu, and Sai Charan Koduru. An empirical analysis of bug reports and bug fixing in open source Android apps. In *Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering*, CSMR '13, pages 133–143. IEEE Computer Society, 2013.

[25] Ravi Bhoraskar, Seungyeop Han, Jinseong Jeon, Tanzirul Azim, Shuo Chen, Jaeyeon Jung, Suman Nath, Rui Wang, and David Wetherall. Brahmastra: Driving apps to test the security of third-party components. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 1021–1036. USENIX Association, 2014.

[26] Thomas Bläsing, Leonid Batyuk, Aubrey-Derrick Schmidt, Seyit Ahmet Çamtepe, and Sahin Albayrak. An Android application sandbox system for suspicious software detection. In *Proceedings of the 5th International Conference on Malicious and Unwanted Software, MALWARE'10*, pages 55–62, 2010.

[27] Matthias Böhmer, Brent Hecht, Johannes Schöning, Antonio Krüger, and Gernot Bauer. Falling asleep with angry birds, facebook and kindle: A large scale study on mobile application usage. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '11, pages 47–56. ACM, 2011.

[28] Theodore Book, Adam Pridgen, and Dan S. Wallach. Longitudinal analysis of android ad library permissions. *CoRR*, abs/1303.0857, 2013.

[29] Hudson S Borges and Marco Tulio Valente. Mining usage patterns for the Android API. *PeerJ Computer Science*, (1:e12), 2015.

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

[31] Bogdan Carbunar and Rahul Potharaju. A longitudinal study of the google app market. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, ASONAM '15, pages 242–249. ACM, 2015.

[32] Lei Cen, Christopher S. Gates, Luo Si, and Ninghui Li. A probabilistic discriminative model for Android malware detection with decompiled source code. *IEEE Trans. Dependable Sec. Comput.*, 12(4):400–412, 2015.

[33] Lei Cen, Deguang Kong, Hongxia Jin, and Luo Si. Mobile app security risk assessment: A crowdsourcing ranking approach from user comments. In *Proceedings of the 2015 SIAM International Conference on Data Mining*, pages 658–666, 2015.

[34] Lei Cen, Luo Si, Ninghui Li, and Hongxia Jin. User comment analysis for Android apps and CSPI detection with comment expansion. In *Proceeding of the 1st International Workshop on Privacy-Preserving IR (PIR)*, pages 25–30, 2014.

[35] Saurabh Chakradeo, Bradley Reaves, Patrick Traynor, and William Enck. Mast: Triage for market-scale mobile malware analysis. In *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '13, pages 13–24. ACM, 2013.

[36] Rishi Chandy and Haijie Gu. Identifying spam in the iOS App Store. In *Proceedings of the 2Nd Joint WICOW/AIRWeb Workshop on Web Quality*, WebQuality '12, pages 56–59. ACM, 2012.

[37] Kai Chen, Peng Wang, Yeonjoon Lee, XiaoFeng Wang, Nan Zhang, Heqing Huang, Wei Zou, and Peng Liu. Finding unknown malice in 10 seconds: Mass vetting for new threats at the Google-Play scale. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, pages 659–674. USENIX Association, 2015.

[38] Miao Chen and Xiaozong Liu. Predicting popularity of online distributed applications: iTunes App Store case analysis. In *Proceedings of the 2011 iConference*, iConference '11, pages 661–663. ACM, 2011.

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

[40] Ying Chen, Heng Xu, Yilu Zhou, and Sencun Zhu. Is this app safe for children?: A comparison study of maturity ratings on Android and iOS applications. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 201–212. International World Wide Web Conferences Steering Committee, 2013.

[41] Pern Hui Chia, Yusuke Yamamoto, and N. Asokan. Is this app safe?: A large scale study on application permissions and risk signals. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 311–320. ACM, 2012.

[42] Luisanna Cocco, Katiuscia Mannaro, Giulio Concas, and Michele Marchesi. Simulation of the best ranking algorithms for an app store. In *Mobile Web Information Systems*, volume 8640 of *Lecture Notes in Computer Science*, pages 233–247. Springer International Publishing, 2014.

[43] Stefano Comino, Fabio M Manenti, and Franco Mariuzzo. Updates management in mobile applications. iTunes vs Google Play. *Centre for Competition Policy (CCP), University of East Anglia*, 2015.

[44] Federal Trade Commission. Complying with COPPA: Frequently Asked Questions, 2015.

[45] Luis Corral and Ilenia Fronza. Better code for better apps: A study on source code quality and market success of Android applications. In *Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems*, MOBILESoft '15, pages 22–32. IEEE Press, 2015.

[46] COSMIC. Common software measurement international consortium, 2015.

[47] Paul Coulton and Will Bamford. Experimenting through mobile 'apps' and 'app stores'. *Int. J. Mob. Hum. Comput. Interact.*, 3(4):55–70, 2011.

[48] Jonathan Crussell, Clint Gibler, and Hao Chen. Attack of the clones: Detecting cloned applications on Android markets. In *Computer Security–ESORICS 2012*, pages 37–54. Springer, 2012.

[49] Jonathan Crussell, Clint Gibler, and Hao Chen. Andarwin: Scalable detection of semantically similar Android applications. In *Computer Security–ESORICS 2013*, pages 182–199. Springer, 2013.

[50] Jonathan Crussell, Ryan Stevens, and Hao Chen. Madfraud: Investigating ad fraud in Android applications. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 123–134. ACM, 2014.

[51] DAI-Labor. Androlyzer, 2015.

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

[53] Loris D'Avanzo, Filomena Ferrucci, Carmine Gravino, and Pasquale Salza. Cosmic functional measurement of mobile applications and code size estimation. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, pages 1631–1636. ACM, 2015.

[54] Horace Dediu. When will Android reach one billion users?, 2012. Accessed 28th April 2014.

[55] Zhui Deng, Brendan Saltaformaggio, Xiangyu Zhang, and Dongyan Xu. iris: Vetting private API abuse in iOS applications. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 44–56. ACM, 2015.

[56] Matthew L. Dering and Patrick McDaniel. Android Market reconstruction and analysis. In *Proceedings of the 2014 IEEE Military Communications Conference*, MILCOM '14, pages 300–305. IEEE Computer Society, 2014.

[57] Nico d'Heureuse, Felipe Huici, Mayutan Arumaithurai, Mohamed Ahmed, Konstantina Papagiannaki, and Saverio Niccolini. What's app?: a wide-scale measurement study of smart phone markets. *ACM SIGMOBILE Mobile Computing and Communications Review*, 16(2):16–27, 2012.

[58] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–6. USENIX Association, 2010.

[59] William Enck, Machigar Ongtang, and Patrick McDaniel. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 235–245. ACM, 2009.

[60] Dejan Erić, Radovan Bačík, and Igor Fedorko. Rating decision analysis based on iOS App Store data. *Quality Innovation Prosperity*, 18(2):27–37, 2014.

[61] Denzil Ferreira, Vassilis Kostakos, and Anind K. Dey. Lessons learned from large-scale user studies: Using Android Market as a source of data. *Int. J. Mob. Hum. Comput. Interact.*, 4(3):28–43, 2012.

[62] Filomena Ferrucci, Carmine Gravino, Pasquale Salza, and Federica Sarro. Investigating functional and code size measures for mobile applications. In *41st Euromicro Conference series on Software Engineering and Advanced Applications (SEAA '15)*, pages 365–368. IEEE, 2015.

[63] Filomena Ferrucci, Carmine Gravino, Pasquale Salza, and Federica Sarro. Investigating functional and code size measures for mobile applications: A replicated study. In *16th International Conference on Product-Focused Software Process Improvement (PROFES '15)*, 2015.

[64] Anthony Finkelstein, Mark Harman, Yue Jia, William Martin, Federica Sarro, and Yuanyuan Zhang. App store analysis: Mining app stores for relationships between customer, business and technical characteristics. Technical report, 2014.

[65] Rita Francese, Carmine Gravino, Michele Risi, Giuseppe Scanniello, and Genoveffa Tortora. On the use of requirements measures to predict software project and product measures in the context of Android mobile apps: a preliminary study. In *41st Euromicro Conference series on Software Engineering and Advanced Applications (SEAA '15)*, pages 357–364. IEEE, 2015.

[66] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Mach. Learn.*, 29(2-3):131–163, 1997.

[67] Bin Fu, Jialiu Lin, Lei Li, Christos Faloutsos, Jason Hong, and Norman Sadeh. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 1276–1284. ACM, 2013.

[68] Laura V. Galvis Carreño and Kristina Winbladh. Analysis of user comments: An approach for software requirements evolution. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 582–591. IEEE Press, 2013.

[69] Cuiyun Gao, Hui Xu, Junjie Hu, and Yangfan Zhou. Ar-tracker: Track the dynamics of mobile apps via user review mining. In *2015 IEEE Symposium on Service-Oriented System Engineering, SOSE '15*, pages 284–290, 2015.

[70] Rajiv Garg and Rahul Telang. Inferring app demand from publicly available data. *MIS Q.*, 37(4):1253–1264, 2013.

[71] Clint Gibler, Jonathan Crussell, Jeremy Erickson, and Hao Chen. AndroidLeaks: Automatically detecting potential privacy leaks in Android applications on a large scale. In *Proceedings of the 5th International Conference on Trust and Trustworthy Computing*, TRUST'12, pages 291–307. Springer-Verlag, 2012.

[72] Clint Gibler, Ryan Stevens, Jonathan Crussell, Hao Chen, Hui Zang, and Heesook Choi. Adrob: Examining the landscape and impact of Android application plagiarism. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 431–444. ACM, 2013.

[73] GinLemon. Smart launcher 3 — simple. light. fast., 2011.

[74] GitHub, Inc. Github, 2014.

[75] William Glodek and Richard Harang. Rapid permissions-based detection and analysis of mobile malware using random decision forests. In *Military Communications Conference, MILCOM 2013-2013 IEEE*, pages 980–985. IEEE, 2013.

[76] María Gómez, Matias Martinez, Martin Monperrus, and Romain Rouvoy. When app stores listen to the crowd to fight bugs in the wild. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ICSE '15, pages 567–570. IEEE Press, 2015.

[77] Maria Gomez, Romain Rouvoy, Martin Monperrus, and Lionel Seinturier. A recommender system of buggy app checkers for app store moderators. In Danny Dig and Yael Dubinsky, editors, *2nd ACM International Conference on Mobile Software Engineering and Systems*. IEEE, 2015.

[78] Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. Checking app behavior against app descriptions. In *ICSE '14: Proceedings of the 2014 International Conference on Software Engineering*, pages 292–302. ACM Press, 2014.

[79] Michael Goul, Olivera Marjanovic, Susan Baxley, and Karen Vizecky. Managing the enterprise business intelligence app store: Sentiment analysis supported requirements engineering. In *Proceedings of the 2012 45th Hawaii International Conference on System Sciences*, HICSS '12, pages 4168–4177, 2012.

[80] Michael Grace, Wu Zhou, Xuxian Jiang, and Ahmad-Reza Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WISEC '12, pages 101–112. ACM, 2012.

[81] Michael Grace, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang. RiskRanker: Scalable and accurate zero-day Android malware detection. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, pages 281–294. ACM, 2012.

[82] Xiaodong Gu and Sunghun Kim. What parts of your apps are loved by users? In *In Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE'15)*, 2015.

[83] Latifa Guerrouj, Shams Azad, and Peter C. Rigby. The influence of App churn on App success and StackOverflow discussions. In *Proceedings of the 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 321–330. IEEE, 2015.

[84] Jiaping Gui, Stuart Mcilroy, Meiyappan Nagappan, and William G. J. Halfond. Truth in advertising: The hidden cost of mobile ads for software developers. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, pages 100–110. IEEE Press, 2015.

[85] Emitza Guzman, Omar Aly, and Bernd Bruegge. Retrieving diverse opinions from app reviews. In *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–10. IEEE, 2015.

[86] Emitza Guzman, Muhammad El-Haliby, and Bernd Bruegge. Ensemble methods for app review classification: An approach for software evolution (N). In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015*, pages 771–776, 2015.

[87] Emitza Guzman and Walid Maalej. How do users like this feature? a fine grained sentiment analysis of app reviews. In *22nd IEEE International Requirements Engineering Conference (RE'14)*, pages 153 – 162, 2014.

[88] Elizabeth Ha and Dietmar Wagner. Do Android users write about electric sheep? examining consumer reviews in Google Play. In *Consumer Communications and Networking Conference (CCNC), 2013 IEEE*, pages 149–157. IEEE, 2013.

[89] You Joung Ham and Hyung-Woo Lee. Detection of malicious Android mobile applications based on aggregated system call events. *International Journal of Computer and Communication Engineering*, 3(2):149–154, 2014.

[90] Shuai Hao, Bin Liu, Suman Nath, William G.J. Halfond, and Ramesh Govindan. PUMA: Programmable UI-Automation for Large-Scale Dynamic Analysis of Mobile Apps. In *Proceedings of the 12th International Conference on Mobile Systems, Applications, and Services (MobiSys'14)*, 2014.

[91] Mark Harman, Yue Jia, and Yuanyuan Zhang. App Store Mining and Analysis: MSR for App Stores. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (MSR'12)*, pages 108–111, 2012.

[92] Xiuqiang He, Wenyuan Dai, Guoxiang Cao, Ruiming Tang, Mingxuan Yuan, and Qiang Yang. Mining target users for online marketing based on app store data. In *2015 IEEE International Conference on Big Data*, pages 1043–1052. IEEE, 2015.

[93] Henning Heitkötter, Sebastian Hanschke, and Tim A. Majchrzak. Evaluating cross-platform development approaches for mobile applications. In *Web Information Systems and Technologies - 8th International Conference, WEBIST 2012, Porto, Portugal, April 18-21, 2012, Revised Selected Papers*, pages 120–138, 2012.

[94] Niels Henze and Susanne Boll. Release your app on sunday eve: Finding the best time to deploy apps. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI'11)*, pages 581–586, 2011.

[95] Tsung-Hsuan Ho, Daniel Dean, Xiaohui Gu, and William Enck. PREC: Practical root exploit containment for Android devices. In *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*, CODASPY '14, pages 187–198. ACM, 2014.

[96] Leonard Hoon, MiguelAngel Rodriguez-Garca, Rajesh Vasa, Rafael Valencia-Garca, and Jean-Guy Schneider. App reviews: Breaking the user and developer language barrier. In *Trends and Applications in Software Engineering*, volume 405 of *Advances in Intelligent Systems and Computing*, pages 223–233. Springer International Publishing, 2016.

[97] Leonard Hoon, Rajesh Vasa, Gloria Yoanita Martino, Jean-Guy Schneider, and Kon Mouzakis. Awesome!: Conveying satisfaction on the app store. In *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*, OzCHI '13, pages 229–232. ACM, 2013.

[98] Leonard Hoon, Rajesh Vasa, Jean-Guy Schneider, and John Grundy. An analysis of the mobile app review landscape: trends and implications. Technical report, Faculty of Information and Communication Technologies, Swinburne University of Technology, 2013.

[99] Leonard Hoon, Rajesh Vasa, Jean-Guy Schneider, and Kon Mouzakis. A preliminary analysis of vocabulary in mobile app user reviews. In *Proceedings of the 24th Australian Computer-Human Interaction Conference*, OzCHI '12, pages 245–248. ACM, 2012.

[100] Jianjun Huang, Zhichun Li, Xusheng Xiao, Zhenyu Wu, Kangjie Lu, Xiangyu Zhang, and Guofei Jiang. SUPOR: Precise and scalable sensitive user input detection for Android apps. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, pages 977–992. USENIX Association, 2015.

[101] Wei Huang, Yao Dong, Ana Milanova, and Julian Dolby. Scalable and precise taint analysis for Android. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ISSTA 2015, pages 106–117. ACM, 2015.

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

[103] Claudia Iacob, Rachel Harrison, and Shamal Faily. Online reviews as first class artifacts in mobile app development. In *Proceedings of the 5th International Conference on Mobile Computing, Applications, and Services. MobiCASE '13*, 2014.

[104] Claudia Iacob, Varsha Veerappa, and Rachel Harrison. What are you complaining about?: A study of online reviews of mobile applications. In *Proceedings of the 27th International BCS Human Computer Interaction Conference*, BCS-HCI '13, pages 29:1–29:6. British Computer Society, 2013.

[105] Sun-Young Ihm, Woong-Kee Loh, and Young-Ho Park. App analytic: A study on correlation analysis of app ranking data. *International Conference on Cloud and Green Computing (CGC)*, 0:561–563, 2013.

[106] Jinseong Jeon, Kristopher K. Micinski, Jeffrey A. Vaughan, Ari Fogel, Nikhilesh Reddy, Jeffrey S. Foster, and Todd Millstein. Dr. Android and mr. hide: Fine-grained permissions in Android applications. In *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '12, pages 3–14. ACM, 2012.

[107] He Jiang, Hongjing Ma, Zhilei Ren, Jingxuan Zhang, and Xiaochen Li. What makes a good app description? In *Proceedings of the 6th Asia-Pacific Symposium on Internetware on Internetware*, INTERNETWARE 2014, pages 45–53. ACM, 2014.

[108] Yiming Jing, Gail-Joon Ahn, Ziming Zhao, and Hongxin Hu. Riskmon: Continuous and automated risk assessment of mobile applications. In *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*, CODASPY '14, pages 99–110. ACM, 2014.

[109] Yohan Jo and Alice H. Oh. Aspect and sentiment unification model for online review analysis. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 815–824. ACM, 2011.

[110] Mona Erfani Joorabchi, Mohamed Ali, and Ali Mesbah. Detecting inconsistencies in multi-platform mobile apps. In *Proceedings of the 26th IEEE International Symposium on Software Reliability Engineering*, ISSRE '15, 2015.

[111] Euy-Young Jung, Chulwoo Baek, and Jeong-Dong Lee. Product survival analysis for the App Store. *Marketing Letters*, 23(4):929–941, 2012.

[112] Haliyana Khalid, Meiyappan Nagappan, and Asif Hassan. Examining the relationship between FindBugs warnings and end user ratings: A case study on 10,000 Android apps. *IEEE Transactions on Software Engineering*, 2015. preprint.

[113] Hammad Khalid. On identifying user complaints of iOS apps. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 1474–1476. IEEE Press, 2013.

[114] Hammad Khalid, Meiyappan Nagappan, Emad Shihab, and Ahmed E. Hassan. Prioritizing the devices to test your app on: A case study of Android game apps. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 610–620. ACM, 2014.

[115] Hammad Khalid, Emad Shihab, Meiyappan Nagappan, and Ahmed E. Hassan. What do mobile app users complain about? *IEEE Software*, 32(3):70–77, 2015.

[116] Mubasher Khalid, Muhammad Asif, and Usman Shehzaib. Towards improving the quality of mobile app reviews. *International Journal of Information Technology and Computer Science (IJITCS)*, 7(10):35, 2015.

[117] Mubasher Khalid, Usman Shehzaib, and Muhammad Asif. A case of mobile app reviews as a crowdsource. *International Journal of Information Engineering and Electronic Business (IJIEEB)*, 7(5):39, 2015.

[118] Kobra Khanmohammadi, Mohammad Reza Rejali, and Abdelwahab Hamou-Lhadj. Understanding the service life cycle of Android apps: An exploratory study. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '15, pages 81–86. ACM, 2015.

[119] Daeyoung Kim, Amruta Gokhale, Vinod Ganapathy, and Abhinav Srivastava. Detecting plagiarized mobile apps using API birthmarks. *Automated Software Engineering*, pages 1–28, 2015.

[120] Jieun Kim, Yongtae Park, Chulhyun Kim, and Hakyeon Lee. Mobile application service networks: Apple's App Store. *Service Business*, 8(1):1–27, 2014.

[121] Daniel E. Krutz, Mehdi Mirakhorli, Samuel A. Malachowsky, Andres Ruiz, Jacob Peterson, Andrew Filipski, and Jared Smith. A dataset of open-source Android applications. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, pages 522–525. IEEE Press, 2015.

[122] Nathaniel Lageman, Mark Lindsey, and William Glodek. Detecting malicious Android applications from runtime behavior. In *Military Communications Conference, MILCOM 2015-2015 IEEE*, pages 324–329. IEEE, 2015.

[123] Gunwoong Lee and T. S. Raghu. Product portfolio and mobile apps success: Evidence from App Store market. In Vallabh Sambamurthy and Mohan Tanniru, editors, *Proceedings of the 17th Americas Conference on Information Systems AMCIS '11*. Association for Information Systems, 2011.

[124] Gunwoong Lee and TS Raghu. Determinants of mobile apps' success: Evidence from the app store market. *Journal of Management Information Systems*, 31(2):133–170, 2014.

[125] Li Li, Alexandre Bartel, Tegawendé François D Assise Bissyande, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick McDaniel. Iccta: detecting inter-component privacy leaks in Android apps. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE '15)*, 2015.

[126] Ting-Peng Liang, Xin Li, Chin-Tsung Yang, and Mengyue Wang. What in consumer reviews affects the sales of mobile apps: A multifacet sentiment analysis approach. *International Journal of Electronic Commerce*, 20(2):236–260, 2015.

[127] S Lim, Peter Bentley, Natalie Kanakam, Fuyuki Ishikawa, and Shinichi Honiden. Investigating country differences in mobile app user behavior and challenges for software engineering. *IEEE Transactions on Software Engineering (TSE)*, 2015.

[128] Soo Ling Lim and Peter J. Bentley. App epidemics: Modelling the effects of publicity in a mobile app ecosystem. In *Artificial Life 13: Proceedings of the Thirteenth International Conference on the Simulation and Synthesis of Living Systems (ALIFE)*, 2012.

[129] Soo Ling Lim and Peter J Bentley. How to be a successful app developer: lessons from the simulation of an app ecosystem. *ACM SIGEVOlution*, 6(1):2–15, 2012.

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

[131] Soo Ling Lim, Peter J Bentley, and Fuyuki Ishikawa. The effects of developer dynamics on fitness in an evolutionary ecosystem model of the App Store. *IEEE Transactions on Evolutionary Computation (TEVC)*, PP, 2015.

[132] Jialiu Lin, Shahriyar Amini, Jason I. Hong, Norman Sadeh, Janne Lindqvist, and Joy Zhang. Expectation and purpose: Understanding users' mental models of mobile app privacy through crowdsourcing. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 501–510. ACM, 2012.

[133] Jovian Lin, Kazunari Sugiyama, Min-Yen Kan, and Tat-Seng Chua. Addressing cold-start in app recommendation: Latent user models constructed from twitter followers. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, pages 283–292. ACM, 2013.

[134] Jovian Lin, Kazunari Sugiyama, Min-Yen Kan, and Tat-Seng Chua. New and improved: Modeling versions to improve app recommendation. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '14, pages 647–656. ACM, 2014.

[135] Mario Linares-Vásquez. Supporting evolution and maintenance of Android apps. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 714–717. ACM, 2014.

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

[137] Mario Linares-Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Rocco Oliveto, Massimiliano Di Penta, and Denys Poshyvanyk. Mining energy-greedy API usage patterns in Android apps: An empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 2–11. ACM, 2014.

[138] Mario Linares-Vásquez, Andrew Holtzhauer, Carlos Bernal-Cárdenas, and Denys Poshyvanyk. Revisiting Android reuse studies in the context of code obfuscation and library usages. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 242–251. ACM, 2014.

[139] Bin Liu, Deguang Kong, Lei Cen, Neil Zhenqiang Gong, Hongxia Jin, and Hui Xiong. Personalized mobile app recommendation: Reconciling app functionality and user privacy preference. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, WSDM '15, pages 315–324. ACM, 2015.

[140] Bin Liu, Bin Liu, Hongxia Jin, and Ramesh Govindan. Efficient privilege de-escalation for ad libraries in mobile apps. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '15, pages 89–103. ACM, 2015.

[141] Bin Liu, Suman Nath, Ramesh Govindan, and Jie Liu. Decaf: Detecting and characterizing ad fraud in mobile apps. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 57–70. USENIX Association, 2014.

[142] Xing Liu and Jiqiang Liu. A two-layered permission-based Android malware detection scheme. In *Proceedings of the 2014 2Nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, MOBILECLOUD '14, pages 142–148. IEEE Computer Society, 2014.

[143] Siqi Ma, Shaowei Wang, David Lo, Robert Huijie Deng, and Cong Sun. Active semi-supervised approach for checking app behavior against its description. In *39th IEEE Annual Computer Software and Applications Conference, COMPSAC 2015, Taichung, Taiwan, July 1-5, 2015. Volume 2*, pages 179–184, 2015.

[144] Walid Maalej and Hadeer Nabil. Bug report, feature request, or simply praise? on automatically classifying app reviews. *Requirements Engineering (RE15)*, 2015.

[145] Ivano Malavolta, Stefano Ruberto, Tommaso Soru, and Valerio Terragni. End users' perception of hybrid mobile apps in the Google Play store. In *Proceedings of the 4th International Conference on Mobile Services (MS)*. IEEE, 2015.

[146] Ivano Malavolta, Stefano Ruberto, Valerio Terragni, and Tommaso Soru. Hybrid mobile apps in the Google Play store: an exploratory investigation. In *Proceedings of the 2nd ACM International Conference on Mobile Software Engineering and Systems*. ACM, 2015.

[147] Larry M. Manevitz and Malik Yousef. One-class svms for document classification. *J. Mach. Learn. Res.*, 2:139–154, 2002.

[148] marketsandmarkets.com. World Mobile Applications Market - Advanced Technologies, Global Forecast, 2010. Accessed 29th April 2014.

[149] William Martin, Mark Harman, Yue Jia, Federica Sarro, and Yuanyuan Zhang. The app sampling problem for app store mining. In *Proceedings of the 12th IEEE Working Conference on Mining Software Repositories*, MSR'15, 2015.

[150] William Martin, Federica Sarro, and Mark Harman. Causal impact analysis applied to app releases in Google Play and Windows Phone Store. Technical report, University College London, 2015.

[151] Tyler McDonnell, Baishakhi Ray, and Miryung Kim. An empirical study of API stability and adoption in the Android ecosystem. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance*, ICSM '13, pages 70–79. IEEE Computer Society, 2013.

[152] Stuart McIlroy, Nasir Ali, and Ahmed E Hassan. Fresh apps: an empirical study of frequently-updated mobile apps in the google play store. *Empirical Software Engineering*, pages 1–25, 2015.

[153] Stuart McIlroy, Nasir Ali, Hammad Khalid, and Ahmed E. Hassan. Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews. *Empirical Software Engineering*, pages 1–40, 2015.

[154] Stuart McIlroy, Weiyi Shang, Nasir Ali, and Ahmed Hassan. Is it worth responding to reviews? a case study of the top free apps in the Google Play store. *IEEE Software*, PP, 2015.

[155] David Mimno, Hanna Wallach, Edmund Talley, Miriam Leenders, and Andrew McCallum. Optimizing Semantic Coherence in Topic Models. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 262–272. Association for Computational Linguistics, 2011.

[156] Roberto Minelli and Michele Lanza. Samoa — a visual software analytics platform for mobile applications. In *Proceedings of ICSM 2013 (29th International Conference on Software Maintenance)*, pages 476–479. IEEE CS Press, 2013.

[157] Roberto Minelli and Michele Lanza. Software analytics for mobile applications–insights & lessons learned. *2013 15th European Conference on Software Maintenance and Reengineering*, 0:144–153, 2013.

[158] Israel J. Mojica, Meiyappan Nagappan, Bram Adams, Thorsten Berger, Steffen Dienst, and Ahmed E. Hassan. An examination of the current rating system used in mobile app stores. *IEEE Software*, 2015. To appear.

[159] Israel J. Mojica Ruiz, Meiyappan Nagappan, Bram Adams, Thorsten Berger, Steffen Dienst, and Ahmed E. Hassan. On ad library updates in Android apps. *IEEE Software, Accepted on 14 May 2014*, 2014.

[160] Shahab Mokarizadeh, Mohammad Tafiqur Rahman, and Mihhail Matskin. Mining and analysis of apps in Google Play. In *Web Information Systems and Technologies - 9th International Conference, WEBIST '13*, 2013.

[161] João Eduardo Montandon, Hudson Borges, Daniel Felix, and Marco Tulio Valente. Documenting APIs with examples: Lessons learned with the APIMiner platform. In *20th Working Conference on Reverse Engineering (WCRE)*, pages 401–408. IEEE, 2013.

[162] Kevin Moran, Mario Linares-Vásquez, Carlos Bernal-Cárdenas, and Denys Poshyvanyk. Autocompleting bug reports for Android applications. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 673–686. ACM, 2015.

[163] Patrick Mutchler, Adam Doupé, John Mitchell, Christopher Kruegel, and Giovanni Vigna. A Large-Scale Study of Mobile Web App Security. In *Proceedings of the Mobile Security Technologies Workshop (MoST)*, 2015.

[164] Andreas Mller, Stefan Diewald, Luis Roalter, Technische Universitt Mnchen, Florian Michahelles, and Matthias Kranz. Update behavior in app markets and security implications: A case study in Google Play. In *In Proc. of the 3rd Intl. Workshop on Research in the Large. Held in Conjunction with Mobile HCI*, pages 3–6, 2012.

[165] Meiyappan Nagappan and Emad Shihab. Future trends in software engineering research for mobile apps. In *23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering SANER '16*, 2016.

[166] Maleknaz Nayebi and Guenther Ruhe. Trade-off service portfolio planning–a case study on mining the Android app market. *PeerJ PrePrints*, 2015.

[167] Yi Ying Ng, Hucheng Zhou, Zhiyuan Ji, Huan Luo, and Yuan Dong. Which Android app store can be trusted in China? In *Proceedings of the 2014 IEEE 38th Annual Computer Software and Applications Conference*, COMPSAC '14, pages 509–518. IEEE Computer Society, 2014.

[168] Jeungmin Oh, Daehoon Kim, Uichin Lee, Jae-Gil Lee, and Junehwa Song. Facilitating developer-user interactions with mobile app review digests. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '13, pages 1809–1814. ACM, 2013.

[169] Dennis Pagano and Walid Maalej. User feedback in the appstore: An empirical study. In *Proceedings of the 21st. IEEE International Requirements Engineering Conference (RE) '13*. IEEE, 2013.

[170] Fabio Palomba, Mario Linares-Vásquez, Gabriele Bavota, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. User reviews matter! tracking crowdsourced reviews to support evolution of successful apps. In *31st International Conference on Software Maintenance and Evolution (ICSME) '15*, 2015.

[171] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. Whyper: Towards automating risk assessment of mobile applications. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 527–542. USENIX Association, 2013.

[172] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Aaron Visaggio, Gerardo Canfora, and Harald Gall. How can i improve my app? classifying user reviews for software maintenance and evolution. *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, 2015.

[173] Dae Hoon Park, Mengwen Liu, ChengXiang Zhai, and Haohong Wang. Leveraging user reviews to improve accuracy for mobile app retrieval. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 533–542. ACM, 2015.

[174] Naser Peiravian and Xingquan Zhu. Machine learning for Android malware detection using permission and API calls. In *Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, ICTAI '13, pages 300–305. IEEE Computer Society, 2013.

[175] Hao Peng, Christopher S. Gates, Bhaskar Pratim Sarma, Ninghui Li, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Using probabilistic generative models for ranking risks of Android apps. In *ACM Conference on Computer and Communications Security*, pages 241–252. ACM, 2012.

[176] Thanasis Petsas, Antonis Papadogiannakis, Michalis Polychronakis, Evangelos P. Markatos, and Thomas Karagiannis. Rise of the planet of the apps: A systematic study of the mobile app ecosystem. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, pages 277–290. ACM, 2013.

[177] Rahul Potharaju, Andrew Newell, Cristina Nita-Rotaru, and Xiangyu Zhang. Plagiarizing smartphone applications: Attack strategies and defense techniques. In *Proceedings of the 4th International Conference on Engineering Secure Software and Systems (ESSoS'12)*, pages 106–120. Springer-Verlag, 2012.

[178] T Preuss. *Mobile Applications, Functional Analysis, and the Customer Experience*. Auerbach Publications, 2012.

[179] T Preuss. Mobile applications, function points and cost estimating. In *International Cost Estimation & Analysis Association Conference*, 2013.

[180] Zhengyang Qu, Vaibhav Rastogi, Xinyi Zhang, Yan Chen, Tiantian Zhu, and Zhong Chen. AutoCog: Measuring the description-to-permission fidelity in Android applications. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1354–1365. ACM, 2014.

[181] Vaibhav Rastogi, Yan Chen, and William Enck. Appsplayground: automatic security analysis of smartphone applications. In *Proceedings of the third ACM conference on Data and application security and privacy*, pages 209–220. ACM, 2013.

[182] Lenin Ravindranath, Suman Nath, Jitendra Padhye, and Hari Balakrishnan. Automatic and scalable fault detection for mobile applications. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '14, pages 190–203. ACM, 2014.

[183] Alexander-Derek Rein and Jürgen Münch. Feature prioritization based on mock-purchase: A mobile case study. In *Proceedings of the Lean Enterprise Software and Systems Conference (LESS 2013)*, pages 165–179. Springer, 2013.

[184] Stephen Robertson. Understanding inverse document frequency: On theoretical arguments for idf. *Journal of Documentation*, 60:2004, 2004.

[185] Sankardas Roy, Jordan DeLoach, Yuping Li, Nic Herndon, Doina Caragea, Xinming Ou, Venkatesh Prasad Ranganath, Hongmin Li, and Nicolais Guevara. Experimental study with real-world data for Android app security analysis using machine learning. In *Proceedings of the 31st Annual Computer Security Applications Conference*, ACSAC 2015, pages 81–90. ACM, 2015.

[186] Israel J. Mojica Ruiz, Bram Adams, Meiyappan Nagappan, Steffen Dienst, Thorsten Berger, and Ahmed E. Hassan. A large scale empirical study on software reuse in mobile apps. *IEEE Software*, 31(2):78–86, 2014.

[187] Israel J. Mojica Ruiz, Meiyappan Nagappan, Bram Adams, Thorsten Berger, Steffen Dienst, and Ahmed E. Hassan. Impact of ad libraries on ratings of Android mobile apps. *IEEE Software*, 31(6):86–92, 2014.

[188] Israel J. Mojica Ruiz, Meiyappan Nagappan, Bram Adams, and Ahmed E. Hassan. Understanding reuse in the Android market. In *ICPC*, pages 113–122, 2012.

[189] Alireza Sahami Shirazi, Niels Henze, Albrecht Schmidt, Robin Goldberg, Benjamin Schmidt, and Hansjörg Schmauder. Insights into layout patterns of mobile user interfaces by an automatic analysis of Android apps. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '13, pages 275–284. ACM, 2013.

[190] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, and Pablo Garcia Bringas. On the automatic categorisation of Android applications. In *CCNC'12*, pages 149–153, 2012.

[191] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, Javier Nieves, Pablo G. Bringas, and Gonzalo Álvarez Marañón. MAMA: Manifest analysis for malware detection in Android. *Cybernetics and Systems - Intelligent Network Security and Survivability*, 44(6-7):469–488, 2013.

[192] Borja Sanz, Igor Santos, Javier Nieves, Carlos Laorden, Inigo Alonso-Gonzalez, and Pablo G Bringas. Mads: malicious Android applications detection through string analysis. In *Network and System Security*, pages 178–191. Springer, 2013.

[193] Borja Sanz, Igor Santos, Xabier Ugarte-Pedrero, Carlos Laorden, Javier Nieves, and Pablo Garcia Bringas. Instance-based anomaly method for Android malware detection. In *SECRYPT 2013 - Proceedings of the 10th International Conference on Security and Cryptography, Reykjavík, Iceland, 29-31 July, 2013*, pages 387–394, 2013.

[194] Federica Sarro. The UCLAppA repository: A repository of research articles on mobile software engineering and app store analysis.

[195] Federica Sarro, Afnan A. Al-Subaihin, Mark Harman, Yue Jia, William Martin, and Yuanyuan Zhang. Feature lifecycles as they spread, migrate, remain and die in app stores. In *Proceedings of the Requirements Engineering Conference, 23rd IEEE International (RE'15)*. IEEE, 2015.

[196] Julian Schütte, Rafael Fedler, and Dennis Titze. ConDroid: Targeted dynamic analysis of Android applications. In *29th IEEE International Conference on Advanced Information Networking and Applications, AINA 2015, Gwangju, South Korea, March 24-27, 2015*, pages 571–578, 2015.

[197] Suranga Seneviratne, Harini Kolamunna, and Aruna Seneviratne. A measurement study of tracking in paid mobile applications. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '15, pages 7:1–7:6. ACM, 2015.

[198] Suranga Seneviratne, Aruna Seneviratne, Mohamed Ali Kaafar, Anirban Mahanti, and Prasant Mohapatra. Early detection of spam mobile apps. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 949–959. International World Wide Web Conferences Steering Committee, 2015.

[199] G Sethumadhavan. Sizing Android mobile applications. In *6th IFPUG International Software Measurement and Analysis Conference (ISMA)*, 2011.

[200] Asaf Shabtai, Yuval Fledel, and Yuval Elovici. Automated static code analysis for classifying Android applications using machine learning. In *Proceedings of the 2010 International Conference on Computational Intelligence and Security*, CIS '10, pages 329–333. IEEE Computer Society, 2010.

[201] Chetan Sharma. Sizing up the global mobile apps market. *Report, Chetan Sharma Consulting, Issaquah, WA*, 2010.

[202] Carly Shuler. iLearnII; an analysis of the education category of the iTunes App Store. *The Joan Ganz Cooney Center at Sesame Workshop*, 2012.

[203] Ryan Stevens, Jonathan Ganz, Vladimir Filkov, Premkumar Devanbu, and Hao Chen. Asking for (and about) permissions used by Android apps. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*, pages 31–40. IEEE Press, 2013.

[204] Zorana Svedic. *The Effect of Informational Signals on Mobile Apps Sales Ranks Across the Globe*. PhD thesis, SIMON FRASER UNIVERSITY, 2015.

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

[206] Mark. D. Syer, Meiyappan Nagappan, Bram Adams, and Ahmed E. Hassan. Studying the relationship between source code quality and mobile platform dependence. *Software Quality Journal*, 23(3):485–508, 2015.

[207] Mark D. Syer, Meiyappan Nagappan, Ahmed E. Hassan, and Bram Adams. Revisiting prior empirical findings for mobile apps: An empirical case study on the 15 most popular open-source Android apps. In *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '13, pages 283–297. IBM Corp., 2013.

[208] Joshua Tan, Khanh Nguyen, Michael Theodorides, Heidi Negrón-Arroyo, Christopher Thompson, Serge Egelman, and David Wagner. The effect of developer-specified explanations for permission requests on smartphone user behavior. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 91–100. ACM, 2014.

[209] Peter Teufl, Michaela Ferk, Andreas Fitzek, Daniel Hein, Stefan Kraxberger, and Clemens Orthacker. Malware detection by applying knowledge discovery processes to application metadata on the Android Market (Google Play). *Security and Communication Networks*, 2013.

[210] Peter Teufl, Stefan Kraxberger, Clemens Orthacker, Gnther Lackner, Michael Gissing, Alexander Marsalek, Johannes Leibetseder, and Oliver Prevenhueber. Android Market analysis with activation patterns. In *Security and Privacy in Mobile Information and Communication Systems (MobiSec)*, volume 94 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 1–12. Springer Berlin Heidelberg, 2012.

[211] Yuan Tian, Meiyappan Nagappan, David Lo, and Ahmed E Hassan. What are the characteristics of high-rated apps? a case study on free Android applications. In *31st International Conference on Software Maintenance and Evolution (ICSME)*, pages 1–10, 2015.

[212] Yong-Xin Tong, Jieying She, and Lei Chen. Towards better understanding of app functions. *Journal of Computer Science and Technology*, 30(5):1130–1140, 2015.

[213] Svitlana Vakulenko, Oliver Müller, and Jan vom Brocke. Enriching iTunes App Store categories via topic modeling. In *International Conference on Information Systems (ICIS'14)*, 2014.

[214] Harold van Heeringen and Edwin Van Gorp. Measure the functional size of a mobile app: Using the cosmic functional size measurement method. In *Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2014 Joint Conference of the International Workshop on*, pages 11–16. IEEE, 2014.

[215] Rajesh Vasa, Leonard Hoon, Kon Mouzakis, and Akihiro Noguchi. A preliminary analysis of mobile app user reviews. In *Proceedings of the 24th Australian Computer-Human Interaction Conference*, OzCHI '12, pages 241–244. ACM, 2012.

[216] Nicolas Viennot. GitHub - nviennot/playdrone: Google Play Crawler, 2014.

[217] Nicolas Viennot, Edward Garcia, and Jason Nieh. A measurement study of Google Play. In *The 2014 ACM international conference on Measurement and modeling of computer systems*, SIGMETRICS '14, pages 221–233. ACM, 2014.

[218] Luigi Vigneri, Jaideep Chandrashekar, Ioannis Pefkianakis, and Olivier Heen. Taming the Android appstore: Lightweight characterization of Android applications. *CoRR*, abs/1504.06093, 2015.

[219] Lorenzo Villarroel Pérez. Mining mobile apps reviews to support release planning. Master's thesis, ETSI_Informatica, 2015.

[220] VirusShare. Virusshare.com, 2011.

[221] Vision Mobile. Developer Economics 2013: The tools report, 2013.

[222] Vision Mobile. Developer Economics Q1 2015: State of the Developer Nation, 2015.

[223] Alexander von Rhein, Thorsten Berger, Niklas Schalck Johansson, Mikael Mark Hardø, and Sven Apel. Lifting inter-app data-flow analysis to large app sets. Technical report, Fakultät für Informatik und Mathematik, Universität Passau, 2015.

[224] Phong Minh Vu, Tam The Nguyen, Hung Viet Pham, and Tung Thanh Nguyen. Mining user opinions in mobile app reviews: A keyword-based approach. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE '15)*, pages 749–759. IEEE, 2015.

[225] Phong Minh Vu, Hung Viet Pham, Tam The Nguyen, and Tung Thanh Nguyen. Tool support for analyzing mobile app reviews. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE '15)*, pages 789–794. IEEE, 2015.

[226] Timo Johann Walid Maalej, Maleknaz Nayebi and Gunther Ruhe. Toward data-driven requirements engineering. *IEEE Software Jan/Feb 2016: Special Issue on the Future of Software Engineering*, 2016. To appear.

[227] Mian Wan, Yuchen Jin, Ding Li, and William GJ Halfond. Detecting display energy hotspots in Android apps. In *Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on*, pages 1–10. IEEE, 2015.

[228] Haoyu Wang, Yao Guo, Ziang Ma, and Xiangqun Chen. WuKong: A scalable and accurate two-phase approach to Android app clone detection. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ISSTA 2015, pages 71–82. ACM, 2015.

[229] Haoyu Wang, Jason Hong, and Yao Guo. Using text mining to infer the purpose of permission use in mobile apps. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '15, pages 1107–1118. ACM, 2015.

[230] Wei Wang, Xing Wang, Dawei Feng, Jiqiang Liu, Zhen Han, and Xiangliang Zhang. Exploring permission-induced risk in Android applications for malicious application detection. *IEEE Transactions on Information Forensics and Security*, pages 1869–1882, 2014.

[231] Yang Wang, Jun Zheng, Chen Sun, and Srinivas Mukkamala. Quantitative security risk assessment of Android permissions and applications. In *Proceedings of the 27th International Conference on Data and Applications Security and Privacy XXVII*, DBSec'13, pages 226–241. Springer-Verlag, 2013.

[232] Megumi Wano and Jun Iio. Relationship between reviews at app store and the categories for software. In *Proceedings of the 2014 17th International Conference on Network-Based Information Systems*, NBIS '14, pages 580–583. IEEE Computer Society, 2014.

[233] Takuya Watanabe, Mitsuaki Akiyama, Tetsuya Sakai, Hironori Washizaki, and Tatsuya Mori. Understanding the inconsistencies between text descriptions and the use of privacy-sensitive resources of mobile apps. In *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*. USENIX Association, 2015.

[234] Shi Wenxuan and Yin Airu. Interoperability-enriched app recommendation. In *Data Mining Workshop (ICDMW), 2014 IEEE International Conference on*, pages 1242–1245. IEEE, 2014.

[235] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, pages 38:1–38:10. ACM, 2014.

[236] Zhen Xie and Sencun Zhu. AppWatcher: Unveiling the underground market of trading mobile app reviews. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '15, pages 10:1–10:11. ACM, 2015.

[237] Wei Xu, Fangfang Zhang, and Sencun Zhu. Permlyzer: Analyzing permission usage in Android applications. In *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, pages 400–410. IEEE, 2013.

[238] Wei Yang, Xusheng Xiao, Benjamin Andow, Sihan Li, Tao Xie, and William Enck. AppContext: Differentiating malicious and benign mobile app behavior under contexts. In *2015 International Conference on Software Engineering (ICSE)*, 2015.

[239] Yingyuan Yang, Jinyuan Stella Sun, and Michael W. Berry. APPIC: Finding the hidden scene behind description files for Android apps. Technical report, Dept. of Electrical Engineering and Computer Science University of Tennessee, 2014.

[240] Peifeng Yin, Ping Luo, Wang-Chien Lee, and Min Wang. App recommendation: A contest between satisfaction and temptation. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13, pages 395–404. ACM, 2013.

[241] Fangfang Zhang, Heqing Huang, Sencun Zhu, Dinghao Wu, and Peng Liu. Viewdroid: Towards obfuscation-resilient mobile application repackaging detection. In *Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless & Mobile Networks*, WiSec '14, pages 25–36. ACM, 2014.

[242] Mu Zhang, Yue Duan, Qian Feng, and Heng Yin. Towards automatic generation of security-centric descriptions for Android apps. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 518–529. ACM, 2015.

[243] Nan Zhong and Florian Michahelles. Google Play is not a long tail market: An empirical analysis of app adoption on the Google Play app market. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 499–504. ACM, 2013.

[244] Yajin Zhou and Xuxian Jiang. Dissecting Android malware: Characterization and evolution. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 95–109. IEEE, 2012.

[245] Yajin Zhou, Lei Wu, Zhi Wang, and Xuxian Jiang. Harvesting developer credentials in Android apps. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '15, pages 23:1–23:12. ACM, 2015.

[246] Hengliang Zhu, Cong Liu, Yan Ge, Hui Xiong, and Eason Chen. Popularity modeling for mobile apps: A sequential approach. *IEEE Transactions on Cybernetics*, 45(7):1303–1314, 2014.

[247] Hengshu Zhu, Huanhuan Cao, Enhong Chen, Hui Xiong, and Jilei Tian. Exploiting enriched contextual information for mobile app classification. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1617–1621. ACM, 2012.

[248] Hengshu Zhu, Enhong Chen, Hui Xiong, Huanhuan Cao, and Jilei Tian. Mobile app classification with enriched contextual information. *IEEE Transactions on Mobile Computing*, 13(7):1550–1563, 2014.

[249] Hengshu Zhu, Hui Xiong, Yong Ge, and Enhong Chen. Ranking fraud detection for mobile apps: A holistic view. In *Proceedings of the 22Nd ACM International Conference on Conference on Information & Knowledge Management*, CIKM '13, pages 619–628. ACM, 2013.

[250] Hengshu Zhu, Hui Xiong, Yong Ge, and Enhong Chen. Mobile app recommendations with security and privacy awareness. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 951–960. ACM, 2014.

[251] Hengshu Zhu, Hui Xiong, Yong Ge, and Enhong Chen. Discovery of ranking fraud for mobile apps. *Knowledge and Data Engineering, IEEE Transactions on*, 27(1):74–87, 2015.

[252] Jiawei Zhu, Zhi Guan, Yang Yang, Liangwen Yu, Huiping Sun, and Zhong Chen. Permission-based abnormal application detection for Android. In *Proceedings of the 14th International Conference on Information and Communications Security*, ICICS'12, pages 228–239. Springer-Verlag, 2012.