LARGE-SCALE EMPIRICAL STUDIES OF MOBILE APPS

by

ISRAEL JESUS MOJICA RUIZ

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

Queen's University
Kingston, Ontario, Canada
August 2013

Abstract

Mobile apps (or apps) are software applications developed to run on mobile devices such as smartphones and tablets, among other devices. The number of apps has grown tremendously since Apple opened the first app store in 2008. For example, in March of 2009 the Google Play app store (formerly known as Android Market) had only 2,300 apps, and by mid of 2013 there were more than 800,000 apps. Given the accelerated rate of growth in the number of apps, new software engineering challenges have emerged in order to help ease the software development practices of app developers. In this thesis we examine three examples of these challenges, namely code reuse in mobile apps, app ratings, and the use of ad libraries within apps. We carry out our case studies on thousands of Android apps from the Google Play market.

We find that code reuse in mobile apps is considerably higher than in desktop/server apps. However, identical copies of mobile apps are rare. We find that the current ratings system is not able to capture the dynamics of the evolving nature of apps. Thus, we were able to show the need for a more elaborate rating system for the apps. Finally, we observe that a considerable number of free-to-download apps are dependant on ads for their revenue. Our findings suggest that "ad maintenance" is a tough challenge that developers of mobile apps have to face.

Co-Authorship

Earlier versions of the work in the thesis were published as listed below:

1. *Understanding Reuse in the Android Market (Chap. 3)*

   Israel J. Mojica, Meiyappan Nagappan, Bram Adams, Ahmed E. Hassan, in Proceedings of the 20th IEEE International Conference on Program Comprehension (ICPC 2012). Passau, Germany. June 11-13, 2012. (Acceptance rate: 21/51=41%).

   My contributions: drafting the research plan, gathering and analyzing the data, writing the manuscript and presenting the paper.

2. *A Large-scale Empirical Study on User Ratings of Mobile Apps (Chap. 4)*

   Israel J. Mojica, Meiyappan Nagappan, Steffen Dienst, Thorsten Berger, Bram Adams, Ahmed E. Hassan.

   My contributions: drafting the research plan, analyzing the data, writing the manuscripts.

3. *A Large-scale Empirical Study on Ad Library Maintenance of Mobile Apps (Chap. 5)*

   Israel J. Mojica, Meiyappan Nagappan, Steffen Dienst, Thorsten Berger, Bram Adams, Ahmed E. Hassan.

   My contributions: drafting the research plan, analyzing the data, writing the manuscripts.

Acknowledgments

First and foremost, I thank God for everything. He has always been there for me, giving me this wonderful life, and supporting me in the challenging moments.

I would like to thank my supervisor Dr. Ahmed E. Hassan for his guidance throughout my Master. Ahmed trusted in me before knowing me, and opened the doors of an amazing lab where I met wonderful, smart, and committed people. Ahmed, I will not forget the opportunity you gave me to freely explore my ideas.

I would also like to thank my committee members, Dr. Patrick Martin and Dr. Saeed Gazor for taking time out to read my thesis. Their insightful comments have enhanced this thesis.

I am deeply grateful to my two collaborators and mentors along my Master: Dr. Bram Adams and Dr. Meiyappan Nagappan. I definitely put their patience to the test during the last two years!

I thank to Dr. Steffen Dienst and Dr. Thorsten Berger who kindly shared part of their work for the research presented in this thesis.

I thank to my family of researchers in the Software Analysis and Intelligence Lab (SAIL), who inspired me through their passion and commitment. To the amazing and beautiful Queen's University and the School of Computing for providing a fantastic environment for studying.

I wish thank each of the developers that developed the different open source or free applications that I used to build and run my experiments, and write this thesis.

To my parents who inspired me with their hard work and who gave me so much more than I deserve.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Mobile apps, better known as *apps*, are applications developed to run on mobile devices such as smartphones and tablets, among others. Typically, these apps are publicly available in online app stores such as the Amazon Appstore for Android (Amazon, 2011), the Apple's App Store for iOS (Apple, 2008*a*), the Google Play Store for Android (Google, 2008), the Microsoft's Windows Phone Store (Microsoft, 2010), or the Blackberry App World (Blackberry, 2009). The following statics demonstrate the tremendous success of these app stores:

1. **Number of apps:** By mid of 2013, Amazon Appstore had 75,000 apps (Koetsier, 2013), Apple's App store and Google Play Store had more than 800,000 apps each, Windows Phone Store had more than 145,000 apps, and BlackBerry App World had more than 120,000 apps (Canalys, 2013). These numbers have been increasing since Apple opened the first App Store in 2008 (Apple, 2008*b*).

2. **Number of developers:** Only two years after the first app store opened, in 2010, there were more than 50,000 developers creating mobile apps for the Android or iOS platforms (O'Neill, 2010).

3. **Number of downloads:** In 2009 there were 7 billion apps downloaded, and it is estimated that 50 billion apps (across all platforms) were downloaded in 2012 (ABI, 2012).

4. **Revenue:** The app markets combined are estimated to generate revenues of $74 U.S. billion by 2017 (Columbus, 2013).

Recent studies have analyzed the practice of app development across different platforms (Syer et al., 2011), the software economics behind the app stores (Chen, 2009), and the differences among apps and traditional software (Minelli and Lanza, 2013; Syer, 2013). Nevertheless, we believe that mobile apps bring a number of major and novel software engineering challenges.

1.1 Research Statement

New software engineering challenges have emerged in the current context of mobile apps. App developers, in order to compete in a massive market of apps, must take into consideration traditional aspects such as code reuse and novel aspects such as the approval of their users, and managing advertisement libraries embedded in their apps. In depth empirical studies are needed in order to shape and direct future software engineering research for mobile apps.

1.2 Thesis Overview

We give a brief overview of the three studies, which were all conducted on the Android apps from the Google Play market since it is one of the most active app markets today.

1. *Understanding Reuse in the Google Play App Store (Chap. 3)*

   Traditional software engineering practices such as code reuse has been recognized as a technique to ease the production of software since 1968 (McIlroy, 1968). Hence, we examine the practice of software reuse within the context of mobile apps.

   We found that app developers are re-using a substantial number of classes from third party developers (library providers) by inheritance or class reuse. On average 61% of all classes in each studied category occur in two or more apps. Additionally, we were even able to identify apps that are completely reused by another app (*i.e.*, identical code copies).

2. *A Large-scale Empirical Study on User Ratings of Mobile Apps (Chap. 4)*

   Through app stores, app-users can rate an app from 1 up to 5 stars. The different ratings of an app across its different versions can be averaged and consolidated into a global-rating. App stores, commonly, display this global-rating to inform potential app users about the quality of the app perceived by other app users (Android, 2013*b*). This global-rating has a high correlation with download counts (Harman et al., 2012). Hence, app developers have to take into consideration how users rate their apps.

   Our most notable findings are that *most app stores do not encourage app developers to improve the quality of their apps once they have been rated by a high number of raters*. The global-rating (*i.e.*, the rating advertised in Google Play) is biased towards apps with very few raters. Once the biased apps are filtered, most apps have a high global-rating which, unfortunately, is resilient to fluctuations once an app has gathered a substantial number of raters.

3. *A Large-scale Empirical Study on Ad Library Maintenance of Mobile Apps (Chap. 5)*

   The app markets combined are estimated to generate revenues of $46 U.S. billion by

2016 (ABI, 2012). App developers are embedding ad libraries in the free to download apps in order to generate revenue from their apps. However, unlike other types of libraries, app developers can not neglect the ad libraries along the lifetime of their apps since app libraries might stop serving ads to older copies of that library, reducing the potential revenue for a developer. Hence, app developers have to perform "ad library maintenance tasks".

By studying 236,245 apps in the Google Play in 2011, we find that ad libraries play a prominent role in the free to download apps, at least 50% of the free to download apps have one or more ad libraries. The percentage of ad supported apps varies across the different app categories. Ad supported apps can contain a large number of ad libraries (as much as 28 ad libraries in an app). The number of ad libraries does not impact the rating of an app. However, the intrusive behaviour of some particular ad libraries leads to lower ratings for an app.

1.3 Thesis Contributions

In this thesis, we empirically study hundred of thousands of apps to validate our results. We make the following contributions:

1. We show that reuse in mobile apps is considerably higher than traditional software. However, we dispel the misconception that growth of the number of mobile apps is due to identical copies of apps, we found a limited instance of such copies.

2. We demonstrate the need for more elaborate rating systems for apps. The rating system used today (inspired by physical goods as done at Amazon) are not able to capture well the dynamic and evolutionary nature of the quality of apps.

3. We are the first to highlight the "ad maintenance" problem as an important challenge faced by developers of mobile apps.

1.4 Organization of the Thesis

The remainder of this thesis is organized as follows: chapter 2 presents current research work related to mobile apps.

Chapter 3 presents an empirical study that examines how the practice of software reuse has contributed to the growth in the number of mobile apps.

Chapter 4 studies the rating of mobile apps in order to analyze if the present rating system encourages app developers to improve their apps.

Chapter 5 presents an empirical study of the challenge that app developers have to face in order to generate revenue from their free to download apps.

Finally, chapter 6 concludes the thesis with a discussion of our results, the limitations and threats to validity, and potential directions for future work.

Chapter 2

Background and Related Work

Mobile apps are known for characteristics different from "traditional" software. Wasserman (2010) emphasized that they are comparably small (several KLOC), usually developed by one or two developers only, and without a formal process. Gasimov et al. (2010) surveyed mobile apps and identified development specific challenges, such as security, limited hardware resources, and scalability. Similarly, Syer et al. (2011) studied three pairs of functionally-equivalent apps from Android and BlackBerry to understand their development and maintenance processes. They found that the development practices of mobile apps differ from other types of software. Also, Shabtai et al. (2010), and Kumar Maji et al. (2010) examined apps from a developer perspective.

However, little or no research has been conducted on understanding the fields of software reuse, customer perceived quality, review and rating systems, mobile advertising, or the study of mobile ad libraries, like we do in this thesis. In the following subsections we present the work related to the present thesis.

2.1 Software Reuse

Code reuse has been analyzed since 1968 when McIlroy (1968) proposed the concept of mass produced software with the help of reusable components. One of the closest works to our study of software reuse is by Li (2012), who implemented a system on Amazon EC2 to detect similar apps and malware across 58,000 Android apps in different Android app markets. However, we focus our work on software reuse, instead of software security. Also, unlike Li (2012), we implemented a lighter weight technique to identify patterns of software reuse, and we only used apps gathered from the Google Play app store.

Conte et al. (1998) analyzed the workload of Java applications. They collected Java applications on the Internet between 1997 and 1998. They analyzed the workload of these applications based on code reuse at the program, class, and method level. In contrast, we analyzed the inheritance and class reuse of Android applications.

Mockus (2007) pointed out the lack of any work that quantifies code reuse in open source software outside the area of code clones. He quantified and identified code reuse in a large set of large open source projects, such as different Linux distributions. In contrast, we analyzed closed source of mobile applications from the Google Play app store.

Michail (1999, 2000) built a tool named CodeWeb to analyze the use of library reuse on source code through data mining. Besides library reuse, we looked for inheritance reuse in this thesis. Our goal is to analyze and measure OO reuse on a large and popular mobile app store.

Denier and Guéhéneuc (2008) presented a model through which they analyzed base classes, and classified these classes depending on the number of methods. While they concentrated their efforts on presenting a model for the understanding of the inheritance within a large software product, we analyzed and quantified the reuse through inheritance

among a large set of mobile apps.

Heinemann et al. (2011) analyzed 20 different Java projects in order to discover the extent of reuse among open source Java projects. They classified the type of reuse in two categories: white-box reuse, the type of reuse that includes the source code; and black-box reuse, the one that reuses software in binary form. In contrast, we studied a larger number of closed-source (but free) mobile apps, in order to discover the reuse along two dimensions: inheritance and class reuse.

Ossher et al. (2011) studied 13,000 open source Java projects in order to analyze the extent of code cloning (a subtype of software reuse). They found that 10% of files are cloned, and more than 15% of the projects contain at least one file from another project. Roy and Cordy (2008) presented a study of identifying function clones in open source software through the use of the NICAD tool. Kapser and Godfrey (2008) classified code clones in three groups (forking, templating, and customization). Hemel et al. (2011) used binary clone detection in order to find software license violations. In contrast, our focus in this thesis is to identify software reuse, instead of identifying new clone detection techniques.

2.2 Review and Rating Systems

The electronic commerce system Amazon.com has been the subject of several studies about its review system. Chevalier and Mayzlin (2003) studied behavioural patterns of reviewers on the two online booksellers Amazon.com and Barnesandnoble.com (BN.com). They found that positive reviews on a book results in the increase of sales of that book. They also noticed that ratings are usually positive: 53% of the reviews are 5 stars in Amazon, and 67% have 5 stars in BN. This last finding is consistent with a previous study by Resnick and Zeckhauser (2002) who found that 51% of the reviews in the online auction website, eBay,

are positive. We found a similar tendency in the Google Play app store. However, the rating in the Google Play app store tends to be biased because of the low number of raters.

Wang et al. (2008) proposed a system to improve Amazon's review system by adding two factors: review credibility and time decay. Mui et al. (2001) proposed a Bayesian probabilistic technique to weight a rating based on the rater's personality.

Cui et al. (2010) studied the effect of early electronic Word-Of-Mouth (WOM) in Amazon from data collected during a period of nine months. Surprisingly, they found that negative reviews have a strong positive effect on the sales of new products (*i.e.*, negative reviews help increase the sales) as long as most reviews are positive. Possible reasons behind this effect is that "bad news" are spread faster. The negative review's effect however, decreases with time.

Our work differs from the previous works mainly in that we use a different object of study, the Google Play app store, where the type of product is software (apps), which change frequently over time compared to books and other products.

2.3 Mobile and Web Advertising

Different studies have been performed to help improve the mobile and web advertisement ecosystem from the point of view of the different stakeholders: advertisers (companies willing to promote their brands), advertising companies (companies dedicated to creating publicity), publishers (*e.g.*, app or web developers that display publicity in their websites or apps in order to generate revenue), and users (app, or web users). The interaction among these four stakeholders is the following: advertisers want to promote their brands, in order to do that they pay specialized advertising companies to create publicity about their brands. Advertising companies and publishers sign a contract in order to allow the former

to display the publicity in the publishers' apps or websites. Advertising companies pay publishers under different schemes such as for every 1,000 advertisements viewed by users in the websites or apps (CPC, Cost Per *Mille*, thousand), number of clicks by users on the advertisements (CPC, Cost Per Click), or if an user completes an action after viewing an advertisement (CPA, Cost Per Action).

Specialized advertising companies, focused on mobile advertising, create advertising libraries (ad libraries). App developers embed these ad libraries into their apps in order to display advertisements in their apps and generate revenue from them.

Several studies have fallen in the field of analyzing the effectiveness of electronic advertisement. Their most important findings are that after one is exposed to too many repetitions of an ad, the ad effectiveness decreases (Kim et al., 2012; Lee, 2004). The effectiveness of an ad decreases considerably by more than 50% if it is shown after the ad of a competitor (Bhalgat and Gollapudi, 2012). Also, advertisers should be aware of the existence of fraudulent clicks on ads. Advertisers end up paying for ads that are clicked, however their final goal of attracting attention of users is not achieved. For example, Dave et al. (2012) present an analysis of fraudulent clicks on ads (click-spam). They expect that app users that are really interested on ads, would spend more time on the advertiser's website. They found that 95% of people who clicked on an ad spent less than one second on the advertiser's website.

Stone-Gross et al. (2011) describe different type of frauds that an advertising company can suffer through its publishers's efforts to increase their profit. These frauds can involve fake clickers (humans and machines that click on advertisements with the only intention of increasing the CPC), excessive number of advertisements to increase CPM, and forcing legit

users to click on ads by convincing them to click on it for other reasons than legitime intentions to know about the ad, or by positioning an ad on an area easily clickable accidentally by an user.

Different studies have focused on trying to mediate between both web developers and users in order to find an equilibrium, and improve the user's experience. Li et al. (2012) found that hundred of webpages are infected by malicious ads, which are served by well recognized advertiser companies. Cases like these affect the reputation of both players: publishers, and advertisement companies. Studies also show that most users reject advertisement campaigns that are built by profiling users through collected personal information (behavioral targeting) (Malheiros et al., 2012; Turow et al., 2009; Ur et al., 2012).

Mobile apps, in order to function correctly, require a set of permissions that app developers define in their code. These permissions allow an app to have access to different functions (APIs) in a device (*e.g.*, the use of the camera in a smartphone). Every time that an app user installs an app, she is granting permissions to the app to execute any function covered by the permissions defined in the app (Blum, 2012). Different studies have been performed related to security issues due to the permissions granted to an app with ad libraries.

Grace et al. (2012) mined the Android Market from March to May in 2011 in order to expose risks and leaks of personal information due to the use of analytic (software library that collects information about the use of an app for their later analysis) and ad libraries in Android apps. They analyzed 100 libraries, and found that 57 libraries collect personal information (such as reading contacts, and phone information in a smartphone), and five out of the 100 libraries analyzed allow the possibility to execute code not authorized by the user. Seungyeop Han (2011) in a technical report about tracking via mobile apps found

that advertising and analytic libraries are gathering persistent identifiers such as AndroidId and IMEI, and even geo-location coordinates are obtained by half of the top 10 advertisers. Stevens et al. (2012) compared the commonalities and differences between browser ads and app ads, with a focus on user privacy. They analyzed 13 different ad libraries in Android apps, and found that almost all the ad libraries can collect sensitive information from the app user, and unlike browser ads, ad providers for Android devices can build a long term app user's profile because of the existence of persistent identifiers in Android devices. Enck et al. (2011) analyzed the top 1,100 popular apps from the Google Play app store, and found that 51% of the apps contain ad or analytic libraries, with up to 8 libraries within an app. They also found that the use and reporting of phone identifiers by ad or analytic libraries are sometimes configurable, and depends on the main application code. Zhou et al. (2012) discovered that the practice of adding or replacing ad libraries is one of the major reasons of repackaging legitimate apps and distributing them via unofficial Android app stores.

Some researchers have dedicated their effort to control the use of app users' private information, as well as separating the permissions requested by the ad libraries from the permissions needed by the apps to function correctly. Leontiadis et al. (2012) crawled the Google Play store during 6 weeks in mid 2011. They classified the permissions that are requested by ad libraries. They defined as dangerous permissions those permissions that are considered high risk and are a threat to the privacy of app users such as access to a user's SMS content and location. They found that 73% of free apps ask for at least one dangerous permission, in comparison to 41% of paid apps. They propose a framework where the app users are in charge of sharing their information to the ad networks, instead of using the apps as intermediary. Shekhar et al. (2012) also present their AdSplit design to separate the advertisement and the app process. Davidson and Livshits (2012) introduce an operating

system service for Windows Phone. This service separates permissions required by the ad libraries from the apps. They conducted an experiment on 3,120 Android apps on which they were able to remove 2,684 permissions. They found that the most common permissions are INTERNET (699 apps) and ACCESS_COARSE_LOCATION (502 apps).

Another field of study about mobile ads has been the related costs that users incur because of the ads in the apps. Pathak et al. (2011) found that the battery consumption because of ad libraries in an app game on Android can be between up to 75%. Vallina-Rodriguez et al. (2012) analyzed the data traffic of a carrier in an European country with more than 3 million subscribers during one day. They found that ads generates up to 1% of all traffic. Furthermore, 3% of Android users and iPhone users generates up to 1MB and 3MB of ad related traffic daily.

We mainly differ from all these works in that we do not focus our effort on security issues. We recognize the app developers' effort to maintain the ad libraries in their apps in order to continue generating revenue. We also noticed that the inclusion of multiple ad libraries have as a goal to increase the source of ads. Also, we present the different ad business models, and the app developers' needs to integrate different ad libraries. We present different problems due to a poor ad maintenance process, and its impact on revenues.

Chapter 3

Understanding Reuse in the Google Play App Store

The number of apps has been increasing exponentially since Apple opened the first app store in 2008 (Apple, 2008*b*). Today, there are hundreds of thousands of mobile apps across various app stores (Hartley, 2012; Rosen, 2012). There is no decrease anticipated in the rate of growth of the mobile apps market (Gartner, 2011; Quirolgico et al., 2011). For example, the Google Play app store (formerly named Android Market) had only 2,300 apps in March 2009 (Lawson, 2009), and the last update by mid 2013, indicates that there are currently more than 800,000 apps available (Canalys, 2013).

There are various potential reasons for this current explosion in the number of available mobile apps. One of them could be the large increase in the number of developers for these platforms (Stackpole, 2011), as well as the availability of decentralized mobile app stores (Rowinski, 2012), or the ease of building new apps (Lohr, 2010). A more fundamental reason why so many mobile apps are developed, might be the use of proven software engineering practices, such as code reuse (Basili et al., 1996; Basili and Rombach, 1991). Such practices have been analyzed in depth on desktop and server software systems (Ambler, 1998; Basili et al., 1996; Basili and Rombach, 1991; Frakes and Kang, 2005), and these principles form the core of any university degree on software engineering. Given the low

threshold for entering the mobile app market, it is not clear how such practices are being followed for mobile app development.

In this chapter, we analyze the adoption of code reuse for mobile app development. Frakes and Kang (2005) define software reuse as "the use of existing software or software knowledge to construct new software". Research has shown that the judicious usage of code reuse tends to build more reliable systems and reduce the budget of the total development (Ambler, 1998; Basili et al., 1996; Basili and Rombach, 1991; Frakes and Kang, 2005). Various types of software reuse exists, like inheritance, code, and framework reuse (Ambler, 1998), each having its own advantages and disadvantages.

In this chapter we focus on exploring the extent of reuse among mobile apps. In order to comprehend the software reuse in the Google Play App Store, we make use of the Software Bertillonage technique introduced by Davies et al. (2011), because the app stores only distribute the bytecode of the mobile apps, and not the source code. Software Bertillonage generates a class signature from each class's bytecode, then compares these signatures across different apps.

We focus our study on the apps available in the Google Play app store since mobile devices running the Android OS (in their different versions) together lead the market of smartphone devices worldwide. Today, smartphones running Android OS have a market share of more than 68% of the market around the world (16.9% iOS by Apple, 4.8% by Blackberry, 3.5% by Windows, the rest of the market share is distributed among Symbian, Linux, and others) (AP, 2012). In order to explore the adoption of reuse by mobile app developers, we conduct a case study on 4,323 apps, across five different categories in the Google Play app store (Cards & Casino, Personalization, Photography, Social and Weather). In particular, we analyze the mobile apps for reuse along the following two

research questions:

RQ1: What fraction of the classes reuse code by inheritance?

We found that almost 27% of the classes in the studied mobile apps inherit from a domain specific base class. However, 23% of the classes inherit from Android-specific base classes. Furthermore, nine of the top ten base classes are from the Android API.

RQ2: How often do mobile apps reuse classes from other apps?

We found that on average 61% of the total number of classes in each category occurs in two or more mobile apps. The classes most often reused are the ads classes provided by Google. Most of the classes reused are from third party developers like Apache and Google. Furthermore, 217 mobile apps have the exact same set of classes as another mobile app in the same category.

This chapter is organized as follows: Section 3.1 describes the study design. Section 3.2 presents the results of our case study and discusses our findings. Section 3.3 presents a detailed discussion on a special case of reuse that we identified during our study. Section 3.4 outlines the threats to validity. Finally, section 3.5 concludes the chapter.

3.1 Study Design

In this section we present the subject systems of our study as well as the approach we followed to extract the required data for our study (Figure 3.1).

3.1.1 Subject Systems

The Android operating system has the highest market share among other competing mobile operating systems (AP, 2012), and has Google Play as its most important app store for Android apps (DrMop, 2011). In the Google Play app store, it is estimated that two-thirds of

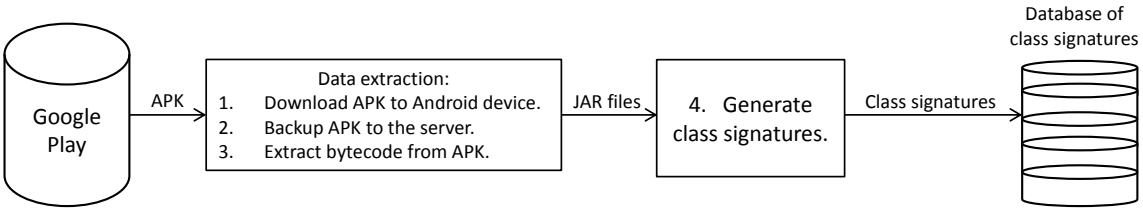


Figure 3.1: Steps for the generation of class signatures from mobile apps in the Google Play app store.

all the available mobile apps are free-to-downloads ("no-cost") (AppBrain, 2013), and free apps are downloaded more frequently (Pettey and van der Meulen, 2012). Hence, we limited the apps in our case study to only the free-to-download Android apps from Google Play.

We selected a sample of free-to-download apps from five different categories in order to perform our analysis of software reuse. The five studied categories are: Cards and Casino, Personalization, Photography, Social, and Weather. We selected these five categories because we expected to find a higher proportion of reuse in them in comparison with the rest of the categories, giving us a best case scenario. For example, the category of Photography focuses only on two kinds of apps in particular: apps to take pictures for devices with camera, and apps to show a collection of pictures. Another example is the category of Weather apps, which focuses on apps for displaying weather information. We felt that the similarity and simplicity of the goals of these apps will very likely lead to a large amount of reuse between apps in each category.

3.1.2 Data Extraction

Each mobile app in the Google Play app store is packaged as an Android Package file (APK). In this subsection, we explain the process that we use to extract the Java Archive file (Jar) from the APKs, *i.e.*, the first three steps in Figure 3.1.

1. Download APK: We manually downloaded every application in the selected categories from the Google Play app store, onto an actual Android device. For the five categories under study, we selected every single free application. To the best of our knowledge, there is no information available about how many mobile apps exist in each category in the Google Play app store. Hence, we continued downloading different mobile apps until no additional apps could be found anymore.

2. Backup APK: We backed up the applications from the Android device to our server with the help of a tool called the Astro File Manager [1]. However, the Astro File Manager was not able to backup mobile apps that were marked as private apps, which are mobile apps that were copy protected. Table 3.1 shows the total number of APKs [2] that we were able to backup to our server using the Astro File Manager.

The process of downloading and backing up the apps required approximately 100 man hours.

3. Extract Jar from APK: Since an APK is an Android specific format, we had to extract the Java bytecode (a regular Java Jar archive) from it. We used an existing tool, dex2jar [3], to perform this operation. We had to modify the names of some APKs in our case study because they made use of special characters that dex2jar was not able to interpret (*e.g.*, Chinese characters).

3.1.3 Class Signature Extraction

We used the Software Bertillonage (Davies et al., 2011) technique to analyze the mobile apps for software reuse. We chose this technique instead of a code clone detection technique, because the Google Play app store (like any app store) does not provide access to the source

[1] Astro File Manager: `http://www.metago.net/`
[2] List of apps used: `http://sailhome.cs.queensu.ca/replication/ICPC2012/`
[3] dex2jar: `http://code.google.com/p/dex2jar`

Table 3.1: Set of Android applications under analysis

Category	Number of applications under study
Cards and Casino	501
Personalization	1,115
Photography	1,034
Social	1,119
Weather	554

code of the mobile apps, only the bytecode. Furthermore, we did not know beforehand the kinds of reuse applied by mobile apps, *i.e.*, whether developers literally cloned whole classes or rather heavily customized some cloned parts of the apps. A more lightweight technique like Software Bertillonage is designed to work with this kind of uncertainty. Below, we briefly explain the Software Bertillonage technique, and how we use it to extract the class signatures that are used in our analysis. This is the fourth step in Figure 3.1.

4. Generate class signatures: Davies et al. (2011) proposed a technique called Software Bertillonage in order to identify the origin (the provenance) of a Jar file in a set of Jar files. Software Bertillonage generates a class signature for each class contained in a Jar file. Then, the class signatures are compared across Jar archives to find similar Jar archives through signature matching.

We developed a tool to obtain the class signatures from the Jar archives of the mobile apps based on a slightly modified version of Software Bertillonage, which we explain below. A class signature typically includes the fully qualified name (name of the class along with the package that it is in) of a Java class. The fully qualified name is chosen to avoid that different classes, in two different packages, that by chance have the same name, and have the same signatures. However, it is important to note that if developers copy and paste code, and alter the package (namespace) of the original class, Software Bertillonage will not be able to build the same signature. Despite these limitations, Software Bertillonage is an intuitive

```
package p.s;

public class ClassName extends
java.lang.Object implements x.y.Z{

        public C(){
                //Class' constructor
        }
        syncronized static int a(
                java.lang.String s) throws
                pack.subPackage.E{
  /*[compiled byte code]*/
                }
}
```

sClass = public class p.s.ClassName extends Object implements Z
sM1 = public C()
sM2 = default synchronized static int a(String) throws E
Bertillonage class signature = <sClass,<sM1,sM2>>

Figure 3.2: Decompiled Java class, class and method lines, and class signature.

technique for our purposes, since it is lightweight and only requires the bytecode to work.

Class signature example: Figure 3.2 shows a decompiled version of a class (ClassName) and its corresponding class signature on the right. A class signature consists of a *sClass* with the *sM* of each method in the class. The first step of our algorithm produces the *sClass*, which is composed of the package and class name. The second step produces a sorted set of method signatures *sM1* and *sM2* (one for each method). This set of method signatures form the second component of the class signature. Note that if the method is neither public, nor private, nor protected, default will be added for visibility.

The three steps in the algorithm are as follows:

(a) Extract the package and class line.

(b) Extract and sort the methods in alphabetical order.

(c) Remove methods introduced to implement non-generic interfaces.

In step (b), Davies et al. (2011) did not sort the methods. However, to avoid the cases where two classes that are identical except for the order of their methods not being recognized as identical, we decided to sort the methods in alphabetical order.

Table 3.2: Set of class signatures by category

Category	Total Number of Class Signatures	Mean	Median
Cards and Casino	59,208	118	65
Personalization	52,425	47	11
Photography	88,156	85	26
Social	250,063	223	93
Weather	84,205	152	45

The above algorithm is repeated for each class contained in each Jar file. The result of the execution of this algorithm is a set of class signatures. We refer the reader to the original paper (Davies et al., 2011) for further details about this algorithm.

We implemented a tool that applies our modified Software Bertillonage technique, in order to identify the software reuse among a set of mobile apps in the Google Play app store. The tool was developed using bcel-5.2 [4], the Apache library that Davies et al. (2011) used to analyze Java binary files.

3.1.4 Database of Class Signatures

The set of class signatures generated from the jar files of the mobile apps using the above steps was stored in a database for our analysis. During the analysis, we found a list of classes named with a single letter (*e.g.*, *a, b, c*). It was unclear whether these classes were introduced by the developers (as we do not have the source code), or by the compiler. For example, developers of Android apps write an XML file that is compiled into a class called R (Resource) in the APK. This class is present in all Android apps that have a user interface. Consequently, we chose to omit classes named with a single letter, since we could not decipher the purpose of these classes from their name.

The total number of class signatures for each category is shown in Table 3.2. Additionally,

[4] Apache Commons BCEL: `http://commons.apache.org/bcel/`

the mean and the median of the number of class signatures per app are also shown in Table 3.2 for each category. As we can see in Table 3.2, the median number of class signatures per app in Personalization was just 11, while the median number of class signatures in the Social category was almost eight times as much. This shows that each of the five categories provide a slightly different view about mobile apps.

3.2 Case Study

The goal of our study is to analyze and understand reuse in the mobile apps of the Google Play app store. Software Bertillonage allows us to study two dimensions of reuse: inheritance, and class level reuse. This section presents the approach and results of our case study for the two research questions.

3.2.1 RQ1: What fraction of the classes reuse code by inheritance?

Motivation: In this research question, we analyze the extent of inheritance present in mobile apps of the Google Play app store. Prior research has shown several advantages (improved conceptual modeling) (Taivalsaari, 1996) and disadvantages (tight coupling to base class) (Holub, 2003) of inheritance in OO programming. By identifying which base classes are inherited more often, developers can allocate appropriate resources to make them more reliable, efficient and modular. Hence, in this research question we want to determine:

(a) What proportion of classes in each category inherit a base class?

(b) What are the top base classes that are inherited?

Approach: In Java, inheritance is determined by the

`extends`

Table 3.3: Inheritance by category (percentages calculated with respect to the total number of classes that inherit a base class in the mobile apps of a specific category)

Category	Percentage of base classes from the Android API	Percentage of base classes from other domain specific classes
Cards and Casino	21%	29%
Personalization	29%	26%
Photography	29%	19%
Social	19%	31%
Weather	25%	25%

In order to answer question RQ1.b, we need to determine the base class in each class signature. To determine the base class, we mined the class signature for the base class by searching for the keyword 'extends'. An example of a class line of a class signature is:

```
public class com.google.ads.AdView extends RelativeLayout
```

Therefore the *AdView* class reuses via inheritance the *RelativeLayout* base class that is part of the *android.widget* package. Then, we grouped identical base classes, and calculated their frequency. In the results, we present the top ten base classes in the five studied categories of Android apps.

Results: (a) Table 3.3 shows the percentage of classes in each category that inherit from the platform base classes and non-platform base classes. Overall, about 50% of classes in each category inherit from a base class. In two of the five categories, the classes in the mobile apps inherit more from the platform base classes, and in two other categories they inherit more from the non-platform domain specific base classes.

(b) On investigating which base classes were inherited the most we found that the "Activity" class in the Android API is the most popular with about 24,266 of the class signatures (9.1%) across the five categories inheriting from it. The other base classes in the

Table 3.4: Top 10 base classes (percentages calculated with respect to the total number of classes that inherit a base class in the mobile apps across all five category)

Class name	**Percentage of classes that inherit from the base class**
android.app.Activity	9.1%
java.lang.Enum	3.4%
android.content.BroadcastReceiver	2.6%
android.widget.RelativeLayout	2.3%
java.lang.Exception	1.6%
android.widget.BaseAdapter	1.2%
java.lang.Thread	1.1%
android.os.AsyncTask	0.9%
android.os.LinearLayout	0.8%
org.apache.james.mime4j.field. address.parser.SimpleNode	0.8%

top 10 most frequently inherited base classes are in Table 3.4. The second column of the table presents the percentage of classes (total classes that inherit = 266,782) across the five categories that inherit particular base class. Nine of the ten classes are provided by Google as part of the Android API. The SimpleNode class was from the *org.apache.james.mime4j* package, this library is specialized to parse Multipurpose Internet Mail Extensions (MIME) messages.

Since it can be expected that mobile apps inherit from platform classes, we also examined the top 10 non-Android API base classes that were inherited. The ranks of the top 10 third party base classes, in the global ranking of base classes, ranged from 10 to 42. The other base classes between ranks 10 and 42 were all from Android. Even among the top 10 third party base classes in Table 3.5, only one (WxFrameworkException by WSI), was from a closed source API. Thus, we were able to identify that most base classes are from available open source (free packages).

Table 3.5: Top 10 base classes that are not part of the Android API (percentages calculated with respect to the total number of classes that inherit a base class in the mobile apps across all five category)

Class name	**Percentage of classes that inherit from the base class**
org.apache.james.mime4j.field.address.parser.SimpleNode	0.77%
com.adwhirl.adapters.AdWhirlAdapter	0.65%
org.apache.http.entity.mime.content.AbstractContentBody	0.44%
org.apache.james.mime4j.field.AbstractField	0.43%
com.wsi.android.framework.wxdata.exceptions.WxFrameworkException	0.39%
org.apache.james.mime4j.field.ParseException	0.38%
org.jivesoftware.smack.packet.IQ	0.36%
gnu.expr.ModuleBody	0.35%
com.qq.taf.jce.JceStruct	0.34%
com.wsi.android.framework.wxdata.exceptions.WxFrameworkException	0.33%

Discussion: Prior research looked at the extent of reuse via inheritance in four open source Java projects (JHotDraw, Log4J, ArgoUML, and Azureus) (Denier and Guéhéneuc, 2008). The results from this research shows that the percentage of classes that inherit from base classes are between 29 and 36% (in ArgoUML and JHotDraw respectively). In comparison, the percentage of reuse via inheritance in each studied category of mobile apps is about 50%.

Software reuse by inheritance is high compared to open source projects. About 50% of the classes in the studied Android apps inherit from a base class. 27% of the classes inherit from a domain specific class, while 23% of the classes inherit from an Android platform specific class.

3.2.2 RQ2: How often does a class from a mobile app gets reused in another app?

Motivation: This question aims to better understand the types of classes that are being reused across mobile apps. By identifying classes that are reused often, the developers of such classes will be able to allocate appropriate resources to make these classes more reliable and efficient. Hence, in this question we want to determine:

(a) How much reuse exists in each studied category?

(b) What is the proportion of classes in each app that are reused in at least one of the remaining apps in the same category?

Approach: In this research question, we used the technique of signature matching proposed by Davies et al. (2011) to identify reuse of class signatures across the mobile apps in each studied category. In order to answer question (a), for each category, we calculated the complement of the proportion of class signatures that occurs in two or more apps. We took this proportion's complement to obtain the Proportion of Class Signatures Reused (PCSR):

$$PCSR = 1 - \frac{Total\ Number\ of\ Unique\ Class\ Signatures}{Total\ Number\ of\ Class\ Signatures}$$

A high value for the PCSR indicates that the reuse of class signatures is high in that category.

The PCSR is only a measure of reuse of class signatures among all mobile apps of a particular category. However, question (b) wonders what happens with the occurrence of the class signatures at the app level, *i.e.*, what percentage of class signatures in each mobile app occurs in other mobile apps of the same category. In order to answer this question, we used another measure called *"Global Reuse"* denoted as *Global(A)*, for a particular mobile app *A*. *Global(A)* is the proportion of class signatures found in a mobile app *A* that are also found

in at least one other app in the same category as *A*. We define *Global Reuse* in a mobile app as:

$$global(A) = \frac{|s(A) \cap s(\bar{A})|}{|s(A)|}$$

Where:

A = Current mobile app under consideration.

$\bar{A}$ = The apps other than A, in the same category as A.

s(A) = Set of class signatures in A.

s($\bar{A}$) = Set of class signatures in $\bar{A}$.

When *Global(A)* = 1, it means that every class signature in *A* occurs in at least one other mobile app in the same category (not necessarily all in the same mobile app). When *Global(A)* = 0, it means that no class signature in the mobile app *A* occurs in any other mobile app in the same category.

Results: (a) Figure 3.3 shows the Proportion of Class Signatures Reused for each category. Our results indicate that on average 61% of class signatures of a category are reused signatures. This high percentage of reuse indicates that very few classes are unique to a mobile app. We also observe that the Personalization category has the highest proportion of class signatures that are reused, followed by Weather and Photography. Social, and Cards & Casino have the lowest proportion of class signatures that are reused.

(b) In order to analyze the distribution of class signatures among individual mobile apps in each category we calculated the *Global Reuse* of each app in all the five categories. We plotted the cumulative values of *Global Reuse* per app in Figure 3.4. The *x-axis* in Figure 3.4 is the percentage of classes in each app that are reused in another mobile app in the same category. The *y-axis* denotes the percentage of apps (cumulative) in each category that has a

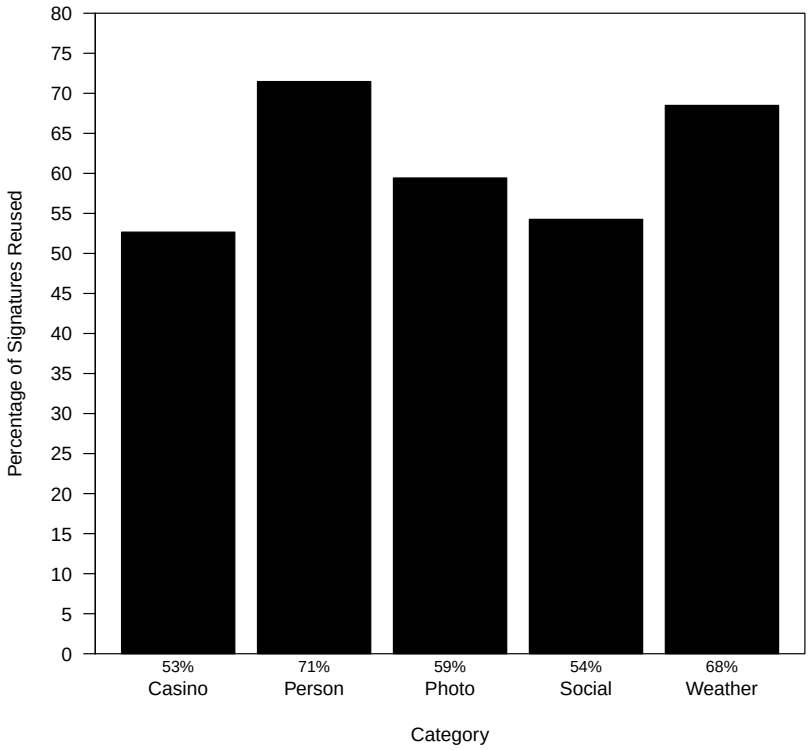


Figure 3.3: Proportion of class signatures reused per category.

particular value of *Global Reuse*. Each category has its own step function in Figure 3.4. We grouped all the apps in ten intervals of *Global Reuse* with each interval having a width of 10%. That is, all the mobile apps in the category with *Global Reuse* in the interval [1, 0.9) are grouped together in one group. All the mobile apps with *Global Reuse* in the interval [0.9, 0.8) are grouped together in another group, and so on.

Figure 3.4 shows that the Personalization category has the highest percentage of apps with *Global Reuse* = 1, that is every class signature in these mobile apps occurs at least in one other mobile app in the Personalization category. The Weather category has the lowest number of mobile apps with *Global Reuse* =1. However, at the same time Personalization has the highest percentage of apps with *Global Reuse* = 0, while Photography is the category

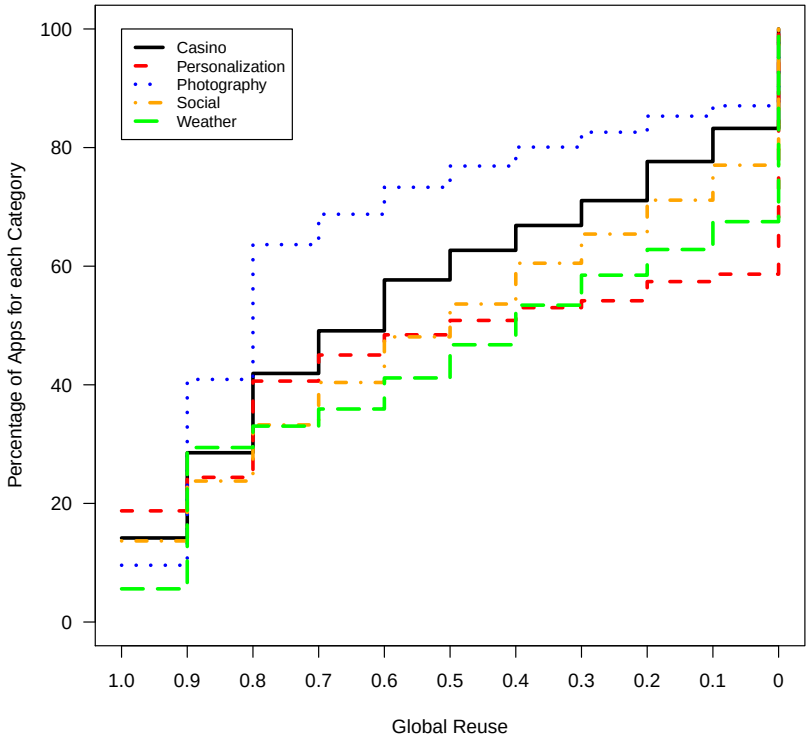


Figure 3.4: Cumulative *Global Reuse* per app among the different apps in each category.

with lowest percentage of apps with *Global Reuse* = 0.

Discussion: Prior research on identifying reuse in open source projects, found that 49% of the analyzed projects (38,700), had at least 80% of their files in common with other projects (Mockus, 2007). From Figure 3.4, we can also observe that anywhere between 20 to 40% of the mobile apps in each category have 80% or more of the class signatures that occur in another mobile app of the same category. However, at 30% code reuse, prior research shows that only 53% of the files were in common. In the case of mobile apps, between 50% and 80% of them have at least 30% of classes in common.

To better understand the reasons for reuse, we took a closer look at the class signatures that were found in two or more apps. First we extracted the name of their

Table 3.6: Top 10 Projects/Organizations

Project / organization name	**Number of unique classes in the project/organization**	**Total number of reused classes from the project/organization**
Apache	4,370	86,489
Google	4,821	39,973
Wsi	311	21,205
Anddev	1,162	20,716
twitter4j	1,363	10,781
Gnu	700	10,220
Android	1,117	7,580
Admob	426	7,023
Jivesoftware	716	6,355
Millenialmedia	388	6,140

Project/Organization from the fully qualified name of the class in the class signature. The top ten projects/organizations are presented in Table 3.6. We noticed that some of the classes are developed by a community of developers who are providing many open source libraries (for example: Apache, Anddev). Only one of them (Wsi) was a company that did not provide its code as open source. From column three we can see that the libraries from the Apache community, are reused the most. We can also see that these ten developers (in Table 3.6) account for 216,482 of the class signatures (almost 50% of the total class signatures across the five studied categories). However, from column two we can identify that most of the unique classes come from the libraries in Google Code.

The high number of reused Google classes are part of Google AdMob Ads API (*com.google.ads*). The purpose of this library is to add an advertisement banner on top of the apps (Google, 2012*a*). The second popular library by Google is part of Google Analytics API (*com.google.android.apps.analytics*). The objective of this library is to provide an API for collecting information about the use of mobile apps (Google, 2012*b*).

Table 3.7: Top five classes across the 5 categories of mobile apps

Class name	Number of apps used in
com.google.ads.AdSize	1,525
com.google.ads.AdRequest	1,280
com.google.ads.InstallReceiver	1,280
com.google.ads.InterstitialAd	8,29
com.google.ads.util.AdUtil	7,27

In the case of Apache, the most popular library reused in the Android apps is to support the MIME protocol (*org.apache.http.entity.mime*) [5]. The second most popular library from Apache is the Apache James Mime4J (*org.apache.james.mime4j*). The main purpose of this library is to provide a parser for e-mail message streams, in two different formats: plain rfc822 and MIME format [6].

WSI is a company that focuses on the commercialization of weather forecast [7]. The WSI classes most reused are *com.wsi.android.framework.wxdata.tiles.TileDownload* and *com.wsi.android.framework.wxdata.geodata.controller.GeoDataCollectionUpdater*.

Next, we wanted to identify the top five classes that were reused. Table 3.7 shows that the AdSize class from Google is the most popular class. It was used in 1,525 apps across the five categories of apps under study. All five of the top five class signatures are from the ads package. The reason for this is the business model used by mobile app vendors. Typically, the free version of an app contains less functionality, uses a freemium model, or shows advertisements. To enable the latter, apps make use of existing ad libraries. Some of the other non-Google classes that were reused often were the ads class from the AirPush API (Airpush, 2010), and classes from the Facebook API.

[5] Apache HTTP MIME: `http://hc.apache.org/httpcomponents-client-ga/httpmime/apidocs/org/apache/http/entity/mime/package-summary.html`

[6] Apache James Mime4J: `http://james.apache.org/mime4j/`

[7] WSI Media Products - WeatherActive Mobile: `http://www.wsi.com/products-media-mobile-weather-active.htm`

Table 3.8: Top 6 Purposes of the classes

Tag/Purpose	Number of classes
Util	33
UI	25
Image	24
Network	24
Ads	22
I/O	22

Finally, we wanted to determine the purpose of the most reused classes whose signatures were reused so often. Hence, we decided to qualitatively analyze the classes based on their class signatures. Since we could not manually tag all 190K classes, we picked a statistically valid random sample with a confidence level of 95% and a confidence interval of 5%. For this, we had to randomly choose 383 signatures. We then manually tagged each class with the purpose of the class. This tagging was based on the name of the class, the package it was in, the classes that it extended, and the interfaces that it implemented. Then we aggregated these tags according to their purpose.

As can be seen in Table 3.8, of the sample of 383 classes, 33 of them (10±5%) were Utility classes. The other tags in the top 5 positions are classes for user interface, image handling, network, advertisements, and input/output.

An example of a highly reused Util class is *org.anddev.andengine.util*. The main goal of this class is to provide libraries for developing videogames in Android [8]. One of the most reused classes of UI is *com.wsi.android.framework.ui.overlay.item.GeoObjectOverlayItem* by WSI. As for the purpose of Image, we found the class *InternalStorageFileBitmapTextureAtlasSource* part of the *org.anddev.andengine.opengl.texture.atlas.bitmap.source.InternalStorageFileBitmapTextureAtlasSource* package by AndEngine. In Network, the most reused

[8] AndEngine - Free Android 2D OpenGL Game Engine: `http://www.andengine.org/`

class is *org.apache.http.conn.params.ConnRouteParams*. This class is part of the HttpComponents library that provides support for the HTTP protocol [9]. In the Ads category, we identified *com.admarvel.android.ads.AdMarvelAnalyticsEvents* developed by AdMarvel (AdMarvel, 2006). Finally, for I/O, Apache provides libraries as part of its project Commons IO. One of these classes identified is *org.apache.commons.io.output.ByteArrayOutputStream*, this is an alternative implementation of the standard *java.io.ByteArrayOutputStreamclass* [10].

On average 61% of the classes in each studied category appear in two or more apps of that category. The classes most often reused in the studied apps are advertising classes.

3.3 Additional Analysis

In the second research question (Figure 3.4) we identified mobile apps that have *Global Reuse* = 1, *i.e.*, every class signature in the mobile app is present in another mobile app in the same category. This is a special case of reuse, since the mobile apps that we studied are end user mobile apps and not just libraries that are intended for reuse. To better understand this special case of reusing classes, we performed additional analysis of this subset of mobile apps. Hence, in this analysis we want to determine:

(a) How many pairs of mobile apps have an identical set of classes?

(b) Who are the developers of such mobile apps?

Motivation: In cases where two or more mobile apps have an identical set of signatures, it is possible that a common framework of classes is being used, especially since the final products are different in terms of their look and feel or purpose. Cardino et al. (1997); Fayad

[9] Apache HttpComponents: `http://hc.apache.org/`

[10] Apache Commons IO Overview: `http://commons.apache.org/io/`

and Schmidt (1997); Johnson and Foote (1988); Mohamed et al. (1997) define framework as a set reusable components that collaborate in together as in a "*semi-complete*" application to create new applications. They show that reusing a framework can increase productivity. Cardino et al. (1997) suggest various advantages and disadvantages of framework reuse. However, the cost of building and adapting a framework can be very expensive. Since there is no current research that discusses the extent of framework reuse in mobile apps, we intend to study this type of reuse. Also, since the mobile apps that we are considering in this section are identical to each other with respect to the set of signatures, we wonder if the reuse is done by the same developer or different developers.

Approach: In order to answer question (a), we introduce a new measure called *Local Reuse* of a mobile app, denoted as *local(A,B)*, in each category. For a pair of mobile apps A and B, *Local Reuse* is the proportion of class signatures found in A, that also occur in B. Unlike *Global Reuse*, we performed a pair wise comparison of mobile apps to determine *Local Reuse*, which is defined as:

$$local(A, B) = \frac{|s(A) \cap s(B)|}{|s(A)|}$$

A high value of *Local Reuse* means that a high number of class signatures are being reused in another app within the same category. Since *Local Reuse* is calculated for every pair wise combination of mobile apps, we only considered the largest value of *Local Reuse* for each mobile app.

Note that we only consider pairs of apps that both have *Local Reuse* = 1 to address question (a). This means that both mobile apps have the same number of classes and each class signature in one mobile app is identical to a signature in the other mobile app.

In order to answer question (b), we looked for the developers of those applications with at least one *Local Reuse* = 1 in the Google Play app store. In order to find the mobile app in the

Table 3.9: Number of sets of mobile apps with identical class signatures and their developer information

Category	# subsets in which mobile apps are identical	# mobile apps across all subsets	# sets in which all apps have same developer
Cards and Casino	4	13	3
Personalization	14	56	13
Photography	18	70	17
Social	8	57	5
Weather	4	21	4

Google Play app store we had to first look for the application package information contained in the AndroidManifest.xml file in each APK. We extracted the AndroidManifest.xml from the APK for every mobile app with *Local Reuse* = 1, using the apktool [11]. We looked for the package information in the AndroidManifest.xml file, and we used it to locate the mobile app in the Google Play app store in order to determine the developer.

Results: The results for questions (a) and (b) are presented in Table 3.9. We were able to find different subsets of apps with *Local Reuse* = 1, *i.e.*, sets of mobile apps that were all identical to each other. We present the total number of subsets in each category in column two of Table 3.9. We also present the number of mobile apps present in each subset, along with the number of signatures in each of them. On the whole, there were 48 sets of mobile apps that had the same set of class signatures. These 48 sets amounted to 217 (5% of the studied mobile apps) individual mobile apps across the five categories. We notice from Table 3.9 that the Photography category has the largest number of subsets, namely 18. The Photography category has as well the most number of mobile apps that have a *Local Reuse* = 1, namely 70.

[11] Android-apktool: `http://code.google.com/p/android-apktool`

For question (b), we found that in 42 out of the 48 sets, all the mobile apps were developed by the same developer. In the other six subsets, different developers built mobile apps that were identical to each other. Most of these cases were found in the social category. **Discussion**: Although, out of the 4,323 mobile apps, only 217 of them were identical to at least one other mobile app, this is a significant number. We found that three of the six sets of apps that were developed by different developers (organization), were from companies that provide mobile solutions to other companies. For example, Forum Runner [12] develops and commercializes libraries to manage a forum within a mobile app. They publish apps in the Google Play app store for different commercial sectors. Other companies in this category use their libraries to generate apps. The remaining three sets of apps were built from sets of open source libraries (from Apache and twitter4j).

Furthermore, even in the 42 sets of mobile apps that were each developed by a single developer, we found that a framework was used. For example in the weather category, there were 11 mobile apps with the same set of class signatures that were all developed by the same developer. When we looked at the actual mobile app in the Google Play app store, we found that each app was basically the same weather app, but for different metropolitan cities in Europe. Since we know that the developer used the same set of class signatures and that the apps look the same, we can infer that the source code was changed minimally. We think that the change could be the parameter that indicates the city in order to get the weather data for the corresponding cities.

Hence, in mobile apps that are identical to another mobile app, it seems like one of the following three types of frameworks is used: (a) private closed source owned by companies for their own purposes, or (b) private closed source owned by companies to develop solutions for their clients, or (c) public open source collection of libraries.

[12] Forum Runner: `http://www.forumrunner.com/`

5% (217) of the studied apps have identical set of classes. App developers are using a framework to build applications that are identical to another app.

3.4 Threats to Validity

This section discusses the main threats to validity that can affect our study and its results.

3.4.1 Internal Validity

The total number of mobile apps under analysis represents a confidence level of more than 99% with a confidence interval of 2%, based on the total number of apps reported in the Google Play app store when this study was performed (380,000 apps). This limited subset of mobile apps was used instead of the whole set, since each app had to be manually downloaded. To back up mobile apps, we used the widely popular tool Astro File Manager. We could not get applications marked as private apps.

Our tool to generate class signatures made use of bcel-5.2, the same library used by Davies et al. (2011). Also, we made use of popular open source tools like dex2jar and apktool. We had to discard class files named with just one letter (*e.g.*, a, b, c) because we could not decipher the purpose of these classes from their name. Also, we excluded mobile applications with *Global Reuse* = 1 that had in total one class signature.

3.4.2 External Validity

Our study analyzes free-to-download mobile apps in five different categories of the Google Play app store. Some developers may have uploaded their applications in the wrong category (*i.e.*, an app that displays a snow city theme in the category of Weather).

To find out if our results apply to other app stores and mobile platforms, we need to

perform additional studies on those environments. Also, in order to generalize our results, we require an analysis of a larger number of apps across all categories available in the store.

3.4.3 Construct Validity

Our results are based on a modified version of the Software Bertillonage technique, and the *Global* and *Local Reuse* metrics. The modifications were made to identify the cases where the developer modifies the order in which the methods are declared in the class that is being reused. Also, *Local Reuse* is the same metric used in Davies et al. (2011) (*Inclusion Index*), while *Global Reuse* is a complementary metric of *Local Reuse*.

3.5 Conclusions

This chapter analyzed Android applications to identify the amount of code reuse in them. We employed several techniques based on the concept of Software Bertillonage to find patterns that indicate how frequently software is reused among all the apps, either via inheritance or via class reuse.

In summary, we found that almost 50% of the classes in the mobile apps of the five studied categories inherit from a base class. We found that most of the base classes that are reused were from the Android API or a third party API.

Additionally, we identified that on average 61% of the classes in each studied category appear in two or more apps of that category. We also found that 20-40% of the apps had 80% or more of their class signatures in at least one other mobile app in the same category. We found that the top classes are those that handle notifications of advertisements and that more than half of the classes that were found in two or more apps were from just ten projects/organizations. In our qualitative analysis, we identified that the top purpose of these

classes were Util classes.

In conclusion, our study provides an initial insight into the code reuse practices of mobile apps. We conclude that software reuse is more prevalent (compared to regular open source software) in mobile apps of the Google Play. *Developers reuse software through inheritance, libraries and frameworks. Most of the reused classes are from publicly available open source artifacts. Developers of these highly reused artifacts can use our results to make them more reliable and efficient. The results also help the developers of mobile apps, to understand the risk they run when the library/framework they build on, changes*.

Chapter 4

A Large-scale Empirical Study on User Ratings of Mobile Apps

Downloads of mobile apps across all platforms (*e.g.*, iOS, Blackberry, and Android) have risen from 7 billion in 2009 to more than 45 billion in 2012, with the corresponding revenues rising from $4.1 billion in 2009 to $6.5 billion in 2010 and $16.7 billion in 2012 (Pettey and van der Meulen, 2012; Sharma, 2010). The revenues are expected to grow to $74 billion by 2017 (Columbus, 2013). In 2010, more than 50,000 developers were creating mobile apps for the Android or iOS platforms (O'Neill, 2010).

This multi-billion dollar market (Gartner Inc., 2011; Sharma, 2010) is centered around the concept of app stores, such as the Google Play app store [1] and Apple's App Store [2], that act as distribution platforms similar to Amazon.com for books [3] by offering thousands of mobile apps. In merely five years, Google Play and Apple's App Store have built up a corpus of more than 800,000 apps each (Canalys, 2013).

Through app stores, users not only can download apps, but also rate them from 1 up

[1] Google Play app store: `play.google.com/store/apps`
[2] Apple's App Store: `www.apple.com/iphone/apps-for-iphone`
[3] Amazon: `www.amazon.com`

to 5 stars. All such ratings of an app version are aggregated into a version-rating, and the version-ratings are then aggregated across all app versions into one average-rating for each app, which we call 'global-rating' in the rest of this chapter. Today many app stores display this global-rating to indicate users about the quality of an app. Recent research shows that such global-ratings have a high correlation with download counts, a key measure of success for a mobile app (Harman et al., 2012). In other words, the global-rating of an app informs potential app users about how other app users perceive the quality of the app (Android, 2013*b*).

Studies in other types of online markets such as online bookstores (*e.g.* Amazon.com), have analyzed the influence of ratings on the decision of a customer about whether or not to acquire a product (Chevalier and Mayzlin, 2003; Zacharia et al., 1999). Similar to online bookstores, in app stores, users of apps are influenced (among other factors) when deciding to acquire an app, by the rating of an app (Harman et al., 2012). App developers generate revenue from an app by (a) selling their apps, (b) selling exclusive content in their apps, or (c) displaying advertisements in their apps (Megan, 2012). However, there is a key difference between mobile apps and books or other products that have a similar ratings system: an app can be updated to a new version in a very short time period, whereas books or other products take more time and are often released as a new product, and not just as a new version of an old product. Despite this difference, app stores still use the same static rating system to help users differentiate the low-quality apps from the high-quality ones. Furthermore, little is known about the major factors that play a role in affecting such mobile app ratings, even though they play a critical role in the success of an app, and therefore the economic success of app developers.

We hypothesize that a good rating systems should encourage the creation of higher

quality apps and be representative of the changing nature of apps (contrast to books which rarely change). Therefore, we sought to study the rating system used by the Google Play store by mining the ratings of all the free-to-download mobile apps in the store during 2011 (128K+ unique apps in total). In our case study we examine the following three research questions.

RQ1. Is the global-rating a reliable measure of the preceived quality of an app?

No. We find considerable evidence to suggest that the global-rating of many android apps is biased. 86% of the 5-star apps throughout the Google Play in 2011 are apps with very few raters (less than 10 raters). When we filter apps rated by fewer than 10 users, we find that most of the apps have an average-rating between 3.8 and 4.4. We also find that these results on the ratings of apps are consistent across all categories of apps in the market. However, the number of raters per app varies considerably across categories.

Such a phenomena misleads users who might think that a version of an app is of good quality when they might not be. We advocate the need for improved rating systems that account for such situations.For instance, we recommend that app-stores not display a rating until a certain number of users have rated the app, or at least make the app users aware about the lack of confidence in the displayed rating.

RQ2. Does the global-rating fluctuate over time as an app evolves from version to version?

No. Once a substantial number of users have rated an app, the app's global-rating is resilient to fluctuations since it is based on the average rating over its entire lifetime. Hence, due to this fact, for apps with a large number of raters we find that it is very difficult to move their global-rating (*i.e.*, the displayed rating) up or down. Therefore even after releasing an improved version of an app, it might be very difficult for an app to have a good global-rating

if it has been given a poor rating by a substantial number of users before, and vice-versa. The resilience of ratings is a major problem that can discourage developers from improving the quality of their app since the rating system cannot factor in recent improvements in ratings. This resilience might also make developers less careful since *botched* releases won't have a major impact on their global-rating. However, this resilience might also encourage developers of well-established apps to experiment since any user discontent can be absorbed by their already high global-rating.

RQ3. Can we predict changes to the perceived quality of an app from version to version?

Yes, we can detect if a new version of an app will lead to an improved version-rating with a precision of 0.64 and recall of 0.60. Our models can be used by app stores to prevent the release of an app which is expected to have a drop in version-rating, and to suggest upgrades to the already-installed user base of a particular app (if an app is expected to have an improved version-rating compared to their installed version).

Overall our study highlights the need for a careful re-thinking of the rating system used today by app-stores. The study also outlines a few possible uses of the historical rating data to improve the quality and speed up the verification of new versions of apps.

This chapter is organized as follows: section 4.1 presents the data used in our study. Section 4.2 presents the results of our study. Section 4.3 expands on the findings along two dimensions - detailed analysis of version-ratings, and qualitative analysis of filtered data. Section 4.4 outlines the threats to validity. Finally, section 4.5 concludes the chapter.

4.1 Data Used in Our Study

For at least the five most popular app-stores (Amazon's Appstore, Apple's iTunes, Blackberry App World, Google Play, and Windows Phone App-store) the global-rating of an app is an aggregation of 1 to 5-star ratings assigned to an app by its users. Our study examines such a rating for the free-to-download apps (henceforth called free apps) that were available through the Google Play store throughout 2011. We focused on free apps since we needed access to the actual binaries for our third research question. Moreover, around 90% of the apps downloaded across all app-stores are free apps (Pettey and van der Meulen, 2012). It is estimated that 75% of the Google Play store apps are free apps (AppBrain, 2013). App-developers therefore, tend to use a different business model than the traditional model in which app users buy an app outright. Developers use advertisements or in-app purchases of additional content in their free apps in order to generate revenue (Megan, 2012). Under this 'freemium' business model, since the price of an app is the same (free), the rating of an app (which is the perceived quality of an app) is the differentiating factor.

At the time of writing, in four of the five important app-stores (Apple's iTunes being the exception), only the global-rating of an app, *i.e.* the average (mean) rating across all ratings of all versions of the app is displayed. In the Apple's iTunes store, the version-rating, *i.e* the average rating across the ratings of only the current version of the app is displayed, along with the global-rating.

We gathered our dataset by crawling the official Google Play store throughout 2011. While downloading the data only once a day in the first half of the year, we were later able to speed up our downloading to twice a day. We needed to crawl the Google Play with such high frequency because we did not know how frequently app developers release a new version of an app. This resulted in a set of 349,906 app versions of 281,079 mobile apps.

For more details about this process a technical note can be consulted (Dienst and Berger, 2012). Our crawl resulted in two data sets:

- **APK files**: These are the binaries of the Android apps, contained in the Android-specific Android Package (APK) format, of each version for each free app collected during our crawls.
- **Metadata**: Information related to each app version. In particular, we have the app category, number of raters, the global-rating, binary size (in bytes), version number, company name, company website, and presence of marketing material like promotional videos.

Since at the time of writing Google Play gathers only the global-rating of an app version (Google, 2013*d*), we need to reconstruct the version-rating of each version of an app as follows:

$$version - rating = \frac{global_i * cumul\_count_i - global_{i-1} * cumul\_count_{i-1}}{cumul\_count_i - cumul\_count_{i-1}} \quad (4.1)$$

where $global_i$ is the global-rating of version i, and $cumul\_count_i$ is the cumulative number of raters of all versions up until version i. A large part of our analysis focuses on apps with at least two or (when studying increases in version-rating) three versions, since we cannot calculate this metric for the first fetched version.

Since Google can adjust ratings in case of fraud [4], or in order to assign a rating to unrated apps [5], we had to sanitize our data to eliminate anomalies in ratings, such as the cases where the total number of raters in a later version was less than the total number of raters in an earlier version. There were 3,891 versions of 3,454 apps which had this issue. We removed these apps from our dataset.

[4] http://support.google.com/googleplay/bin/answer.py?answer=113417

[5] Google Play Developer Distribution Agreement: http://play.google.com/about/developer-distribution-agreement.html

After further filtering out apps for which we did not have complete data, we ended up with 238,198 versions of 128,195 apps.

4.2 Case Study Results

This section discusses the results of our three research questions. For each question, we present our motivation, approach and results. We end each question with a brief discussion about recommendations for app developers and app stores based on our results.

4.2.1 RQ1. Is the global-rating a reliable measure of the preceived quality of an app?

Motivation: This question is exploratory in nature as we attempt to grasp how ratings are distributed across apps and how accurate these ratings are. Since ratings are assigned by users, this means that we also need to explore the number of raters of each app.

Approach: We examine the global-rating of the last version of each app in our data set, as well as its relation with the number of raters and the number of versions of the app. We do this both across all app categories and per app category. The global-rating of an app is the weighted average of the version-rating of all versions of an app. When a user gives a rating, they can choose to award a specific number of "stars" (from 1 up to 5). If two users gave 3 stars to version 1 of an app, and five people give 4 stars to version 2, then version 1 has a version-rating of $\frac{2*3}{2} = 3$, version 2 has a version-rating of $\frac{5*4}{5} = 4$ and the app's global-rating is $\frac{2*3+5*4}{2+5} = 3.7$. While during the time of data collection (2011) Google Play only showed the resulting global-rating, they currently (as of July 2013) show as well a histogram of the number of people giving a particular star rating to an app [6]. This breakdown

[6] http://www.androidauthority.com/android-market-now-shows-apps-star-ratings-breakdown-17916/

of ratings are therefore used only in our manual analysis of special cases and outliers.

Results: **The displayed global-rating is trustworthy with many apps having a low number of versions and/or raters.**

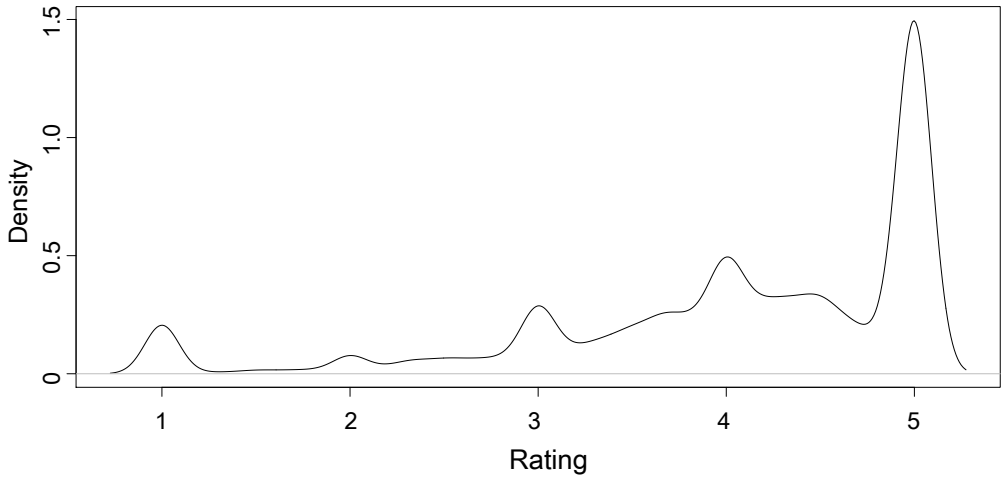
Figure 4.1(a) shows the distribution of the global-rating of the last version of all apps in our data. We note that a large proportion of apps have a rating close to 5, with 33.8% (43,302) of the apps having a rating of exactly 5. The median rating is 4.3, and we notice local peaks for ratings 4, 3, 2 and 1.

However, this distribution is biased, for a number of reasons. First, not every app attracts the same level of user feedback, in the form of ratings. Figure 4.2 shows the number of raters for the latest version of each studied app. The plot uses the natural logarithmic scale because the apps with the highest number of raters have substantially more raters than apps with a lower number of raters. The first 116,570 apps have at most 150 (5.01 logarithmically) raters in a version, while the top 27 apps have at least 200,000 raters (12.21 in logarithmic scale). 25.72% (32,971) of the apps have at most one rater in the latest version of the app in 2011.

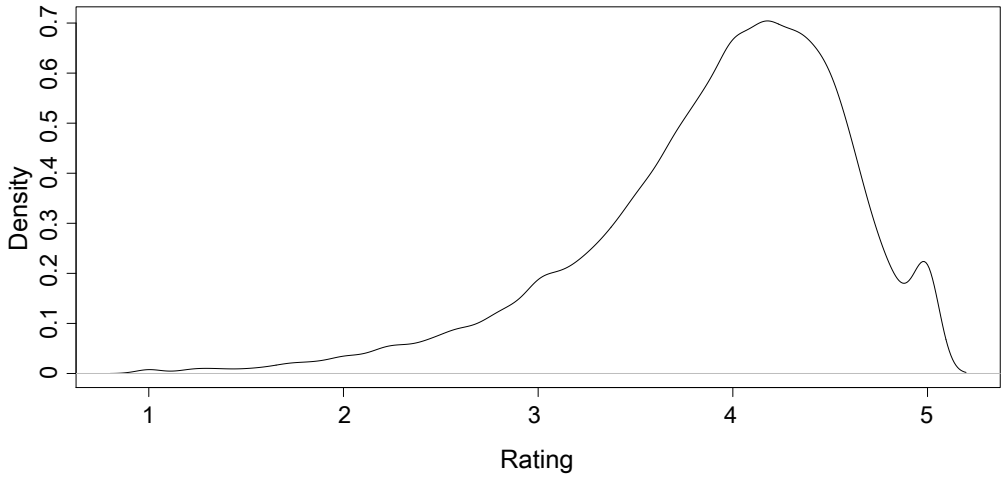
Similar to findings in open source software (Herraiz et al., 2011), the variation and sparsity of the number of raters is a major source of bias for app ratings. If no one rates a new version, this version simply inherits the global-rating of the previous version, even though the quality of the new version may not be equal to that of the old version.

Half of the apps have a rating between 3.6 and 4.4.

To rule out the bias caused by the number of raters of an app, we filtered out apps having less than 10 raters per version. This threshold (dashed line in Figure 4.2) is based on our intuition as we examined the trend lines in Figure 4.2. Higher thresholds like 20 or 50 result in similar distributions, yet with less data. Lower thresholds might contain more unreliable



(a) Density Plot of the ratings of all apps



(b) Density Plot of the ratings of all apps with at least 10 raters

Figure 4.1: Density plot for the ratings of apps

ratings. With our current threshold, the data set shrinks from 238,198 versions of 128,195 apps to 105,808 versions of 42,214 apps.

We can observe from Figure 4.1(b) that the distribution of global-ratings across filtered

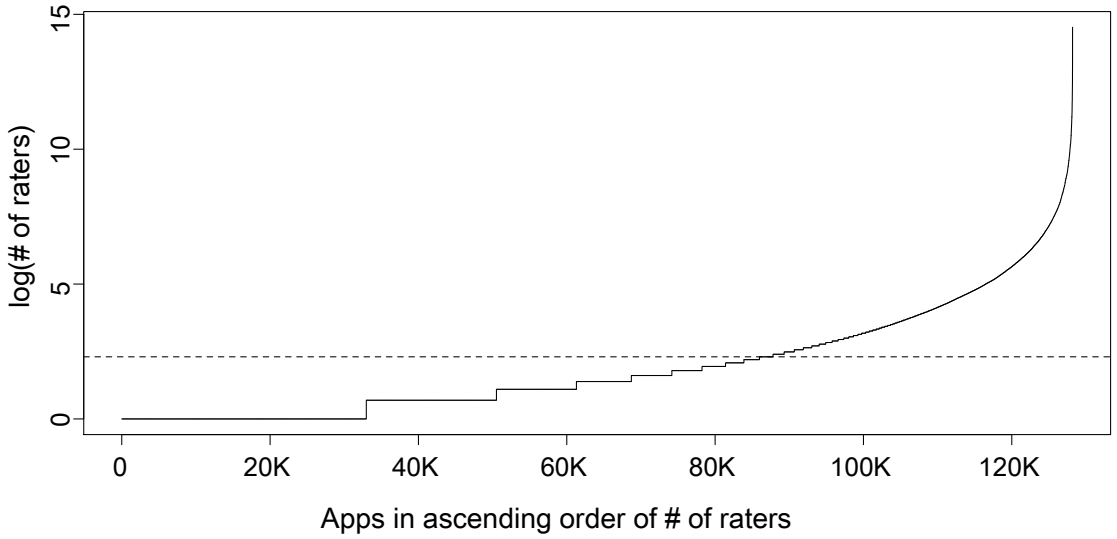


Figure 4.2: Natural logarithm of the number of raters for the last version of each app. The dashed line shows the threshold for the minimum number of raters (10) used to filter the data. Note that the logarithm of 0 is plotted as 0.

apps is slightly skewed to lower ratings (skew of -0.999), with a median rating of 4.02. The apps in the second and third quartile of the filtered apps have a rating between 3.6 and 4.4. The fact that the number of apps with rating 1 and 5 dropped considerably (by 97% and 86% respectively), confirms our intuition that apps with low or high ratings were biased towards apps that were filtered out, *i.e.* apps with less than 10 raters.

Apps having global-ratings between 3.6 and 4.4 also have the largest number of raters. Figure 4.3 shows a scatter plot of the number of raters of the last version of an app in 2011 versus the global-rating of that version. The plot uses hexagonal binning to aggregate multiple observations, while avoiding overplotting. Darker bins contain more observations. The top 3 rated apps in our dataset are the Facebook app [7] (7,474,695 raters), *Words With*

[7] https://play.google.com/store/apps/details?id=com.facebook.katana

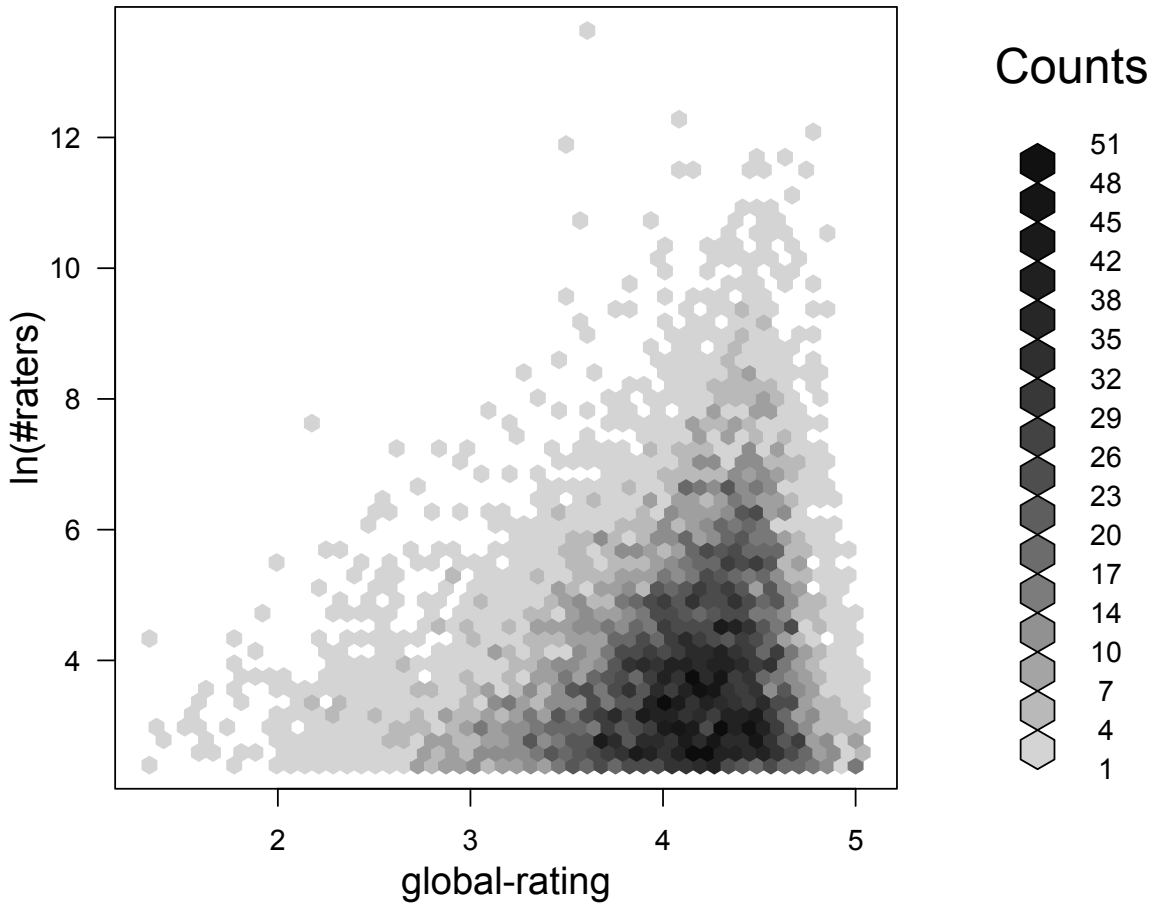


Figure 4.3: Global-ratings vs. the (natural logarithm of the) number of raters for the last version in 2011 of all mobile apps with at least two versions in 2011 and 10 raters per version.

Friends Free [8] (1,304,150 raters) and the Youtube app [9] (1,504,490 raters). 47% (3,519,541) of the Facebook app raters awarded five stars, while at the other extreme 20% (844,842) of the raters awarded one star. As such, the average-rating dropped to 3.6, right below the 25th percentile.

Ratings do not vary much across app categories, while the number of raters varies considerably.

[8] https://play.google.com/store/apps/details?id=com.zynga.words

[9] https://play.google.com/store/apps/details?id=com.google.android.youtube

First, we analyzed for each app category the median global-rating of the last version of all apps in that category. We found that all categories have median global-ratings in the same ballpark, between 3.5 and 4.5 (largely overlapping with our earlier findings) with the second and third quartile of apps in each category having a median distance of 0.6 apart. The "Themes", "Health" and "News & Weather" categories are exceptions to this, with a median global-rating between 0 and 0.3. This low rating can be explained by the fact that our filtered data set contains just 1, 9 and 88 apps respectively.

While global-ratings are relatively within the same range across categories, the number of raters varies considerably. Figure 4.4 shows for each category the median number of raters in the last version of all apps in that category, as well as the 25th and 75th percentile. Popular mobile app categories like games ("Racing", "Sports Games" and "Arcade & Action"), multimedia ("Multimedia" and "Music & Audio"), "Productivity" and "Social" have apps with median value of 340 raters in their last version. Categories with only a small number of apps, like "Health", "Demo" (4 apps) and "Themes" have a low median number of raters of up to 23.

Global-ratings are not reliable when there are not enough raters. Most of the apps have the same range of global-ratings, even across app categories.

Recommendations: A displayed global-rating based on a small number of raters could greatly misrepresent the actual quality of an app. Therefore, our recommendation for the app-store-owners is that *they should display a warning that enough raters are not available for a particular app to calculate the global-rating*. For example, in our study, we determined that 10 developers would give a good general representation. Displaying an unreliable rating for an app, impacts both the app user and app-developer as follows:

- App user - would end up downloading an app with a high yet non-representative

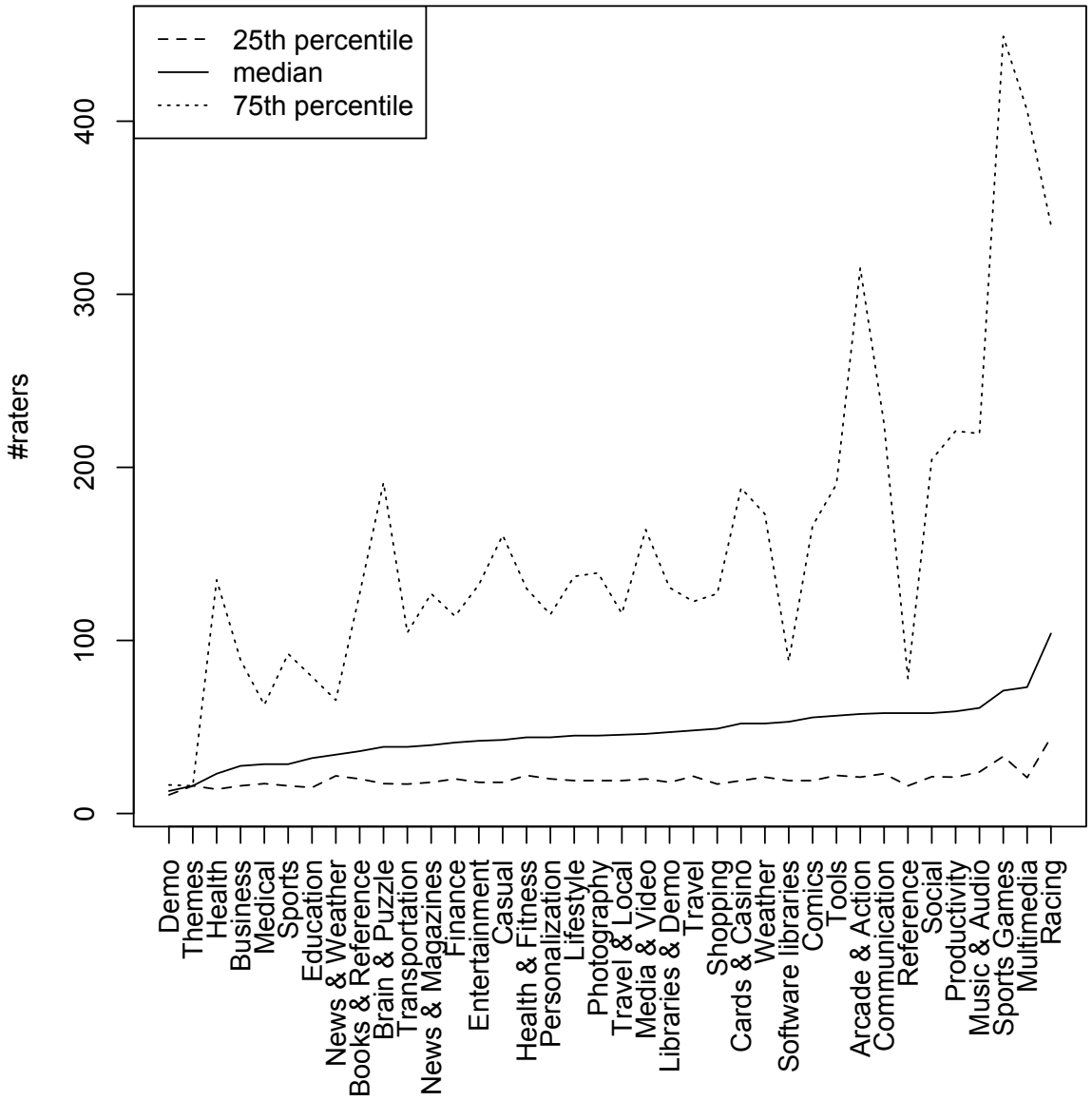


Figure 4.4: 25th/50th/75th percentiles per app category of the number of raters in the last version of the mobile apps with at least two versions in 2011 and 10 raters per version.

global-rating, even though the app is of low quality. Our manual analysis shows that in some cases apps with low number of raters yet high global-ratings are due to bogus high ratings. This could be possible because of a few friends of the app-developer, or

the app-developer with pseudo-identities.

- App-developers - If the global-rating is calculated with a small number of raters, then developers need to be very cautious about their initial release. Otherwise, their app can start with a low rating, and due to the snowball effect, continue to be a poorly rated/downloaded app. Therefore, developers should take more time to produce their initial release and to put great effort and resources into marketing that initial release and securing essential positive ratings early-on (possibly through temporary advertising campaign and improved service).

4.2.2 RQ2. Does the global-rating fluctuate over time as an app evolves from version to version?

Motivation: The displayed global-rating of an app is an important indicator of its user perceived quality. It is essential for such a rating to be representative of the evolving nature of apps, in contrast to books which do not undergo much change (often no change) after they are released. Hence, in this question, we wish to examine the responsiveness of the global-rating to new versions of an app. In particular, we carefully examine whether we would notice a change in the global-rating of an app, given a rise or a drop in the version-rating of that app.

Approach: In addition to filtering apps with fewer than 10 raters, we also filter apps with just one app version in our dataset. This is because, in order to calculate the version-rating, we need at least two app versions. Hence, the set of apps that we use (32,596 app versions across 10,150 apps) have at least 10 raters and at least 2 app versions each. We calculate the version-rating for each app version.

Results: **The global-rating of an app is resilient to fluctuations, once a significant**

number of people have rated the app.

Figure 4.5 plots for each app in our filtered data set the increase in global-rating between their first and last version in 2011 versus the total number of people who had rated a version up until the first version in 2011. Given that the X axis is in logarithmic scale, the plot clearly shows that the more raters for an app throughout its lifetime, the more resilient the app becomes to changes in its global-rating. In other words, apps with a large number of raters technically have some leeway to experiment in a new version without their rating being affected negatively. However, if the global-rating of an app was low, it is nearly impossible to improve it once the app has a large number of raters.

We looked at eight more months of ratings data from four of the apps with the largest number of raters to see how their global- and version-rating have fluctuated (*i.e*, until August 2012). Facebook (777,676 raters in 2011, and 4,336,968 raters in August 2012) dropped slightly from a global-rating of 3.7 to 3.6, while the version-rating only had one significant increase from 3.3 to 3.6. GoogleMaps (693,248 raters in 2011, and 2,060,638 raters in August 2012) dropped slightly from 4.5 to 4.4, while the version-rating decreased once by 0.48 and increased twice by 0.37, ending at 4.6. AngryBirds (545,506 in 2011, and 1,286,863 raters in August 2012) slightly increased from 4.5 to 4.6, while the version-rating increased from 4.5 to 4.7. Finally, KakaoTalk (229,869 in 2011, and 936,497 raters in August 2012) global-rating stayed on 4.5, while by version-rating dropped once by 1.54 and increased once by 1.44. This last case is the most representative example of an app with highly fluctuating version-ratings that do not impact its very stable global-rating.

These observations suggest that the real quality of an app does not really matter once a massive number of users rated the app favourably, since the displayed global-rating remains the same. While this displayed global-rating might be resilient, the version-rating of a

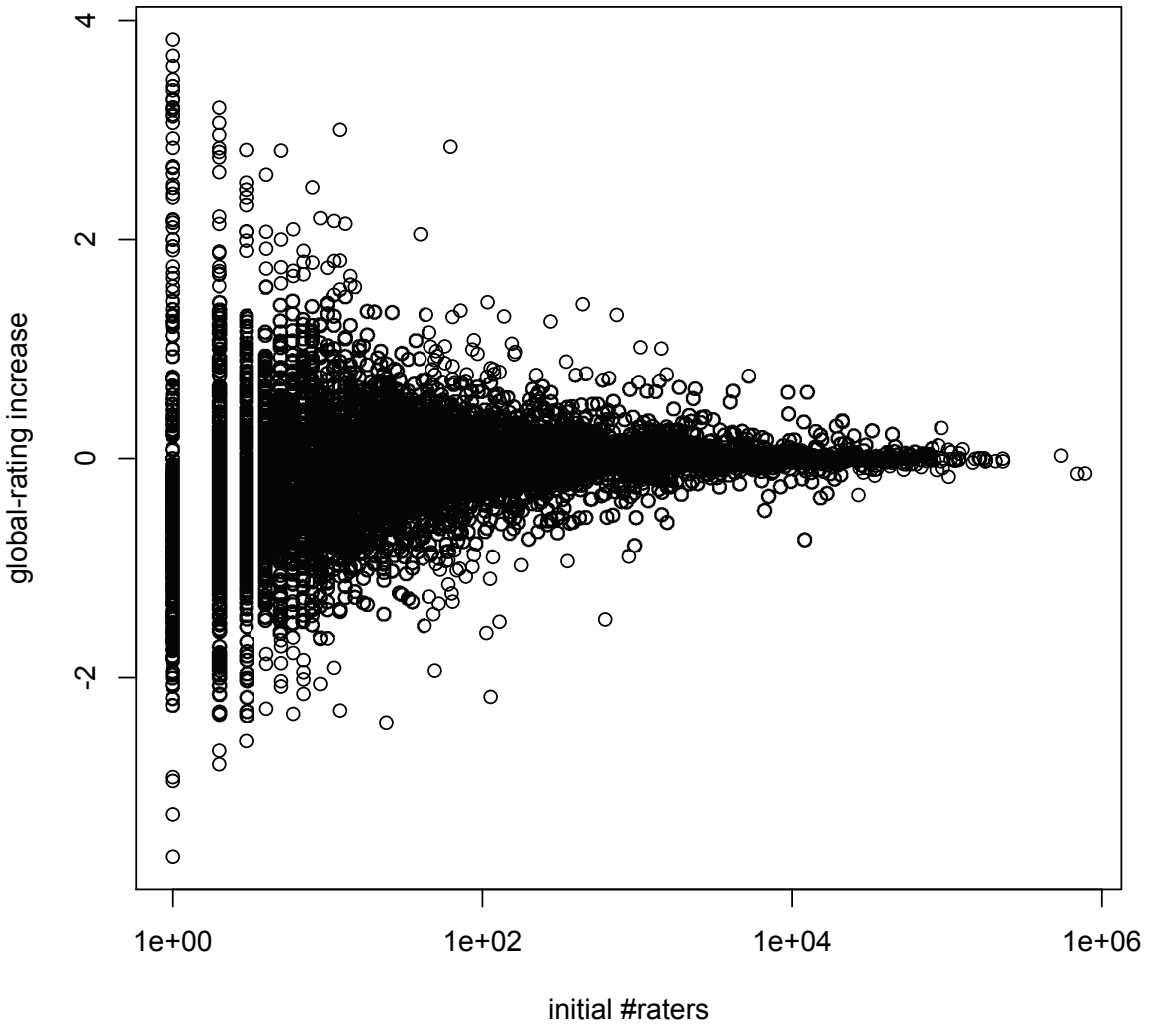


Figure 4.5: Scatterplot of global-rating increase between the first and last version of apps with at least 2 versions in 2011 versus the total number of raters up until the first version in 2011.

specific app version provides a more instantaneous and hence accurate view of the perceived quality of a particular app.

The global-rating is very resilient to changes in the version-rating.

Figure 4.6 plots for each successive pair of app versions the increase in version-rating versus the increase in global-rating. We can see three phenomena. First, the majority of rating

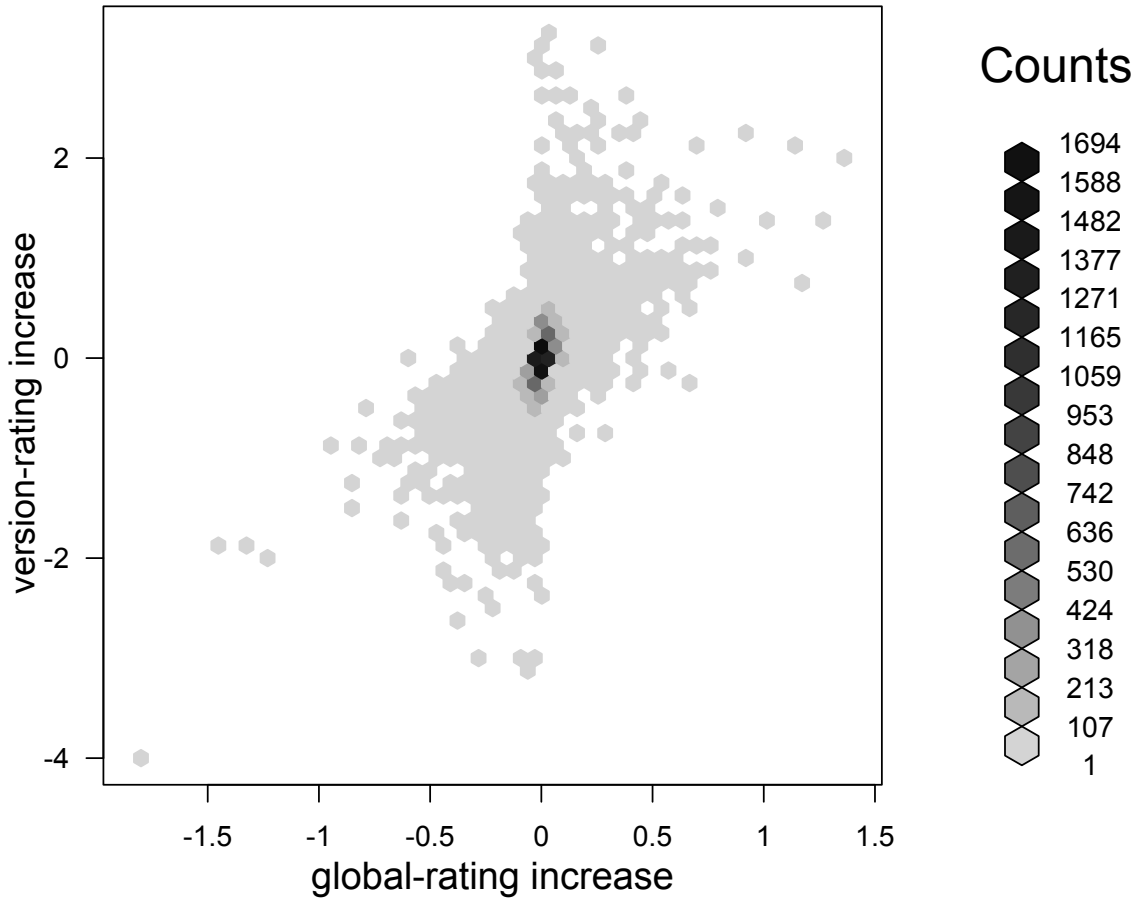


Figure 4.6: Scatterplot of increase in version-rating for each successive pair of all versions of the apps with at least 2 versions in 2011 and 10 raters per version versus the corresponding increase in global-rating.

changes float around 0 for both version- and global-rating (dark black cells), *i.e.*, this is the case where no rating change happens. Second, a long vertical stretch of version delta ratings between -3 and 3 did not have any corresponding change in the corresponding global-rating, confirming that the global-rating indeed dampens the effect of fluctuations in quality. Third, we observe a slight diagonal pattern from the bottom left to the upper right showing that changes in version-rating can have a corresponding, but dampened (across a smaller range of changes), changes in global-rating.

From our data, we also calculate the percentage of app versions in which a change in version-rating results in a change in global-rating. We used the discretized star ratings for this. In 78.8% to 97.4% of all app versions, a version-rating change does not result in a change in the global-rating. 11.2%, 21.0% and 14.3% of app versions with a version-rating change of -1, -2 and -3 stars cause a global-rating change of -1 star respectively, while 9.3%, 20.0% and 14.3% of apps with a version-rating change of 1, 2 and 3 stars causes a global-rating change of 1 star, respectively. Other global-rating changes hardly ever happen.

Global-ratings are resilient to change after a large number of raters have contributed to build the rating, even though the version-rating might fluctuate heavily.

Recommendations: Highly resilient global-ratings prevent apps that have a considerable number of raters from either losing a current high global-rating or improving a poor global-rating. Therefore our recommendations are:

- App-store-owners - should display both the global-rating as well as version-rating (for at least the current version, which the Apple iTunes store currently practices). Otherwise, app-developers have no incentive to maintain/improve the rating (perceived quality) of the app. Therefore the app users do not have access to the best quality apps. Moreover, more advanced methods to create a global-rating should be explored, one option being, a rating system that exponentially decays the older ratings – hence putting more emphasis on recent version-ratings over much older (and most likely outdated) version-ratings. Similar technique has been proposed previously by Wang et al. (2008).

- App-developers - If the current practice of only displaying the global-ratings is followed, then there is limited benefit from releasing a new and improved version of a

low-rated app. Instead we recommend that the app-developer release the new version as a new app, and redirect their current user base to the new app.

4.2.3 RQ3. Can we predict changes to the perceived quality of an app from version to version?

Motivation: One of the key steps, in app markets with verification process, is having the new version of the app go through the verification process of the market (Google Play is the only app market from the five most popular app markets that does not have a verification process). Such verification is often done manually and is quite lengthy (Slivka, 2012) – slowing down the update frequency of apps. Moreover once an app is updated, users are often given very little information about the benefit of updating to the new app. In this question, we explore whether we can create models that can accurately determine if a new version of an app will lead to an improved version-rating. Our models can be used by app-store-owners to prevent the release of an app which is expected to have a drop in its version-rating, and to suggest upgrades to the already-installed base of a particular app (if an app is expected to have an improved version-rating compared to their installed version).
Approach: We examine the relationship between the version-rating and the six dimensions of metrics for mobile app development presented in Appendix A. We hierarchically build logistic regression models to model the increase or decrease in the version-rating of an app relative to its prior version. A logistic regression model is a regression model that outputs a probability, in this case the probability that an app version will see an increase in its version-rating. Probabilities higher than a chosen cutoff value, in our case 0.5, mean that we classify this app version as having an increase in version-rating. By building a regression model hierarchically, groups of related independent variables (the six dimensions of metrics)

are added one at a time, such that the impact of each group of metrics on the dependent variable can be analyzed.

We use the same filtered dataset that we used in RQ2 (apps with at least 10 raters and at least 2 app versions). However, before we can build the model, we need to preprocess the data and build the training and test data sets. To eliminate dependencies between versions of the same app, we randomly select one app version of each app. We randomly split this set of app versions into a training set and a test set with 2/3 and 1/3 of the data, respectively. We then build a model on the training set, and evaluate its precision and recall on the test set. Precision measures the percentage of app versions classified as having a version-rating increase that actually had a version-rating increase. Recall measures the percentage of app versions having a version-rating increase that were classified as such by the model.

Results: **Size, OO Design and Reuse metrics are most closely related with version-rating increases.**

Table 4.1 shows the resulting model after carrying out hierarchical logistic modelling and rebuilding the model with only the statistically significant dependent variables. Applied on the test set, this model achieves a precision of 0.64 and recall of 0.60. That is, 64% of the times when the model says an app version will have an increase in version-rating, it is correct, and 60% of all app versions with an increase in version-rating are correctly classified as such by the model. Hence, the performance of this model is considered moderate (Connolly and Sluckin, 1953). Repeating the model building with different random splits of the data yielded similar results.

In order to measure the impact of each dependent variable on the probability that an increase in version-rating of an app, we calculate the odds ratio of each variable, *i.e.*, the factor with which the probability of version-rating increase is multiplied if the variable is

Table 4.1: Logistic model to classify an app version as having an increase or decrease in version-rating. ***/**/*/. Means statistically significant independent variable for confidence level of 0.001/0.01/0.05/0.1. Odds ratios in bold are statistically significantly different from 1 with a confidence level of 0.05.

Indep. Variable	Estimate	Odds Ratio	Dimension
(Intercept)	3.70533 ***	**40.66**	
Previous version-rating	-1.04351 ***	**0.35**	Base
log(Binary size)	0.06926 *	**1.07**	Size
log(Δ Total classes)	0.36670 **	**1.44**	Size
CA	-0.17642 .	0.84	OO Design
log(Δ NPM)	-0.48288 *	**0.62**	OO Design
log(Δ Unique classes)	-0.06319 .	**0.94**	Reuse

increased by one. An odds ratio of 1 indicates that the particular variable has no significant effect. An odds ratio between 0 and 1 (not including 1) indicates a decrease in the probability of a version-rating increase. The larger the odds ratio becomes, the larger the impact on the probability of a version-rating increase, and the lower the odds ratio, the lower the impact.

We can see that large apps (Binary Size) and app versions that have added a large amount of functionality (Δ Total classes) both increase the probability of an increase in version-rating. On the other hand, adding substantial app-specific classes (Δ Unique classes), making more methods public (Δ NPM) and already having a high version-rating (Previous version-rating) decrease the probability of an increase of version-rating. CA's odds ratio is not statistically significantly different from 1 (confidence level of 0.05).

Our model can predict if the version-rating of the upcoming version an app will have an increased or decreased rating with a precision of 0.64 and recall of 0.60.

4.3 Discussion

In this section, we analyze the version-ratings in more detail and manually go through some of the filtered apps and ratings to better understand the studied data.

4.3.1 In-depth analysis of version-ratings

Our analysis shows that the global-rating is very resilient to changes in version-ratings. However, this finding might not be significant if the version-ratings are relatively stable from version to version. Hence we sought to examine how version-ratings fluctuate over time for the studied apps.

We calculate the percentage of app versions (with a particular version-rating), that has a new version whose version-rating differs anywhere between -4 and 4 stars. Such a percentage indicates the chance that a new release of an app will have a better or worse rating (perceived quality). We can also see if improving the rating is more likely than decreasing the rating. Figure 4.7 graphically illustrates the percentage for each transition. Each node is a specific version-rating, and size of the node is proportional to the percentage of app versions where there is no change in version-rating between successive app versions. The edges show the percentage of new app versions that have an increased or decreased version-rating between successive app versions, with the edge width being proportional to these percentages. We note that apps with a 4 or 5 version-rating have a relatively stable future version-rating since the node size is large, while a substantial percentage of apps with lower version-ratings increase to a version-rating of 4.

Also except for apps with a version-rating of 5, the chances of a new version improving or at least keeping the version-rating of an app is quite large – indicating the importance of having the global-rating more responsive to changes in the version-rating. With up to 45% (apps with version-ratings 1, 2, 3, 4) of apps increasing by 1 star, up to 30% (apps with version-ratings 1, 2 and 3) of apps increasing by 2 stars, and even up to 50% (apps with version-ratings 1, 2) of apps increasing by 3 stars. On the other hand, apps with version-rating 4 (80%) and to a lesser extent rating 5 (64%) are stable with no change in

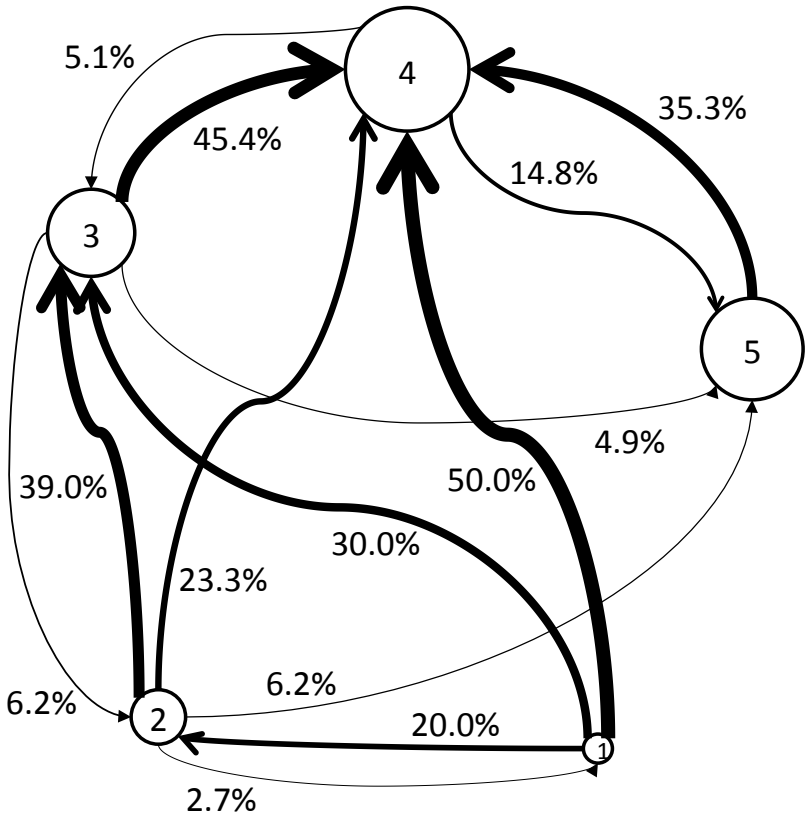


Figure 4.7: Graph showing the percentage of apps that increases or drops its rating from a certain version-rating (node). The node size is proportional to the percentage of apps retaining the version-rating, while the edge size is proportional to the percentage of apps losing/gaining a rating. Erased edges had a percentage lower than 1%.

version-rating between successive app versions. Finally, 35% of apps with version-rating 5 decrease by one star.

To measure the recovery of version-ratings, we filtered our data set down from 10,150 apps to the 4,537 apps that have at least 3 versions, since otherwise apps do not have a chance to recover from an increase/decrease in rating. The median number of versions in the corresponding data set is 4, with the top quartile of apps having at least 5 versions (with a maximum of 36). We then processed these 12,296 versions of 4,537 apps, to determine for

to → / ↓ from	1	2	3	4	5
1	0	0	0	4	0
2	2	15	30	16	5
3	0	33	233	239	24
4	2	15	216	4148	756
5	0	2	11	663	1345

(a)

to → / ↓ from	1	2	3	4	5
1				0.00	
2	1		0.30	0.00	0.00
3		0.91		0.38	0.12
4	1	0.87	0.83		0.75
5		0.50	0.45	0.60	

(b)

to → / ↓ from	1	2	3	4	5
1				X	
2	1		3	X	X
3		2		10	2
4	1	1	4		8
5		2	2	9	

(c)

Figure 4.8: Recovery from an increase/decrease of version-rating in row X to the version-rating in column Y for the year of 2011: (a) #app versions experiencing an increase/decrease, (b) the percentage (expressed as a value from 0 - 1) of apps that recovers from an increase/decrease later on, and (c) the maximum time (in #versions) for an app to recover. An X means that no app recovered (recovery percentage is 0), while an empty cell is infeasible.

each version whether it had experienced an increase/decrease in version-rating, and, if so, we checked the later versions to see if a corresponding decrease/increase in version-rating occurred. For example, if a version increased its version-rating from 3 to 4, we checked if and how often a later version dropped back to 3 or below. Figure 4.8(a) shows the number of app versions that experienced an increase/decrease from the version-rating of row X to that of column Y for the year of 2011. For example, 239 times an app's version-rating increased from 3 to 4. In general, the majority of rating changes involved rating 4, *i.e.*, 5 to 4, 3 to 4, 4 to 3 and 4 to 5.

Figure 4.8(b) shows the percentage of apps that bounce back from an increase/decrease

in version-rating for 2011. Between 60% and 100% of the apps could recover from a drop of 1 star in version-rating. This number decreases up to 60% for apps recovering from 5 to 4 stars. 45% of apps dropping from 5 to 3 stars recover, and even 87% of apps that drop from 4 to 2 stars recover. 25% to even 70% of apps that increased 1 star have a chance of keeping this extra rating. In general, a large number of apps recover from a drop in their version-rating.

Finally, Figure 4.8(c) shows the maximum time (in terms of number of versions) that it takes for an app to bounce back or recover from an increase/decrease in version-rating for 2011. The maximum time ranges from 1 version (*e.g.*, 2 to 1) to 10 (3 to 4). The median time is 1 version (except for 2 to 3, 3 to 5, and 5 to 2, when the median time is 2 versions), meaning that in at least 50% of the cases a change in rating is only temporary.

The perceived quality of an app changes from version to version. App-developers are more likely to release a new version of the app that will have an improved rating. Most apps recover a drop in rating within the next few versions. Hence, it is important for the app market to display a responsive and representative global-rating for each app.

4.3.2 Manual Analysis of Filtered Data

Since we filtered a large number of apps with unstable ratings, we manually examined those apps with high instability in their ratings by consulting the app users' comments posted on Google Play alongside the users' ratings. We found 42 apps that drastically decreased their rating from five to one star, while 11 apps that did the opposite. Twenty-nine out of these 42 apps do not exist anymore in Google Play, while six apps do not have any comments. Three apps only have one short comment like *"works!"*. Only four apps have more extended

comments.

One reason for large drops in ratings are likely biased initial ratings, possibly by the app's author or friends. The app *Glamour Friends & Fans* [10] was rated with five stars and described as *"Awesome!"*, but the same version (1.0.4) quickly got one star ratings complaining that it *"Does not work"*. Another app, *Remote Gallery (SSH)* [11], has three 5-star ratings as first comments. However, its last 2 ratings drop to only one star, one complaining that the app is very slow, and one that it is too complex to set up. Finally, the app *Bitcoin Trader* [12] has only one comment stating that the app works and awarding five stars. However, all subsequent users rate the app with one star, with comments such as *"Copy of another app"* or it *"doesnt work"*.

A second reason for large drops are annoyances with actual features. The app *UK Trains Journey Planner Free* [13] has as first comment a user rating the app with 3 stars not because she did not like the features, but because the app forces the app to be closed upon updating. Other users routinely mention their disapproval of the use of Airpush (a system for advertisements that pushes notifications even when the app is not in use), for example (user *"derek_c"*): *"I uninstalled this app as soon as I discovered that it uses the offensive Airpush system which pushes adverts to your phone's Notifications. Note, I have no objections to in-game advertising, just to ads spilling out onto my phone's desktop."* [14]

Of the 11 apps that increase drastically their rating from one to five, four could no longer be found and one did not have any comments. Two apps have only one negative comment from this year (not 2011). The remaining three apps initially only have positive comments,

[10] https://play.google.com/store/apps/details?id=com.dvmobile.spyderlynk.glamourapp

[11] https://play.google.com/store/apps/details?id=net.cachapa.remotegallery

[12] https://play.google.com/store/apps/details?id=net.dazoe.android.bitcointrader

[13] http://goo.gl/rY67V

[14] http://goo.gl/vNxR8

before the users' comments become negative. Some of the associated comments were even removed, possibly by Google [15] or by accident [16]. In any case, the comments do not seem to shed any light on the jump from 1 to 5 stars.

It is difficult to make all app users happy - different app users rate the same app with extreme opposite number of stars. During our analysis we also observed that there is instability among the app ratings. Popular apps receive a high number of five stars, but also a high number of one stars. This observation is consistent with a previous study that showed that the number of defects reported for an application is highly dependent on its popularity (Herraiz et al., 2011). We decided to look manually for some users' comments that could give us an insight about this instability in rating.

In particular, we analyzed the users' comments of the Facebook app [17], which is the free app with the most ratings in the Google Play store (7,474,695 raters). 47% (3,519,541) of users rated it with five stars, in contrast to 20% (1,461,654) of the users rating it with one star. We looked for comments from those two groups and compared them on version 1.9.7 of the app, across a common set of devices (to rule out differences due to hardware capabilities or version features).

We found that when users are *happy* with an app, they only show their gratitude with a brief statement like *"Great..."* or *"Love it :-)"*. In contrast, if users are *unhappy* with an app, they point out the reasons for this in much more detail, for example *"The news feed takes ages to load and the photos dont even show up unless you click on them. Please fix !!!"* or *"Pls bring back the old chat with a status bar icon. Take away the messenger.. unable to see online friends.. :-("*. These opposite user perspectives are an indication about how difficult it is to reconcile all users and opinions into one rating number. However, alternatives for

[15] http://support.google.com/googleplay/bin/answer.py?answer=113417
[16] http://goo.gl/A7wha
[17] https://play.google.com/store/apps/details?id=com.facebook.katana

ratings as approximation for user appreciation are not immediately available.

4.4 Threats to Validity

4.4.1 External Validity

We only studied the Android platform with data spanning 2011 since we had readily available APIs to mine apps from the Google Play store. We chose to study only the free apps because we required access to the source code or bytecode, which is not possible for paid apps without actually paying. Since the Android platform has the largest user base (Kellogg, 2011), apps are downloaded more often (Pettey and van der Meulen, 2012), and free apps represent 75% of the total number of apps in the Google Play app store (AppBrain, 2013). Hence, we believe that our case study space is broad and practical enough to draw valid conclusions.

4.4.2 Internal Validity

We used publicly available APIs to mine the Google Play app store. Our crawling tools collected the data from the store automatically. Although we have taken every possible step to ensure correct working of our tools, we could not collect every app in the market every day, since we are restricted by the number of queries that can be run against the app store. This means that potentially we have missed particular versions of an app. However, since these crawling APIs are used extensively by others too, and since we crawled the data with high frequency, we believe that our results do not suffer significantly from such threats.

4.4.3 Construct Validity

We use the user-assigned ratings of apps as a measure of user-perceived quality. Although an app might be of high quality from the developer's perspective, if the app does not have attractive features, generates too many ads or behaves poorly compared to a competitor, it might have a poor rating, regardless of the app design or other characteristics that we considered in our study. Furthermore, users might rate an app negatively on purpose or might neglect to rate an app because they are extremely happy or unhappy.

In the present Google Play app-store, it is not clear when app users rate an app. App users may rate an app at any time independently of the version released on Google Play. Hence the version-rating, that we calculate may not be perfectly accurate. However, the version-rating is not inherently present in any of the app-stores (except the Apple iTunes app-store), and since we downloaded the apps at frequent intervals, we believe that we captured the version-rating as accurately as possible. Finally, only a small number of people might rate a particular app, yielding a relatively unreliable rating. Since we filtered out apps with too low number raters, the effect of few anomalous raters should be dealt with.

4.5 Conclusions

This study examined almost 240K versions of over 128K mobile apps from the Google Play store collected over a time span of one year in order to analyze the dynamics of user ratings of mobile apps. Since such ratings are one of the primary means for apps to attract users and generate revenues, we studied how ratings vary across apps, what is the impact of using the global-rating for an app, and how the ratings of apps vary from one version to another.

The four major findings in our study and the corresponding conclusions are as follows:

- The current method of rating apps using global-ratings is extremely biased (most apps

have an above-average rating of 5, while many apps do not have sufficient raters) and does not accurately capture the actual user-perceived quality of a specific version. Hence, app-store-owners (Google Play app store in our case) need to come up with a rating system that can handle apps with few raters appropriately.

- Due to the cumulative nature of the global-rating, it is very difficult to climb up from an initial poor rating after a large number of raters have rated an app. Hence, it is in the best interest of the app-developers to release a good first version, especially since current ratings dictate the future ratings significantly. Conversely, it is difficult, but not impossible, to lose an overall good global-rating. Therefore, app-store-owners should consider decaying older version-ratings when aggregating ratings to create the global-rating of an app, especially since the global-rating has a high impact on the popularity and the revenues from an app.

- We are able to predict the version-rating increase with our models. App-developers would greatly benefit from such a service, where app-store owners can provide early feedback on the quality of the app.

- When we look at the version-rating, we found that more apps tend to climb to a higher version-rating than fall to a lower version-rating. For example, from Figure 4.7, we can see that 45.4% and 4.9% of the apps climb from a version-rating of 3 stars to a version-rating of 4 or 5 stars respectively, when only 6.2% and 0% of the apps fall from a version-rating 3 stars to a 2 or 1 star rating respectively. We can see a similar trend for rest of the version-ratings (1, 2, 4, and 5 star) as well. From this we can conclude that developers are constantly trying to improve the ratings of their app even though such changes are not reflected in the global-rating.

Chapter 5

A Large-scale Empirical Study on Ad Library Maintenance of Mobile Apps

A traditional market system dictates that software users pay an established monetary price in order to acquire a software product. However, it is estimated that 90% of the apps downloaded across the different app stores are free-to-download apps (Pettey and van der Meulen, 2012). In Google Play alone it is estimated that 75% of all apps are free (AppBrain, 2013). This preference for free apps by app users, has led app developers to use a different business model than the traditional model in which app users pay for an app. Various models have popped up, ranging from free demo versions, in-app purchases and in-app advertisements. One of the most popular app business models by far is the mobile ad business model, *i.e.*, in-app advertisement (MobiDev, 2013; Prodhan, 2012).

App developers integrate ads in their apps using ad libraries. Such ads then show up at run-time. If a user clicks on such an ad, she is transferred to the website of the advertiser and the app developer receives a small financial reward for attracting traffic to the advertiser. Given the number of free apps available in the Google Play app store and the percentage of free apps downloaded, it is not surprising that app developers are heavily depending on the

ads to see financial compensation for their efforts. Previous studies have found that between 51% to 60% of free Android apps have at least one type of ad library (Davidson and Livshits, 2012; Enck et al., 2009; Grace et al., 2012). However, despite this high adoption, developers need to be aware of the increased software maintenance costs due to ad library maintenance.

We define ad library maintenance in accordance to the traditional definition of software maintenance (ISO/IEC, 2006) as the different tasks of software maintenance that app developers have to perform in an app after its initial release in order to continue generating the maximum revenue from the ads in their apps. The main goal of a proper ad maintenance is to maximize ad revenue, despite problems like changing ad serving models (Saverio, 2010), the difficulties of some advertising companies to keep on serving new ads (Saverio, 2010), and the bad reputation of certain ad libraries which can harm the reputation of an app (Cleff, 2010; eMarketer, 2012; Oswald, 2011; Virki, 2012).

App developers need to choose among three different ad library models to embed ad libraries into their apps: (a) *ad network* libraries allow developers to request ads direct from an advertising company. Under this model, app developers have to add one ad network library for each advertising company to request ads for each ad library. (b) The *ad mediator* model allows app developers to request ads from different ad network libraries. App developers have to embed an ad mediator library, and the different ad network libraries in their apps. However, app developers only request ads through the ad mediator library. In this ad serving model, app developers continue receiving paying direct from the different ad networks embedded in their apps. (c) The *ad exchange* model allows app developers to request ads from different ad networks through only one ad library that they have to embed into their apps. Different ad exchange companies have agreements with distinct ad network companies.

Given the importance of the ad maintenance as part of the software maintenance tasks performed by app developers, and for the economic success of app developers, we study the ad maintenance in mobile apps. To examine the ad maintenance in mobile apps, we split this chapter into the following four research questions:

RQ1: What percentage of the free-to-download apps are ad supported?

51.21% of the free-to-download studied apps from the Google Play app store have at least one ad library. The distribution of ad libraries varies across the different categories of apps. Up to 35% of the ad-supported studied apps have two or more ad libraries. Apps can have as much as 28 ad libraries.

RQ2: Which ad serving model is more prominent?

88% of the ad-supported studied apps use the ad network serving model. The ad exchange and the ad mediator serving models have a similar share of apps, 11% and 9% respectively. Hence, probably app developers are preferring to obtain high quality of ads (de Souza, 2011), or to deal direct with the source of ads than with a intermediary.

RQ3: How are the ad libraries maintained in the ad-supported apps?

The frequency of ad library maintenance in apps is high, with up to 48% of the versions of an ad-supported app containing changes to ad related libraries.

RQ4: What is the impact of the ads on the app user perceived quality?

The number of ad libraries is not related to the user perceived quality of an app. The logic behind this relation is that the number of ad libraries is not equal to the number of ads displayed at the same time in an app, but a technique to increase the source of ads. However, the intrusive behaviour of some ad libraries has a negative impact on its quality.

This chapter is organized as follows: section 5.1 presents the technical background and motivation for our work from a software engineering perspective. Then, section 5.2

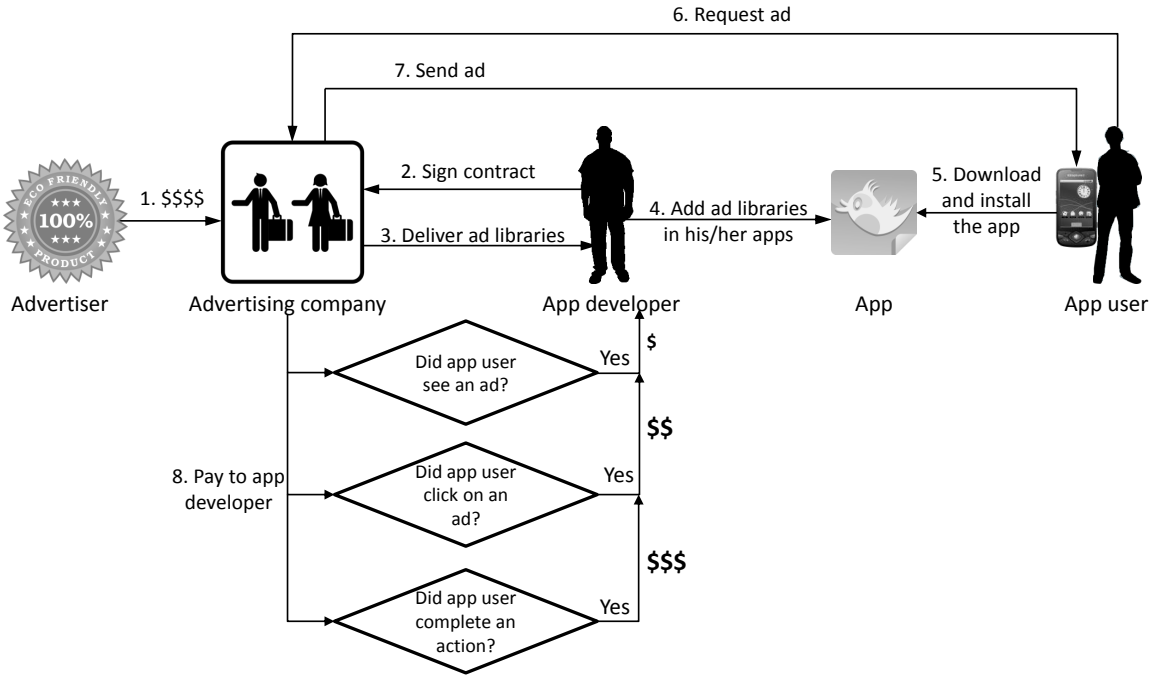


Figure 5.1: Ad serving process.

describes the case study data and design. In section 5.3 we present our results for the four research questions that make up this chapter. Section 5.4 outlines the threats to validity. Finally, section 5.5 concludes the paper.

5.1 Background and Motivation

This section explains the different concepts and challenges of the ad business model for mobile apps.

5.1.1 Ad Serving Process

The basic ad business model has five main players, as illustrated in Figure 5.1:

(a) **Advertiser**: The company that strives to promote its brands.

(b) **Advertising company**: A specialized company that brings the app developer and the advertiser into contact. The advertising company is in charge of creating ad libraries and delivering ads to the apps.

(c) **App developer**: The software developer that creates an app, and incorporates the ad libraries into their mobile app to request ads.

(d) **App**: The app developer's product. App developers integrate ad libraries to generate revenue from their free-to-download apps.

(e) **App user**: The end user who views, and potentially clicks on an advertisement in the app.

The ad serving process has the following steps, also illustrated in Figure 5.1:

1) The advertiser pays the advertising company to promote its brand (product) across mobile apps.

2) The app developer signs a contract with the advertising company. The contract is basically an agreement in which the advertising company agrees to pay the app developer for delivering ads to the users of the developer's app.

3) The advertising company provides a specialized library called ad library. The ad library has the required APIs to request ads from the advertising company.

4) The app developer adds the ad library to his/her app. The app developer designs/codes conditions about how and where an ad will be displayed in his/her app.

5) The app user downloads (commonly) the app from an app store (*e.g.*, the Google Play app store), and installs the app into his/her mobile device (*e.g.*, Android smartphone).

6) The app user requests an ad when he/she uses the app (when the condition coded by the app developer happens, *e.g.* to start the app).

7) The advertising company sends an ad to the app on the app user's device.

8) The advertising company pays to the app developer based on the number of ads displayed (Cost Per *Mille*, thousand), if the app user clicked on an ad (Cost Per Click), or if the app user completed an action (*e.g.*, to fill a survey) after an ad is displayed (Cost Per Action).

Although app developers only need to integrate an ad library into their apps, ad libraries introduce new maintenance challenges in order to keep on generating revenue. The rest of this section describes the different revenue models for ads, and the challenges that app developers have to face in order to keep on generating revenue from the ad libraries included in their apps.

5.1.2 Ad Revenue Models

The basic money flow in the ad business model sends money from an advertiser (the one with a product/service to promote), to an advertising company (the one with the technological infrastructure to distribute ads), to the app developer (the one willing to sell ad space in his/her apps). Three pricing models are currently used, all of which were inherited from web advertising (Alexandrou, 2013; Mackenzie, 2012; on Mobile, 2010; Strube, 2012):

(a) **CPM**: Cost per thousand (the M stands for *mille* which means thousand in Roman numerals). App developers receive a payment for each 1,000 ads displayed.

- Upside: app developers only have to pull ads from the advertising companies' servers, and wait for app users to see them.

- Downside: this payment system generates the lowest revenue.

(b) **CPC**: Cost per click. App developers receive a payment for each click on an ad.

- Upside: this ad pricing model generates a higher revenue than CPM.
- Downside: app users have to click on an ad.

(c) **CPA**: Cost per action. App developers receive a payment for each action completed by an app user (*e.g.*, to install another app, or to sign for a credit card). Other pricing models have been derived from CPA such as: CPL (Cost per Lead), CPV (Cost per View), and CPI (Cost per Install), where all of them are based on a specific action by the app user (Kaufman, 2013).

- Upside: this is the pricing model that generates the highest revenue for app developers.
- Downside: app users have to complete specific actions, which app users may not be interested in doing (Mackenzie, 2012).

It is estimated that on average an app developer can make $1,498 U.S. monthly from only ad revenue (VisionMobile, 2012). However, some successful apps can make even $1 million U.S. monthly (Weintraub, 2010).

5.1.3 Different Ad Serving Models

The previous section shows how the more complex ad revenue models yield higher revenues, yet depend substantially on a large number and variety of high quality ads to serve to app users. This section explains why getting such high quality to serve is a challenge and how the developers try to address this challenge.

Increasing fill rate

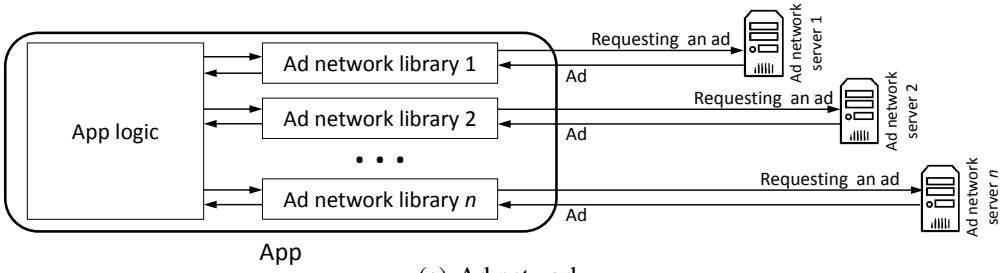
In order to display ads, app developers use the APIs available in the ad libraries to send a request to the advertising companies' servers to request for an ad. The percentage of ads received from the advertising company relative to the number of requests made is called fill rate (Adfonic, 2012). If all the requests for ads are successful, then the fill rate is 100%. However, in the first half of 2011 the average fill rate for the worldwide top 40 ad networks was less than 18%. This low fill rate is mainly because the number of ads being requested by apps are increasing faster than the number of ads available (Candeias, 2011). Furthermore, ad networks that tend to pay more tend to have low fill rates, while ad networks with the highest fill rates tend to pay less (Opera, 2013).

To deal with the low average fill rate, app developers can decide to fill the ad space in their apps with unpaid ads known as house ads (Burstly, 2012). Such an ad typically is created by the app developer in order to fill the ad space (Organized Shopping, LLC, 2012), and can contain anything (*e.g.*, advertise another app created by the app developer) (Google, 2013*b*).

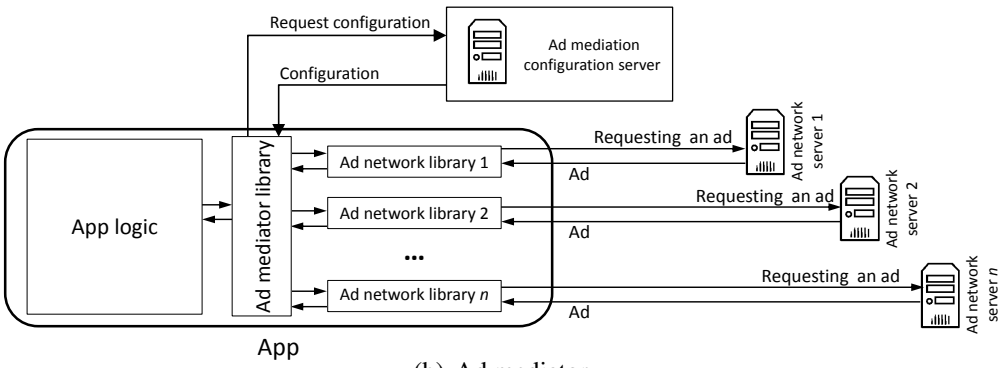
More frequently though, instead of just filling up ad space with unpaid ads, app developers try to integrate ads from different advertising companies, *i.e.*, get ads from different ad network libraries, and/or mediate them through ad mediator or ad exchange libraries. We now discuss the three major types of ad serving models used today in practice.

Ad network

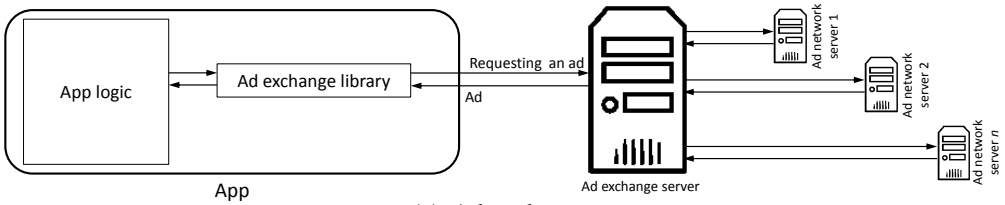
An advertising company can be seen as an ad network, an intermediary between advertisers and app developers. Ad networks sell ad space to the advertisers in order to fill with ads. Then the ad network offers these ads to the app developers. Ad networks distribute ad



(a) Ad network



(b) Ad mediator



(c) Ad exchange

Figure 5.2: The three ad serving models from an app's point of view. (a) Ad network libraries are bundled in an app to request ads from each ad network. (b) An app with an ad mediator and ad network libraries. The ad mediator library coordinates the requests of ads via the ad network libraries. (c) An app has one ad exchange library. The ad exchange library communicates with the ad exchange's server which request ads from varies ad networks on behalf of the app.

libraries to the app developers so that app developers can access the ads. App developers sign up with an ad network that they want to integrate in their apps, and manage the administration (both financial and technical) through the particular ad network's web site. If

app developers want to work with more than one ad network in order to get access to more ads (for generating more revenues), they have to sign up independently with each new ad network (Saverio, 2010).

- From a financial perspective, app developers obtain their revenue directly from each of the ad networks separately.
- From a technical perspective, app developers have to request ads specifically from each of the ad networks in their code. Additionally, each time an app developer wants to add or remove an ad network, they have to update their app to add or remove the corresponding ad library.

Figure 5.2(a) shows an app developer interacting with different ad networks by integrating different ad libraries. Examples of ad networks are AdMob, and Millennial Media.

Ad mediator

An advertising company can instead of (or in addition to) creating ad network libraries, create ad mediator libraries. An ad mediator library mediates between different ad networks and app developers. They are similar to ad networks in the following aspects: app developers still have to sign up with different ad networks, app developers still have to manage individual administration with each ad network, and still integrate the different ad libraries from the ad networks that the app developers want to bundle in their apps. The main difference between an ad network and an ad mediator is that app developers only generate requests for ads based on the ad mediator library instead of requesting an ad for each ad network library without limitations. App developers configure a priority order to each ad network in order to request an ad. This configuration is usually through the ad mediator's website

(commonly this configuration is fetched by the app from the ad mediator's server before requesting an add [1]). Then, the ad mediator library is in charge of requesting an ad to each ad library based on the priority order assigned, *i.e.*, if the ad network with highest priority does not provide an ad, then the ad mediator will request an ad to the following ad network based on the priority order. App developers receive their pay through the distinct ad network companies (Saverio, 2010).

- From a financial perspective, app developers obtain their revenue directly from each of the ad networks separately.
- From a technical perspective, app developers have to integrate as many ad networks as they want to integrate in their apps. However, app developers only request ads based on the ad mediator library. App developers still have to update their app each time that they want to add, remove or update an ad network library.

Figure 5.2(b) shows an app integrating an ad mediator library and distinct ad network libraries. One of the most popular ad mediators is AdWhirl.

Ad exchange

Finally, advertising companies can create ad exchange libraries. Ad exchanges also mediate between different ad networks and app developers. They are similar to ad mediator libraries in that they mediate with distinct ad networks. However, app developers only have to integrate the ad exchange library, they do not need the distinct ad network libraries. Also, app developers only have to sign up with the ad exchange company that they want to work with. All the administration is through the ad exchange web service where (commonly) app developers can choose among different ad networks from which they want to receive

[1] Adwhirl FAQs: `http://code.google.com/p/adwhirl/wiki/FrequentlyAskedQuestions`

ads. The ad exchange company pays directly to the app developers (Saverio, 2010). Ad exchanges usually have revenue sharing agreements with the different ad networks in order to resell (redistribute) the ad network's inventory. Ad exchanges obtain their revenue from these agreements (Mobclix, 2005) [2].

- From a financial perspective, app developers obtain their revenue directly from the ad exchanges.
- From a technical perspective, app developers only need to integrate one ad exchange library.

Figure 5.2(c) shows an app integrating an ad exchange library and interchanging messages with the ad exchange web service that communicates with the different ad networks. Mobclix and Smaato are examples of ad exchanges.

The differences between the three ad serving models can be summarized in the following characteristics:

- Financial. The entity that pays app developers.
- Revenue share. Wether or not app developers have to share part of their revenue.
- Ad scheduling. Where app developers determine the order of requesting ads (in the app code, in the ad mediator's server, or in the ad exchange's server).
- Ad serving source. Who provides the ads.

Table 5.1 presents a breakdown of the different characteristics for each one of the three ad serving models.

[2] http://goo.gl/VznGJ

Table 5.1: Breakdown of the different ad serving models

Ad serving model	Financial contact	Revenue sharing	Ad scheduling	Ad serving source	Example
Ad network	Ad network	No	App level	Ad network	Google Ads
Ad mediator	Ad network	No	Ad mediator	Ad network	AdWhirl
Ad exchange	Ad exchange	Some ad exchanges	Ad exchange	Ad exchange	Mobclix

5.1.4 Software Engineering Challenges for Ad-Supported Apps

Unlike other type of libraries, app developers have to deal with certain problems only found in ad libraries. App developers are facing the challenges of keeping on generating revenue and not losing app users due to the bad reputation of ads. This section examines some of the challenges that ad libraries bring to app developers.

- To keep on generating revenue

- **Different type of ad serving models to choose.** App developers have to choose between three ad models in order to serve ads: (a) ad network, (b) ad mediator, and (c) ad exchange (Saverio, 2010). Each ad model has its own advantages and disadvantages.

- **Low number of ads available.** App developers have to compete for ads, given that the demand of ads by the number of apps available has increased faster than the number of ads available to serve (Candeias, 2011).

- **Time between ad library update and publication in an app store.** App developers have to redeploy their apps every time they need to add, remove or update an ad library.

Then, (commonly) they have to submit the new app version to a particular app store, and on some mobile platforms wait for the approval of the new version of the app by the app store (Ferrell, 2012). For example in Apple's App Store the average approval time is 6 days (Development, 2013), while in Google Play is practically immediately after submitting the app (Johnson, 2012). Finally, app users have to obtain the update of the new app version. All this time may impact the total revenue that app developers can generate from the ads in their apps.

- Not losing users due to the ads

- **Balancing between app environment and ads.** The presence of ads in an app is *unnatural*. Apps by themselves do not require the ads, nor do app users require them either. However, app developers must use ads in order to generate profit from their apps. Hence, app developers try to keep a balance between the environment of their apps and the presence of ads in them, in order not to disturb the app users' experience with the ads. For example, app users may not like ads because of the ads by themselves(eMarketer, 2012), or because of the costs that the ads implies such as battery consumption (Pathak et al., 2011), or the data consumption (Vallina-Rodriguez et al., 2012).

- **App reputation.** The use of ad libraries is not free of risks. Ad libraries can imply security risks, leaking of personal information (Virki, 2012), or legal issues (Cleff, 2010; Oswald, 2011). Unlike most libraries, all ad libraries need access to the network. Often app developers request access to personal information in order to request ads that will be more likely clicked by an app user. A poorly designed or malicious ad library could leak or misuse personal information about the app user (Cleff, 2010;

Oswald, 2011; Virki, 2012). A bad decision by app developers about the right ad library to use may ruin the reputation of their apps. Also, app developers have to consider the location in the screen where the ads are displayed when the app is in use, as well as when to display an ad, in order not to disturb the app users and at the same time maximize the likelihood of app users clicking an ad, and consequently increasing the revenue of the app developers (PlacePlay, 2012).

Given the importance of the ads in the app business model and the different challenges that ad maintenance represents, we sought to study ad library maintenance across free-to-download Android apps in the Google Play app store.

5.2 Study Design and Methodology

In order to study the ad library maintenance, we analyze 519,739 app versions distributed across 236,245 different Android apps. We chose to conduct our study on Android apps, because the Android platform leads the market of smartphone devices worldwide. Today, smartphones running Android have a market share of more than 68% of all smartphones around the world (16.9% iOS by Apple, 4.8% by Blackberry, 3.5% by Windows, the rest of the market share is distributed among Symbian, Linux, and others) (AP, 2012).

5.2.1 Crawling of the Google Play App Store

We crawled the official Google Play app store using an open source library (API Android Market [3]), once a day during the first half of 2011, and twice a day during the second half of 2011. As a result, we obtained a set of 281,079 mobile apps distributed in 625,067 app versions. Google Play classifies the apps in 27 different app categories (8 subcategories in

[3] Android Market API: `http://code.google.com/p/android-market-api/`

"Games"). The crawling ignored two categories, "Live wallpapers" and "Widgets", and two identical named subcategories under the "Game" subcategory. For more details about this process a technical note (Dienst and Berger, 2012) can be consulted. From this process, we obtained two data sets:

- **Android Packages (APK)**: The binary, in the Android specific format APK, for each free to download app crawled.
- **Metadata**: Specific information for each app version. For the present work, we analyzed number of raters, rating and version number.

5.2.2 Extracting Bytecode from APKs

App developers embed in the code of their apps the ad libraries in order to use the APIs (application programming interfaces) of the ad libraries. Hence, in order to study the ad libraries in the Android apps, we need to recover this information (the type of ad libraries embedded in the apps) available in the bytecode of the apps. Hence, we first used an open source tool [4] to extract the Java bytecode from the APKs. Then, we used the *Apache bcel* library [5] to extract for each app the fully qualified class name (package or namespace in which a class is contained and the class name) and its set of method names (APIs) for each class. We stored the fully qualified class names and their corresponding methods in a database (later on called DB1) for later analysis. We use this data to identify the ad libraries in an app.

After this process, we ended up with 519,739 app versions of 236,245 different apps.

[4] dex2jar: `http://code.google.com/p/dex2jar`
[5] Apache bcel library: `http://commons.apache.org/bcel/`

5.3 Case Study

In this section, we present the four research questions that were introduced at the beginning of this chapter. For each research question, we present the motivation for why we looked into this question, the approach we used to analyze the data to answer this question, and the results of our analysis.

5.3.1 RQ1. What percentage of the free-to-download apps are ad supported?

This question is exploratory in nature. The goal of this RQ is to find the set of ad libraries in the free-to-download Android apps, and understand the distribution of the ad libraries across the different categories of apps.

What is the percentage of ad-supported apps?

Motivation: This preliminary question has two purposes: (a) determine if the ad business model is popular among app developers, (b) identify the different ad libraries embedded in the free-to-download Android apps.

Approach: We filtered the fully qualified class names of all versions of the studied apps, using the following regular expression [aA][dD] (*e.g.*, *com.**Ad**libraryName.**Ad**className*). We did this process across all the studied apps in DB1 in order to find the *ad* key word. Due to the very basic regular expression that is used ([aA][dD]), many class names contained a word including this regular expression that did not correspond to our key word (*i.e.*, *ad*). Hence, we needed to manually clean up the matched results.

We grouped and sorted the fully qualified names of the matched classes according to their popularity. Then, we performed the following algorithm to filter the ad libraries:

(a) For each matched class name, we performed a web search of the package name in order to find the website of its ad library provider (the advertising company). This enabled

us to verify that the name of the libraries found were in fact from a real advertising company. We expect that information about the ad library should exist if this is a real and trusted ad library because an app developer needs to register online and sign a contract with the advertiser company in order to receive ads, and later payments.

(b) If the ad library website is found, we add the package name to our set of ad libraries, otherwise we discard it.

We repeated this process for each identified class. The process was repeated for all class names with at least 200 matches. This means that any library not included is in no more than 200 versions of an app (0.08% of the studied apps). We stopped at this point because there were still thousands (802,012) of fully qualified class names to verify that would not necessarily be part of an ad library. If they were, the presence of such an ad library would be in a very small number of apps.

Results: **Ad libraries play a prominent role in free apps.** 51.21% (120,981 out of 236,245 apps) of the free to download Android apps have at least one ad library. We identified 72 different ad libraries that we will use throughout this study. Also, we recovered from the ad libraries' website the last version of each of these 72 ad libraries.

How many ad libraries are embedded into an app?

Motivation: It is clear that ad libraries are important from a financial perspective. However, it is not clear how important ad libraries are at code level. Therefore, we examine the number of ad libraries that app developers have to maintain within an app. Hence, we count the number of ad libraries embedded into an app. The number of ad libraries in an app gives an estimate of the number of ad libraries that an app developer might have to update in future updates of the app.

Approach: We first determine the number of ad libraries in an app. We count how many of

the 72 ad libraries (previously identified) an app has.

Results: **Ad-supported apps can contain a large number of ad libraries.** At least 42,206 (34.88%) of the ad-supported apps have two or more ad libraries. Figure 5.3 shows a breakdown of the number of apps based on the number of ad libraries bundled in the apps. We observe that the curve in Figure 5.3 is a uniformly decreasing curve. There are few apps with a large number of ad libraries. Most apps have only one ad library in them. However, app developers are also bundling more than one ad library in their apps probably to achieve a high fill rate. We even find eight apps with up to 28 ad libraries.

What is the percentage of ad-supported apps across the different app categories?

Motivation: When app developers submit their apps to an app store, they choose a specific category in which their apps will be classified. From this question, we are interested to know which categories have the highest percentage of ad-supported apps, *i.e.*, *is there any app category that depends more on the ad business model?*

Approach: First, we need to identify which apps are using an ad library. In order to accomplish this, we mapped the ad libraries to the apps. We mapped an app with an ad library based on the set of fully qualified class names previously extracted from the APKs in subsection 5.2.2, and the package name of an ad library obtained in subsection 5.3.1. We matched an app with an ad library if the app has at least one fully qualified class name with the package name of the ad library.

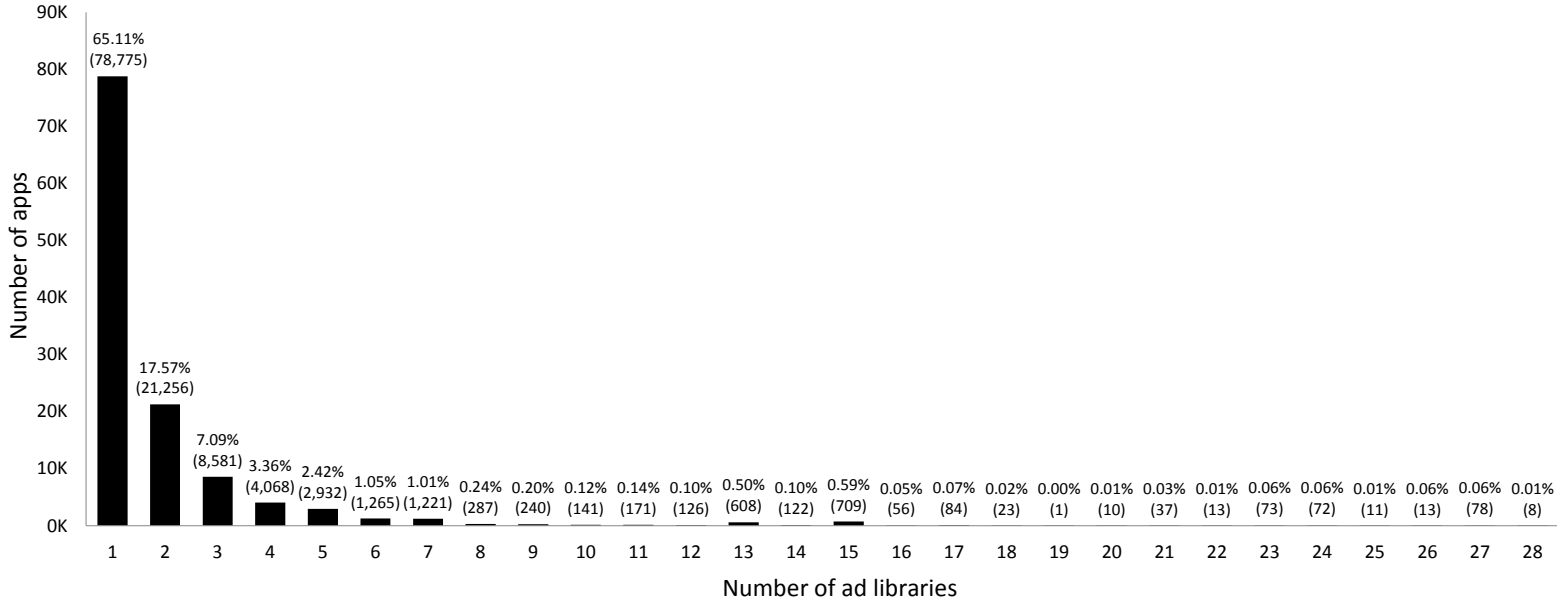


Figure 5.3: Breakdown of the percentage of ad-supported apps (*y*-axis) based on the number of ad libraries bundled in them (*x*-axis).

Results: **The percentage of ad-supported apps varies across the different app categories.** The app categories that rely more on the ad business model are the "Photography" and "Game" categories having 78.1% and 77.8% of ad-supported apps respectively. Figure 5.4 shows the distribution of the ad-supported apps across the 25 app categories. We find that 10 out of 25 categories have at least 50% of their apps with at least one ad library.

Interesting, four out of the top 10 categories with the highest percentage of apps with ad libraries ("Games", "Entertainment", "Personalization", and "Media & video") are among the top 10 most downloaded categories (Android, 2011). However, there are six categories among the top 10 most downloaded ("Tools", "Communication", "Productivity", "Music & audio", "Social", and "Travel & local") (Android, 2011), that have low percentage of apps with ad libraries. Hence, we can suggest that app developers create ad supported apps in categories more frequently downloaded, in order to increase their possibilities of generate higher revenues from ads. Also, we make a recommendation to app users about being cautions when they install free to download apps from categories with high percentage of ad supported apps because the different problems that they can face for the use of apps with ad libraries, such as leaking of personal information (Virki, 2012), high battery consumption (Pathak et al., 2011), and data traffic (Vallina-Rodriguez et al., 2012).

The six subcategories of "Games" are among the subcategories with the highest percentage of ad-supported apps. Table 5.2 presents a breakdown of the "Games" subcategories. The "Racing" subcategory has 90.65% of ad-supported apps. The "Cards & Casino" subcategory has the least percentage of ad-supported apps. This is probably because apps in "Cards & Casino" are using more frequently the "freemium model" where users will have to make in app purchases like virtual currency. However, even with this option, still 71.02% of the studied apps in this subcategory have ad libraries.

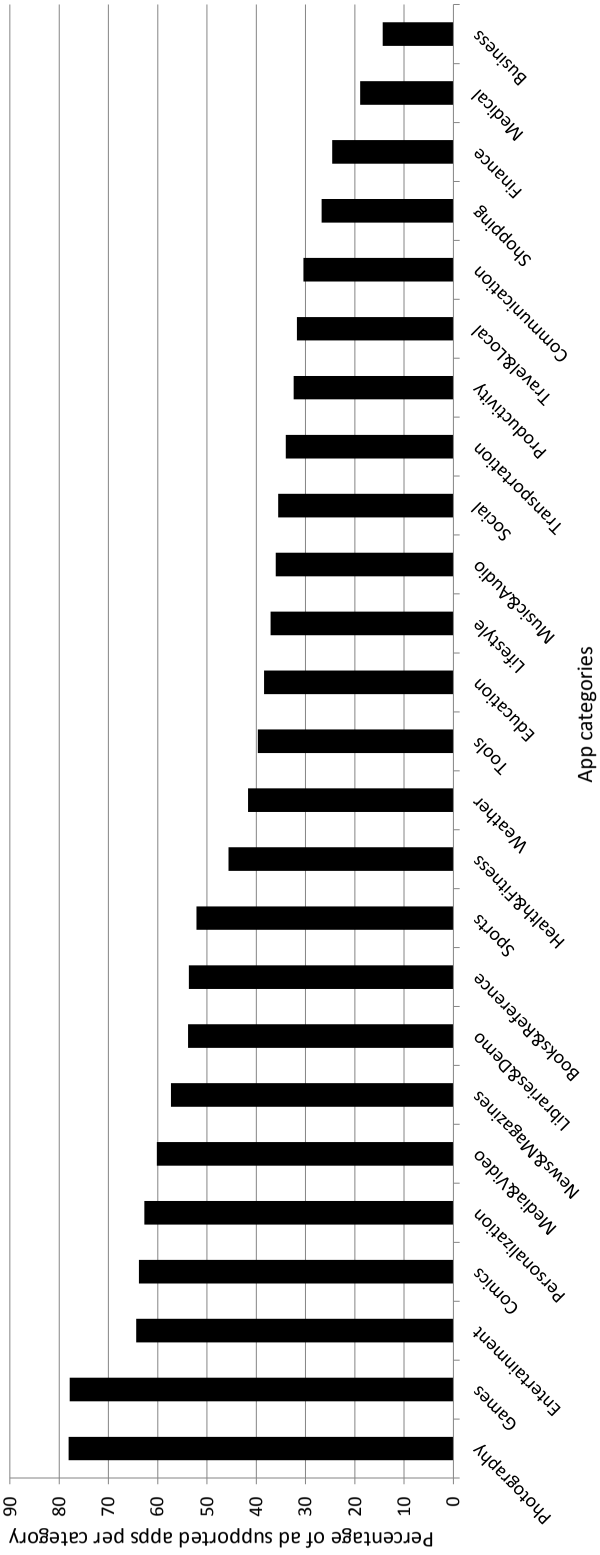


Figure 5.4: Percentage of ad-supported apps for each category. Categories are sorted descending by the percentage of ad-supported apps.

Table 5.2: Breakdown of the Game subcategory for apps with at least one adlibrary

Subcategory	**Apps with at least one ad library**
Racing	1,736 (90.65%)
Sports	2,745 (83.13%)
Brain&Puzzle	9,605 (79.52%)
Arcade&Action	8,937 (76.07%)
Casual	6,532 (74.96%)
Cards&Casino	1,738 (71.02%)

Surprisingly, the "Shopping" category is among the top four categories with the least amount of ad-supported apps, with only 26.72% of the apps having ad libraries. Reviewing the "Shopping" category in the Google Play app store (Google, 2008), we found that the apps were designed by/for specific brands. This implies that they only require to advertise their own brands. This is one of the reasons why they do not add so many ad libraries to the apps in this category. The other three categories with fewer than average ad-supported apps are: "Finance" (24.57%), "Medical" (18.9%), and "Business" (14.3%). The "Business" category is also associated with specific brands of businesses that may not want to advertise and spoil the app user experience with ads. They also could be using the freemium model.

Ad libraries play an important role in the free-to-download apps. About 51% of the free-to-download-apps are ad-supported apps, and about 35% of the ad-supported apps have two or more ad libraries. An app can contain as much as 28 ad libraries. The percentage of ad-supported apps varies according to the category of an app from 14% of ad-supported apps in the Business category, up to 78% of ad-supported apps in the Photography category.

5.3.2 RQ2. Which ad serving model is more prominent?

The goal of this RQ is to understand the current practices of app developers about which ad libraries they are most commonly using, and which ad serving model they prefer to implement in their apps. First, we explore the distribution of the ad libraries across the free to download Android apps. Then, we analyze if there is an ad serving model that predominates across the free to download Android apps.

What is the ad library distribution across the apps?

Motivation: In RQ1 we uncover 72 different ad libraries embedded in the free to download Android apps. This question is a preliminary step to discover if there is a ad serving model that dominates in the free to download Android apps. From these results ad library providers can quantify the impact, and extent of use of their ad libraries embedded on the free to download Android apps.

Approach: From the mapping of ad libraries created in subsection 5.3.1, we count how many ad-supported apps in its last version use each one of the 72 ad libraries. We present the percentage of apps that use each ad library.

Results: **A small number of ad libraries are used frequently throughout the Android market.** Only three ad libraries are embedded in more than 10% of the ad-supported apps. As we can see from Figure 5.5, most of the apps use only a small subset of the 72 ad libraries under examination. *Googleads*[6] 53.39%, *admob*[7] 29.27%, and *millennialmedia*[8] 10.09%, have more than 10% of the apps under examination. Of these three, the two top ad libraries belong to Google. In the other extreme, *mobgold*[9] with only 50 apps using it, is among the the least popular ad libraries for app developers.

[6] http://www.google.com/ads/
[7] http://www.google.com/ads/admob/
[8] http://www.millennialmedia.com/
[9] http://www.mobgold.com/

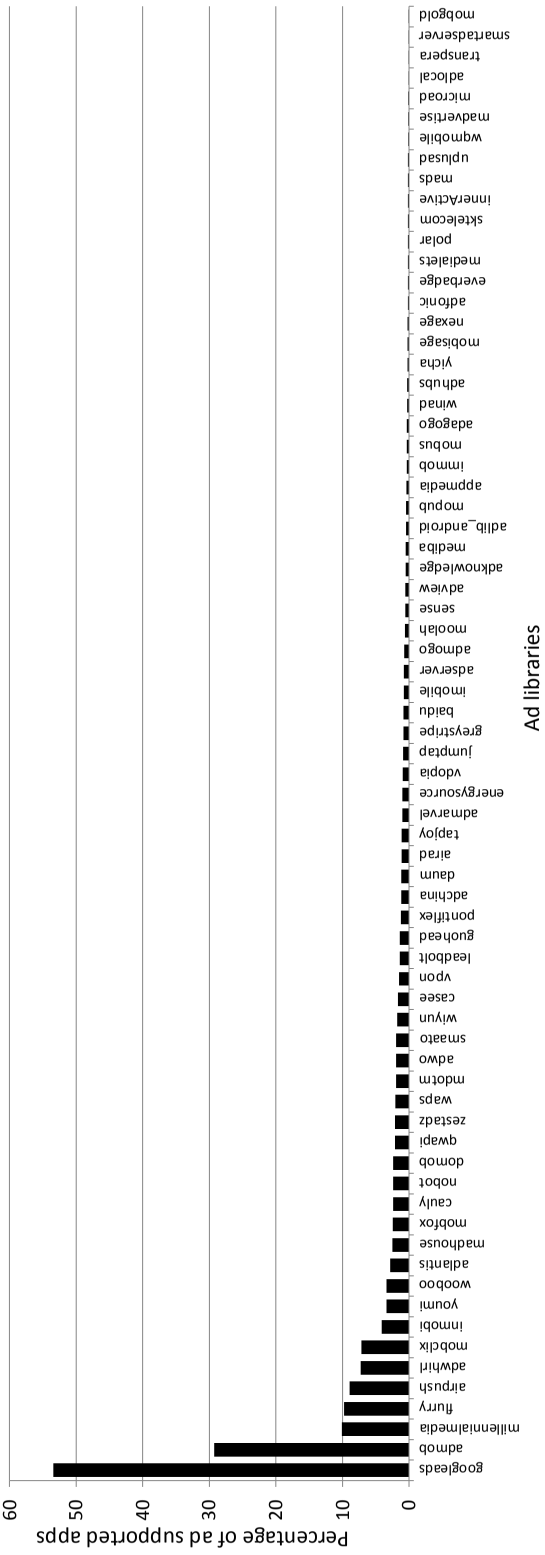


Figure 5.5: Percentage of the 72 most used ad libraries in 2011. The ad libraries are sorted in descending order by the percentage of apps that use them.

The median in Figure 5.5 is shared by two ad libraries (because of the even number of ad libraries in our analysis), *i.e.*, *jumptap* (0.87%, 1,056 apps), and *greystripe* (0.86%, 1,045 apps).

What is the ad serving model distribution?

Motivation: In order to understand the current practices of using ad libraries by the app developers, we analyze which ad serving model dominates across the ad-supported apps.

Approach: We identified the ad serving model for each of the 72 studied ad libraries: ad network, ad mediator, or ad exchange. The identification of the ad serving model for each of the 72 studied ad libraries was based on information gathered from the ad library's website. Then, we clustered the last version of the ad-supported apps based on the ad serving model they use. The ad serving model used by an app is given by the type of ad library embedded in the app. We use the following rules to cluster the apps:

(a) Ad network. Apps with at least one ad network library, but no ad mediator libraries. These apps may have ad exchanges libraries.

(b) Ad mediator. Apps with at least one ad mediator library.

(c) Ad exchange. Apps with at least one ad exchange library.

Given that there is no way to identify whether or not an ad network is being handled by an ad mediator library, and ad mediators are constantly adding new ad networks to their list of ad networks that they can mediate, we include in the count of ad mediators every app that has at least one ad mediator library, and we excluded it from the ad network counting. In the other hand, the ad network counting, and ad exchange counting may overlap because some apps might be using both ad serving models.

Results: **The ad network serving model dominates drastically the ad market.** 61 out of the 72 (84.72%) of the ad libraries identified work as ad networks, 6 (8.33%) are ad mediator libraries, and 5 (6.94%) are ad exchange libraries.

If we look at the percentage of apps using each kind of ad library, we find that the ad network serving model is the most popular model. 88% (106,471) of the ad-supported apps (120,981 apps) are using the ad network serving model. App developers prefer to work with the ad network serving model in spite of providing only one source of ads in comparison to the other two ad serving models, and the increased complexity of requesting ads for each ad network. Probably, the logic behind this beahviour is that app developers are preferring to obtain better quality of ads that are more appealing to be clicked by the app users (de Souza, 2011).

The next most popular ad serving model is the ad exchange, having 11.63% of the ad-supported apps. We assume that app developers prefer this ad serving model over the ad mediator serving model because app developers only need to add one ad library, having multiple sources of ads that can be configured from the advertiser's website. However, this ad model has as a disadvantage that being an intermediary between the ad networks and the publisher (usually app developers), the ads are not premium ads (ads more appealing to the app user by factors such as personalization, or quality) as in ad networks. Ad networks typically retain premium ads instead of giving those ads to ad exchanges (Gold, 2007). *Ad exchanges* are usually good to achieve higher fill rates, but the CPM is low (Miles, 2013). Then, with less attractive ads, the click rate probably will not be as high as with a premium ad. We also found that some ad exchanges take part of the app developers' revenue. This is at least the case for one exchange not found in our set of apps, the Nokia ad exchange (Nokia, 2012).

Finally, the least popular ad serving model is the ad mediator, having 9.92% of the ad-supported apps. The difference between the ad mediator and the ad exchange serving model is minimal, only 1.71%. One of the most popular ad mediator libraries is *adwhirl*, which is an open source ad mediator library by Google that does not require to share any revenue. However, Google will no longer give support to this ad mediator library. Google has released a new version of the *admob* ad library that will work as an ad exchange library [10] [11]. The new version of the *admob* ad library will allow app developers to choose different ad networks and allocate ad traffic through a web interface. Hence, app developers have to perform ad maintenance to update their apps to this version.

Discussion: In the present RQ we discovered that only a few ad libraries are widely embedded in the free to download apps. We wonder, what motivates app developers to pick specific ad libraries. We identified some of the factors in picking an ad library:

(a) **Revenue.** App developers discuss their experiences in forums online about the different ad libraries. They try to find out which ad libraries generate the highest revenue (*i.e.*, high fill rate, high CPC) [12] [13] [14], given that these values vary across ad providers (Mackenzie, 2012; mobiThinking, 2012). From these discussions app developers opt to use certain ad libraries.

(b) **Premium ad networks.** There are advertising companies that focus their effort on leading brands and top publishers (apps). This situation reduces the scope of the number of apps that use these ad libraries, in particular because ad providers will try to

[10] support.google.com/admob/bin/answer.py?answer=2423687

[11] https://support.google.com/admob/faq/2994346?hl=en

[12] Making Money with Android: http://forums.makingmoneywithandroid.com/advertising-networks/

[13] Stackoverflow - Embedding ads on Android app?: http://stackoverflow.com/questions/2471417/embedding-ads-on-android-app

[14] OnStartups - How to increase ad revenue: http://answers.onstartups.com/questions/33530/how-to-increase-ad-revenue

send ads to top apps (mobiThinking, 2013). An example of a premium ad network is *Medialets* (Moby affiliates, 2012).

(c) **Regional vs global.** Some ad companies focus their effort on specific countries, while others distribute their ads to anywhere. Ads that focus on particular population sectors are more likely to be clicked on by that population. An example of a regional ad library is *Wooboo* (China) (MobyAffiliates, 2013), whereas *googleads* is a global library (Moby affiliates, 2012).

(d) **Trust.** Some ad companies have generated mistrust among app developers, in particular for lack of paying. Hence, app developers do not continue using these libraries. One example of an ad library that has generated mistrust among app developers is *mobgold* (Ruckman, 2012), the least used ad library among those that we analyzed.

(e) **Type of content.** Certain ad libraries deliver specific content focus on a specific population (*e.g.*, only for adults), or the ads are delivered outside the scope of the app (*e.g.*, in the notification bar in an Android device) (Moby affiliates, 2012). App developers may opt whether or not to include these libraries depending on the market for their apps because of reputation of their brands (Chien, 2012). One example of an ad library that sends ads to the notification bar is *airpush* (Spring, 2011).

(f) **Platform supports.** Some ad companies may not support ad libraries or have ads available for specific platforms (Siang, 2011)

Besides the previous factors that motivate app developers to choose a specific ad library, we found that the top two ad libraries, *googleads* and *admob*, are developed by the same company that develops the platform (Android) on which these apps run, namely "Google" (Admob, 2013; Google, 2013*c*). For example, *Admob* was an already established

company in the market of mobile ads, before "Google" acquired it (Wojcicki, 2010). Notice that adwhirl, the sixth most popular ad library, also belongs to "Google" (Google, 2013*a*).

The ad network serving model is the most prominent across the ad-supported apps. About 88% of the ad-supported apps use this ad serving model. The ad exchange, and the ad mediator serving models have similar share of apps, 11% and 9% respectively.

5.3.3 RQ3. How are the ad libraries maintained in the ad-supported apps?

This RQ analyzes the current situation of ad maintenance among the ad-supported apps. First, we analyze what is the amount of change in an app related to ad library changes. Then, we present an analysis in order to identify the ad serving model that reduces the number of changes required.

What is the percentage of ad library change in an app?

Motivation: Because of the large number of apps, the competition between the different apps is extremely high. Hence, the time of release/upgrade plays an important factor. In particular, given that for each upgrade, or addition of a new ad library (section 5.1.4), app developers have to redeploy their apps in an app store. The time factor for each upgrade of an app may mean that an app developer might be losing money. For example, if an advertising company releases a high priority upgrade of its ad library, every app developer that uses that ad library has to update his/her apps. Hence, we are interested to know how much change because of the ad libraries is happening in the mobile apps.

Approach: Given that we are analyzing the changes of ads through different versions we need ad-supported apps with at least two versions. Here, we filtered the set of ad-supported apps to get only those with two or more versions. Furthermore, we present in our study the problems that a poor ad library maintenance can carry (RQ4). One of the problems we

study is how ad libraries affect the app rating (the rating that app users assign to an app). The rating of an app depends on the app users' perception. App users rate an app from 1 to 5 "stars" as in any other collaborative online recommendation system (Adomavicius and Tuzhilin, 2005).

However, the rating of an app is subject to rater bias (Chen and Singh, 2001; Harbich and Auer, 2005). Similar to findings in open source software (Herraiz et al., 2011), the number of raters is a source of bias for the rating of an app version. For example, if no app user rates the new version of an app, this version will inherit the rating of the immediately previous version (Google, 2013*d*), even though the app users' perception about this new version is not very likely the same as the previous version (*e.g.*, the users' perspective on quality of an app version might be lower).

Hence, in order to analyze the ad maintenance and minimize rater bias, we decided to only select ad-supported app versions with more than two versions, and at least 10 raters. Henceforth, this new dataset will be called *ad-supported multi-version*. After the filtering, the ad-supported multi-version dataset has 13,983 versions, distributed across 5,937 different apps.

We create a signature for each ad library and the rest of the code contained in an app in order to identify a variation in an ad library across app versions. This technique is based on the software Bertillonage technique, which creates a set of class signatures in order to measure the similarity between two components of a software system (*e.g.*, two JARs) (Davies et al., 2011). In this case, we look for the similarity between two embedded ad libraries in two different versions of an app. We use the following algorithm to generate signatures for each app:

1) Extract the class name and the method names for each class in an app. For this step we

use the *Apache bcel* library.

2) Identify classes and their list of methods that are part of an ad library. The identification is based on the package name for each class.

3) In order to count any modification in the API related to the ad libraries in an app, with respect to the rest of the API of an app (henceforth called "core code"), we group classes with their list of methods for each ad library in an app, and the core code .

4) Generate a hash value for each fully qualified class name with their list of methods. We apply SHA1 to generate the hash values.

The resulting sets of hash values for each ad library and the core code are their corresponding signatures in the app. Then, we compare the signatures in a particular version of an app (version i) with the following version of the app (version $i + 1$) in our dataset. Here, we can obtain one of the following four cases when we compare the signatures of each ad library between the two app versions:

(a) The ad library is identical in both versions.

(b) The ad library is new in the app version $i + 1$.

(c) The ad library is deleted (removed) in the app version $i + 1$.

(d) The ad library is updated in the app version $i + 1$. By update we mean that the list of signatures of a particular ad library in app version i and app version $i + 1$ are different (they are different even if just one class signature in the ad library is different).

Hence, we define that an app has a type of ad library change if any of the cases (b), (c) or (d) occurs between app version i and $i + 1$.

Table 5.3: Ad related change from the ad-supported multi-version app

Type of ad change	Number of app versions
Updated	4,470 (65.25%)
Added	2,993 (43.69%)
Removed	1,894 (27.64%)

Results: **App developers are frequently changing the ad libraries in their apps.** 6,850 (48.98%) of the studied apps have a type of ad related change. From this, we can indeed see that almost every other time that an app is updated, the app developers change something about the ad library. Table 5.3 presents a breakdown of the different type of ad related change across these 6,850 app versions (given that from one version to the next version, an app may have updated one ad library, added another, and removed another ad library, *i.e.*, the percentages overlap with each other). Of these 6,850 app versions that have a type of ad related change, 65.25% (4,470) of the app versions had an ad library updated, while 43.69% (2,993) of the app versions have at least a new ad library added. However, in only 27.64% (1,894 out of 6,850) of the app versions at least one ad library is removed.

From Table 5.3, we can conclude that app developers are more actively updating, then adding, and finally removing ad libraries.

Another interesting finding is that for 942 apps (13.75%) out of the 6,850 apps with a type of ad related change, the API in their core code (rest of the code, *i.e.*, no ad library classes) were not modified. *I.e.*, 942 apps had a modification in the class and/or method names of the ad libraries embedded in them. However, the core class and/or method names of their code did not have any modification. For example, *View.CalCollection.SangGeon.Cauly*, a calculator for different type of conversions, had seven versions. The ad libraries embedded in this app were modified in its seven versions, but not its APIs in its core code. In all the cases the *adlantis* ad library was updated. Cases like this app show how apps that might

require few updates as a calculator, because of its static functionality, require frequent updates because of the ad libraries embedded in them.

Which ad serving model requires the least number of ad changes?

Motivation: The use of ad libraries adds a new layer of maintenance that app developers have to consider in order to continue generating revenue from their apps. Therefore, even though the addition of many ad libraries would increase the fill rate, the app developer might opt out from doing this due to the maintenance effort involved. Hence, app developers might decide to use only one ad library in order to dismiss the ad library maintenance effort. However, this might lead to a low fill rate, and consequently low revenue per app. Hence, we are interested to know if there is a particular ad model such as one ad network library, multiple ad network libraries, ad mediator, or ad exchange, that leads to fewer ad-related updates.

Approach: Figure 5.2 shows the three different ad serving models: ad network, ad mediator, and ad exchange. With respect to the ads, the code of an app can be separated in two:

- Ad library(ies): The different ad libraries provided by the advertising companies that the app developer embed into his/her app.
- App base code: The rest of the code ("core" code).

For the ad-supported multi-version dataset (13,983 versions, distributed across 5,937 different apps), we counted the number of times that the ad libraries underwent a type of ad related change (added, removed, updated) in each of the different ad models. We calculate the ad change for the following ad models (or their combinations):

- *One ad network library*: We count apps with only one ad network library.

- *Exclusive multiple ad networks*: We count apps with exclusive multiple (two or more) ad networks. These apps do not include any ad mediator, nor ad exchange libraries.

- *Multiple ad network libraries*: We count apps with multiple (two or more) ad network libraries. These apps do not use ad mediator libraries but might use ad exchange libraries. Ad exchange libraries are independent of any other libraries, on the other hand ad mediators require ad network libraries and they are part of a different ad model. However, we decided to count apps that also have ad exchange libraries because given that both ad serving models are independent. We only count changes to the ad network libraries.

- *Ad mediator & ad network libraries*: In this group we included any app that has at least one ad mediator library and at least one ad network library. Given that there is no a way to determine if an ad network is being used independently without an ad mediator library, and ad mediator libraries are constantly adding new ad network libraries in the ad networks that they can mediate, we include in this counting any app that has at least one ad mediator library as part of this ad model integration.

- *Ad exchange library*: We count apps that have at least one ad exchange library.

Results: **The only one ad network model has the least number of ad changes, but the source of ads is limited.** The *only one ad network library* required 36.28% of changes in a new app version. However, for the app developers this model has the disadvantage of having only one source of ads. **The ad serving model with the second lowest number of changes is the *ad exchange*** having 49.94% of changes in a new version. However, this ad model has as disadvantage that being an intermediary between the ad networks and the publisher (usually app developers), the ads are not premium ads (ads more appealing to

Table 5.4: Ad related changes for each ad model

Ad model	Num. App-Ver.	At least one ad modif.	Added	Removed	Updated	No core modif.
One ad network library	6,755	2,451 (36.28%)	384 (5.68%)	93 (1.37%)	1,974 (29.22%)	377 (5.58%)
Exclusive multiple ad networks	5,604	3,637 (64.90%)	2,206 (39.36%)	1,628 (29.05%)	1,939 (34.60%)	454 (8.10%)
Multiple ad network libraries	3,275	2,261 (69.03%)	1,452 (44.33%)	1,084 (33.09%)	1,033 (31.54%)	324 (9.89%)
Ad mediator & ad network libraries	1,936	1,117 (57.69%)	453 (23.39%)	377 (19.47%)	713 (36.82%)	105 (5.42%)
Ad exchange library	1,874	936 (49.94%)	324 (17.28%)	111 (5.92%)	541 (28.86%)	92 (4.90%)

the app user by factors such as personalization, or quality) as in ad networks. Ad networks typically retain premium ads instead of giving those ads to ad exchanges (Gold, 2007). *Ad exchanges* are usually good to achieve higher fill rates, but the CPM is low (Miles, 2013). Then, with less attractive ads, the click rate probably will not be as high as with a premium ad. **The third ad serving model with the least number of changes is the *ad mediator*** having 57.69% of changes required.

Finally, **using multiple ad networks requires the largest number of changes** with up to 69.03% of changes in a new app version. Table 5.4 shows the results of the count of the different ad models.

Discussion: Given that app developers are constantly changing the ad libraries embedded in their apps, we looked at what motivates them to apply these constant changes to the ad

libraries. We found that ad library providers (advertisers) are releasing constant updates of their ad libraries. Then, app developers need to integrate (update) the new version of the ad libraries in their apps. Finally, we present the case of *dead ad libraries*, ad libraries that because of changes by the ad library providers (advertising companies) are not working any more (*i.e.*, are not longer serving ads).

Ad library providers are releasing constant updates of their ad libraries.

We looked for the documentation related to ad libraries in order to understand what motivates app developers to change (add, remove, or update) the ad libraries in their apps. In particular, we looked for any change in the log history of the top ten ad libraries in Figure 5.5. Usually, an ad library is distributed in a compressed file (*e.g.*, rar, zip) that contains the ad library, the documentation of how to use the ad library, and the history log (usually a plain text file) of the different versions of the ad library, among others. We are interested in the history log because it comprises the changes that an ad library has exhibited in each of its versions.

Some of the common reasons why ad library providers are releasing updates of their ad libraries are to improve memory management in order to avoid memory leaks, and ad interaction updates with app users. The ad interaction updates have a direct impact on the amount of revenue that app developers generate from ads. App developers generate more revenue when app users click on an ad than when an ad is only displayed (*i.e.*, more revenue from CPC than from CPM) (Bateman, 2012; Bryan, 2011). Also, from the ad libraries' log history documentation, we observed that ad libraries are adding and/or improving ad video capabilities in their ad libraries. One of the main reasons why advertisement companies are adding video features to their ad libraries is because videos help to attract and engage the audience (Saracino, 2012) and therefore help generate more revenue (Palumbo, 2010, 2012)

for all the stakeholders.

For the *adwhirl* ad mediator library, we consulted its changes history online (Adwhirl, 2012) in order to analyze the type of modifications that this library has gone across its different versions. Because of its ad mediator nature, many of the changes are done in order to maintain compatibility with different ad networks. Given that the *adwhirl* ad mediator library is an open source project, bug fixes are sometimes provided by other ad networks. This was the case by the *inmobi* ad library (Adwhirl, 2011). Also, from the log history included in the package where the *youmi* ad library is distributed, we found that this ad library had to release an update in order to fix a bug that was preventing to the display of ads on Android OS 2.3. Android OS 2.3 is the most popular version that Android devices run today (Android, 2013*a*), given that there are many devices that can not run newer Android OS versions.

Interestingly, we found that part of the ad libraries' updates are with respect to the collection and management of personal information. For example, in the log history of the *millennialmedia* ad library, we observed that one of the updates in 2010 was to add support for collecting data for ethnicity, and sexual orientation. Also, the *flurry* ad library (that also works as an analytic library) added in 2011 a variable type related to gender. The *flurry* ad library was also updated in order to pass conditions of user experience that Google imposed (Flurry, 2012). The log history of the *Mobclix* ad library stated that version 3.2.0 had to release an update that ensure that the transmission of personal information is encrypted. With this update, apps that use the *Mobclix* ad library could be accepted in the Amazon App Store.

In summary, some of the main reason why ad libraries were updated are: to enhance interaction capabilities between ads and app users, integrate ad video functionality, fix

Table 5.5: Type of updates performed on the most popular ad libraries

Library	Number of updates in 2011	Updates added across the different versions released in 2011
Googleads & Admob	6	Transition for merging the *googleads* and the *admob* libraries. Bug fixes to improve app users' interaction with ads, and to handle clicks.
Millennialmedia	9	Ads in video format received several improvements across different updates of this ad-library. Improvements to click events for ad-videos.
Flurry	8	First released for a new ad-product called App Circle. Update to pass conditions of user experience imposed by Google. Check of availability of ads.
Adwhirl	7	Updates to work properly with different ad-networks.
Mobclix	11*	Prevent autoplay ads. Add mediation capability for Google/AdMob SDK and Millennial Media Android SDK. Fixing bugs for memory management, and on clicks. Personal information is encrypted in order to be transmitted.
Youmi	15	Fixing bugs that prevent displaying ads on Android OS 2.3. Lightweight initialization process to display ads.

memory management bugs, add compatibility with other ad libraries, manage personal information, and fix bugs that prevent ads to be properly displayed. We did not find any information for the *airpush*, *inmobi*, and *wooboo* ad libraries. Table 5.5 summarizes the main changes of the remaining top seven ad libraries in 2011. Only the *Mobclix*'s log history does not mention dates (year). The 11 updates are all the updates mentioned in the *Mobclix*'s log history at the time of writing.

Dead ad libraries: We define *dead ad libraries* as ad libraries that remain in the apps without

being able to serve any more ads. We have reviewed the ad library providers' website in order to find valuable information that could give us an insight about this problem.

The mobile ad industry produces billions of dollars annually, and is expected to continue growing (Columbus, 2013; Pettey and van der Meulen, 2013). Given the lucrative nature of the ad industry, it is unsurprising that ad companies are acquired/merged by/with other (ad) companies in order to continue growing as business. We explore the case of the dead ad libraries as consequence of the acquisition of Admob by Google.

Googleads was only focused on search ads (Kincaid, 2009), and *Admob* (a company founded in 2006) (Admob, 2013) was focused on mobile ad displays. In May 2010, Google acquired Admob (Wojcicki, 2010) for $750 million of (american) dollars (Vascellaro, 2009). Previously, Admob acquired Adwhirl (Business Wire, 2009), the sixth most popular ad library as shown in Figure 5.5. However, it was not until the *admob* ad library version 4.0 was released in the first trimester of 2011 (Google, 2012*a*) that the *Admob*'s package name (in the library) was replaced from *com.admob.android.ads* to *com.google.ads*. "Google" warns app developers about this change (Google, 2012*a*):

"Warning: All new Android apps created after October 14, 2011 will require an AdMob SDK that was released on or after March 15, 2011. This corresponds to version 4.0.2+ for Android. If you downloaded the library from our official download site, then you're already set. Otherwise you may have an old version of the AdMob SDK that was released prior to March 15, 2011, and your new app will not receive any ad impressions until you update your SDK." (sic)

We found 35,413 ad-supported apps (29.27%) using an ad library identified as *com.admob.android.ads* as shown in Figure 5.5. If the app developers in charge of these apps do not properly perform the needed ad maintenance by updating the actual *admob* ad library with

the new ad library provided by "Google", they will not generate revenue from this ad library in their apps.

Also, given that the new version of the *admob* library changed its package structure (from *com.admob.android.ads* to *com.google.ads*), app developers can not integrate the library "*as is*". App developers have to modify their own code in order to properly use the APIs in the new version of the *admob* library (Basic4Android, 2011; Google, 2011; StackOverflow, 2011).

App developers are frequently updating the ad libraries embedded in their apps. Using only one ad network library means the least number of ad related changes. However, using only one ad network library has as a downside a reduced source of ads. In contrast, the ad exchange and mediator serving models represents a higher number of changes, but a higher source of ads.

5.3.4 RQ4. What is the impact of the ads on the app user perceived quality?

As we mentioned earlier, ads are *unnatural* to an app. Advertisements can be annoying, and distract the attention from the app (Adobe Systems, 2012). These factors may affect the reputation of an app, which can result in the lose of app users, and consequently app developers may see reduced ad-revenue. However, in a market widely dominated by free-to-download apps, app developers use the ads to generate revenue from their apps. Hence, app developers should monitor constantly the impact of the ads in their apps, in order to fix these type of inconveniences that may affect their revenue. This RQ explores how ads are related to the quality of an app, *i.e.*, the perceived quality of an app. We explore two scenarios: a) the number of ad libraries, and b) specific ad libraries.

Relationship between number of ad libraries and app user perceived quality

Motivation: Previously, we found that almost 35% of the ad-supported apps have two or more ad libraries, we even found apps with as many as 28 ad libraries. Given that ad receivers (*e.g.*, app users, tv viewers) might feel annoyed because of the ads (Adobe Systems, 2012), we are interested in studying whether the number of ad libraries in an app has an impact on the user perceived quality.

Approach: In Google Play, app users can rate an app from 1 to 5 "stars" (5 is the highest mark). Here, we use the rating of an app as a measure of the user perceived quality. However, in the Google Play, the rating of an app is the average rating across all the versions of an app, not a specific rating for one specific version of an app. Hence, we reconstruct its version rating.

In order to reconstruct the version rating of an app we applied the following formula:

(1)

$$version{-}rating = \frac{averRat_i * cumCount_i - averRat_{i-1} * cumCount_{i-1}}{count_i}$$

Where:

$averRat_i$ = Average rating of version i.

$cumCount_i$ = Cumulative number of raters of all versions up until version i.

$count_i$ = Number of raters of version i.

Here, we require apps with two or more version, since from formula (1), it is not possible to obtain a version rating for the first version of the ad-supported apps.

Also, as we mentioned before in section subsection 5.3.3 the rating of an app is subject to rater bias (Chen and Singh, 2001; Harbich and Auer, 2005). The number of raters is an important factor to determine the app user perceived quality. For example, an app version that has not been rated will continue having the rating of the previous version. Hence, in order to minimize this rating bias, we decide to use ad-supported app versions with at least

10 raters.

For this question, we use the ad-supported multi-version dataset to analyze the version rating of an app. This dataset has apps with at least one ad library, more than one version, and at least 10 raters per version (13,983 versions, distributed across 5,937 different apps). With the ad-supported multi app version dataset we are also minimizing a rater bias.

Finally, given that some apps have many more versions than others, if we use every version of an app our results can be affected by apps with high number of versions, and by other intrinsic factors of an app (*e.g.*, type of an app). Hence, for each app we use only one app version. In our case, we decided to use the last version of each app.

We present the results of the version rating of an app based on the number of ad libraries that each app has.

Results: **The number of ad libraries is not related to the app user perceived quality of an app.** Figure 5.6 shows a bean plot of the version rating of apps for the last version in the ad-supported multi app version dataset based on the number of ad libraries that each app has. The bean plots are sorted ascending by the number of ad libraries that each group of apps has. We can see that the the first group of apps with the least number of ad libraries (from 1 to 10 ad libraries) tend to have 4 or higher version rating. Apps with a higher number of ad libraries (from 11 to 28 ad libraries) are a combination between slightly lower or even higher than the median version rating. It is important to notice that in general the group of apps with a high number of ad libraries is significantly lower than the apps with few ad libraries (see Figure 5.3, and this number decreased after filtering apps with low number of raters).

We decided to look at the particular case of the eight apps that have 28 ad libraries. Five out of these eight apps are developed by the same app developer, and the other three by two different app developers. Three of the apps were removed from Google Play app

store for undisclosed reasons. We verified the rating and rater numbers for the remaining five apps in order to verify if app users rejected these apps because of the high number of ad libraries. Surprisingly, those apps still have similar rating (a high average rating), and continue accumulating positive ratings. For example, the app with the largest number of raters of these five apps has 1,489 raters, and an average rating of 4.3 out of 5, with 946 app users rating it with five (stars). Interestingly, the number of users keeps increasing in spite of having a huge number of ad libraries. These apps are specialized on displaying pictures, and classified as "mature only for 18+ viewing".

One way to understand the reason why the number of ad libraries is not a factor for the app user perceived quality is that a high number of ad libraries does not mean that an app displays more than one ad at the same time. The inclusion of a large number of ad libraries is instead a strategy to achieve a high fill rate. Apps with less number of ad libraries (ad sources) may be getting a lower fill rate. However, the ad space (when an app does not get an ad) may be still filled by a "house ad".

Relationship between particular ad libraries and user perceived quality

Motivation: A key principle in advertisement is to stand out, otherwise it is a waste of resources (Daye, 2010). Hence, that ads try to attract the app users' attention. Ads form part of the Graphical User Interface (GUI) of an app. Hence, app developers might have to be cautious about the type of ad libraries that they are including in their apps from the design of their apps, in order to avoid a low rating in their apps because of the ads displayed. Otherwise, app developers will have to perform ad maintenance tasks in order to replace the ads in their apps. We study the relation of specific ad libraries in the rating of apps.

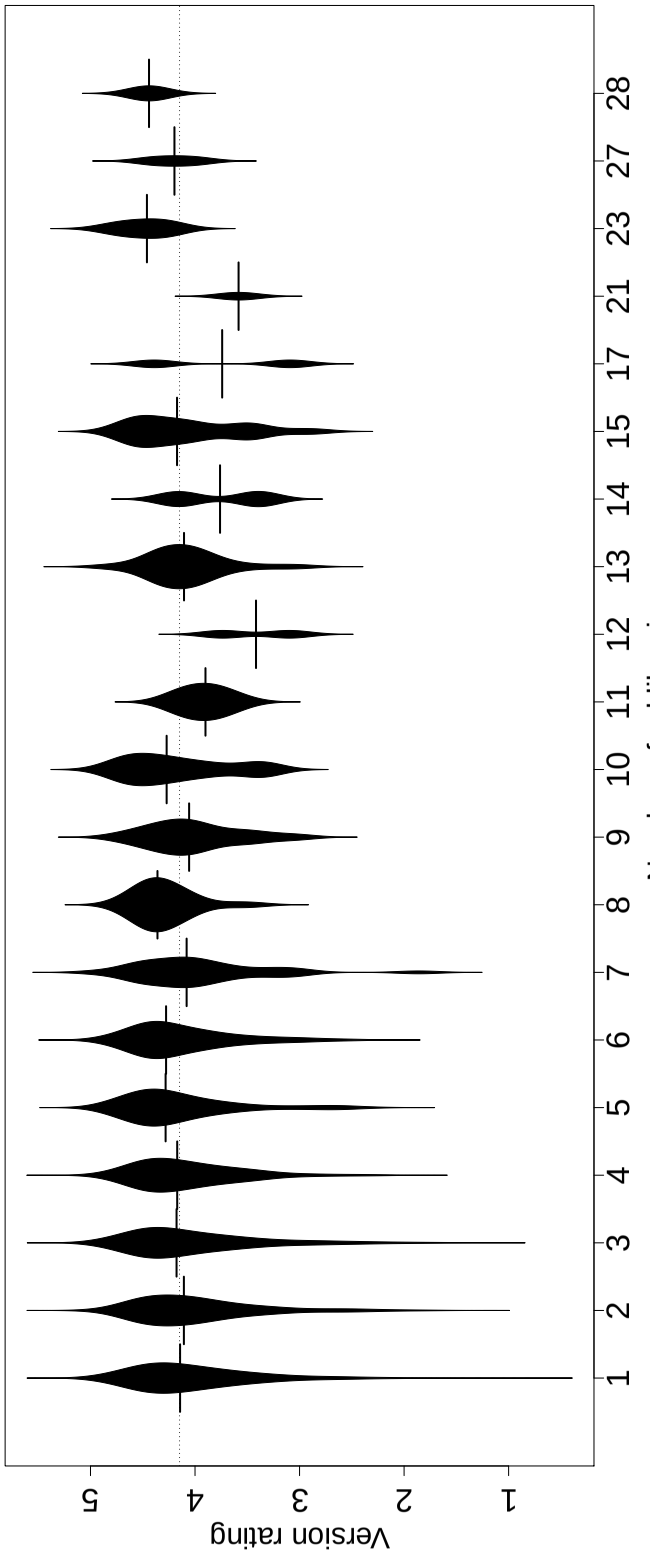


Figure 5.6: Beanplots showing the relation between the number of ad libraries (x-axis) and its version rating (y-axis). Beanplots are sorted ascending by the number of ad libraries. The horizontal lines for each beanplot represent the median of the app version rating. The long line across all the beanplots is the median of the app version rating across all the plotted apps.

Approach: This question uses the ad-supported multi-version dataset, and (as in the previous question) we only use the last version of each app in order to only count once each app instead multiple versions of an app. We queried for the version rating of each app in its last version in the dataset based on the type of ad libraries embedded into them. For example, an app X with version rating 3.0 uses the $ad1$ and $ad2$ ad libraries, and an app Y with version rating 4.0 uses the $ad1$ and $ad3$ ad libraries, and an app Z with version rating 3.5 uses the $ad1$, $ad2$ and $ad3$ ad libraries. Then, the version ratings for each ad library is $ad1$=$\{$3.0, 4.0, 3.5$\}$, $ad2$=$\{$3.0, 3.5$\}$, and $ad3$=$\{$4.0, 3.5$\}$. We present the median rating for each identified ad library.

Results: **The behaviour of an ad library is related to the rating of an app.** Figure 5.7 presents beanplots for 70 out of the 72 ad libraries studied in this paper (*adhubs* and *mobus* are not included since neither of them existed in the last version of the studied apps). The beanplots present the app rating depending on the use of each specific ad library sorted descending by the rating median of the apps. The long break line and the line across each beanplot represent the rating median of the apps. The beanplots show a slight decrease depending on the type of ad libraries.

We looked among the five lowest rated ad libraries in Figure 5.7 in order to find possible causes of their low rating. *Wooboo* [15] is an ad network based in China. This company also develops its own apps. We found that in some cases this ad library has been flagged as spyware (Fortinet Inc., 2012) (software that collects information about users of an app without their consent, and that may be sent to a third party (Federal Trade Commission, 2005)). A clear app user' complaint about an app that has this ad library is: "*This is a direct copy of another app, it displays ads and now your password belongs to someone*

[15] Wooboo:`http://www.wooboo.com.cn/`

in china" [16]. *Leadbolt* [17] is an ad network based in Australia. This ad library shows an intrusive behaviour pushing ads into the notification bar, *i.e.*, ads are displayed when the app is not running (LeadBolt, 2011). Leadbolt added in 2012 a new type of ad called "app icon" (LeadBolt, 2012). An "app icon" ad installs new icons on the Android device without the authorization of the users. App developers have problems with this ad library because it sends the device ID in plain text (Affpaying, 2012). This security issue might result that an app being rejected from the Amazon store (Corona Labs Inc., 2011). *Airpush* [18] is an ad network based in the USA. As well as *leadbolt*, this ad library uses the push notification and the app icon system to send advertisements. This ad library has become quite controversial among app users for its intrusive behaviour (Ionescu, 2012). We did not identify any incident for the *Wqmobile* and *adlantis* ad libraries.

We observe that these apps with ad libraries that have a high intrusive behaviour result in users complaints, and in some occasions ruin the perceived quality of the app. We find comments such as: *Good game, bad ads. I was loving the game until I noticed it put a new shortcut called "Apps" on my launcher. Sorry, but if your idea of advertising is putting sh*t in launcher pages or notifications then I'm not interested. Keep ads INSIDE the app.* [19]. *Good game, bad ads. This game is awesome but the ads...embarrising. the ads totally ruin it and there isn't a way to take them off :(* [20]. We also find that some app users rate an app with five stars, however, their comments are notably sarcastic such as: *The Ads Are GREAT! (; So Is The Game, Love Everything About It. .....Mostly The Ads. XD* [21].

[16] http://goo.gl/HQtTL
[17] Leadbolt:http://www.leadbolt.com/
[18] Airpush:http://www.airpush.com/
[19] http://goo.gl/CAUhk
[20] http://goo.gl/c1Nks
[21] http://goo.gl/nngtV

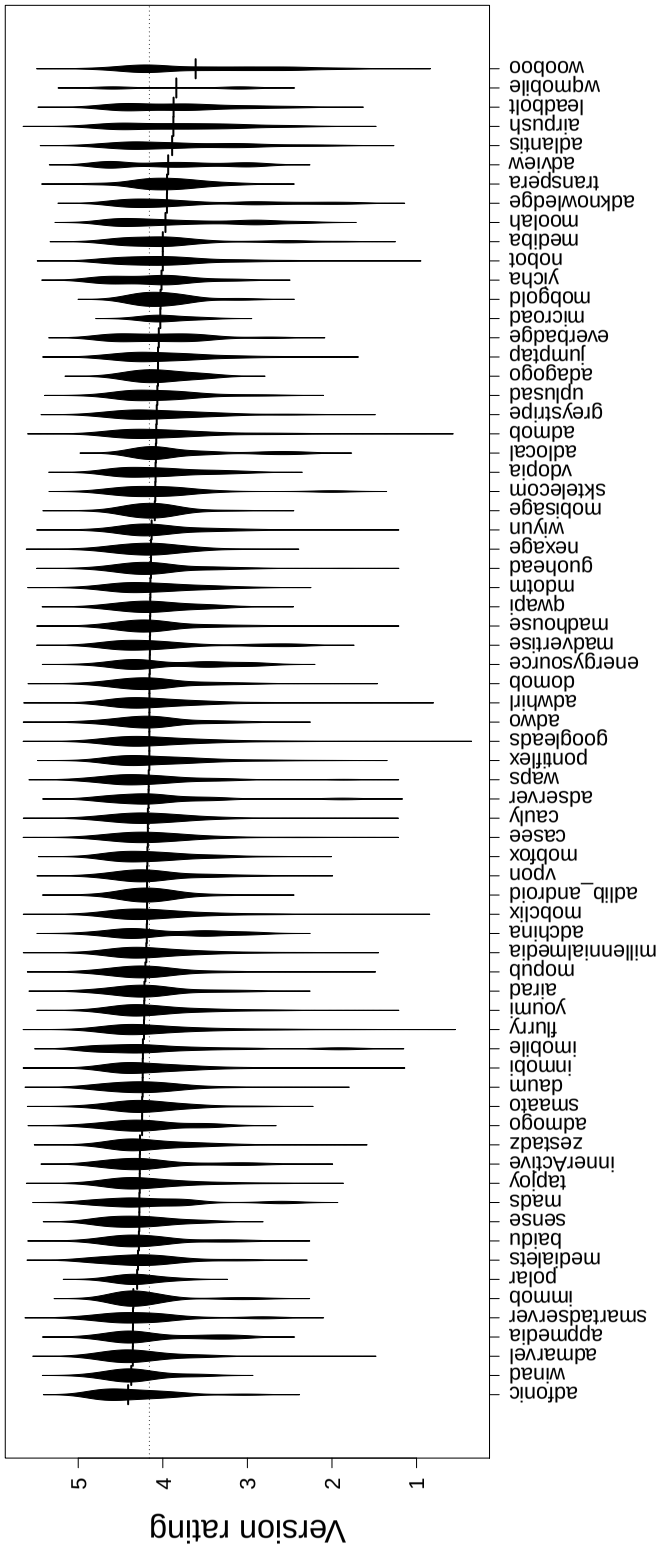


Figure 5.7: Beanplots showing the distribution of the studied ad libraries (x-axis) according to the version rating of the apps that contain the ad library (y-axis). The beanplots are sorted descending by the app version rating media. The horizontal line represents the median of the app version ratings.

App developers also have to consider the type of ad library to use in their apps for security issues (related to spyware as mentioned earlier). Ad libraries with security issues and/or intrusive behaviour may result in apps having low ratings, disapproval from the app users (Chien, 2012), and even rejection from some of the app stores. These situations force the app developers to perform different tasks of ad maintenance (*e.g.*, to change the ad libraries). Recently, Google Play updated its policies that force the app developers to be more conscious about ad libraries with highly intrusive behaviour (Lunden, 2012). App developers need to perform tasks of ad maintenance in order to align their apps with the new policies in Google Play (Google, 2012*c*; Peebles, 2012).

From our observations, we recommend that app developers should monitor the ads' impact on their apps, and perform the required changes if app users complain about the ads in their apps.

App developers are integrating multiple ad libraries as a strategy to achieve a high fill rate. The number of ads is not related to the user perceived quality of an app. However, the intrusive behaviour of some particular ad libraries jeopardizes the quality of an app.

5.4 Threats to Validity

5.4.1 External Validity.

We chose to analyse Android apps because there are readily available APIs to mine the Google Play store. Also, we decided to study only the free to download apps because we required access to the source code or bytecode, which is not possible for paid apps without actually paying. However, given that the ad business model is more prevalent in free apps, we keep this threat under control. Finally, we only have data spanning 2011. Due to these

limitations, our results can potentially suffer from generalizability issues.

5.4.2 Internal Validity.

We used publicly available APIs to mine the Google Play market. Our crawling tools collected the data from the store automatically. Although we have taken every possible step to ensure correct working of our tools, we could not thoroughly evaluate the accuracy of the crawling API. Additionally, we could not collect every app in the market every day, since we are restricted by the number of queries that can be run against the app store. This means that potentially we have missed particular versions of an app. However, since these crawling APIs are used extensively by others too, and our frequency of data collection was relative high, we believe that our results do not suffer significantly from such threats.

5.4.3 Construct Validity.

We identified the distinct ad libraries based on a regular expression that describes the *ad* key word. This might result in the omission of some ad libraries that do not follow this convention. However, we were able to identify a large set of ad libraries based on this process that we consider enough to perform our analysis. In fact, compared to prior studies of ad libraries, we identified a larger number of ad libraries.

We use the user assigned ratings of apps as measure of user perceived quality. Although an app might be of high quality from the developer's perspective, if the app does not have attractive features, generates too many ads or behaves poorly compared to a competitor, it might have a poor rating. Furthermore, users might rate an app negatively on purpose or might neglect to rate an app because they are extremely happy or unhappy. Finally, only a small number of people might rate a particular app, yielding a relatively unreliable rating.

Since we filtered out apps with too low number of versions or raters, the effect of a few anomalous raters should be dealt with.

5.5 Conclusions

Given that the percentage of downloads across the different app stores is vastly dominated by free apps, app developers have to rely on ads in order to generate revenue from their apps. There exists basically three ad models in the ad business model: ad network, ad mediator, and ad exchange. In any ad model, app developers embed ad libraries in their apps. Then, app developers can give their apps for free. In exchange, app users are exposed to advertisements that are shown in the free apps. Finally, the advertising companies pay app developers for the ads in their apps.

However, app developers can not neglect the ad libraries after the release of their apps. App developer have to maintain the ad libraries in their apps throughout the lifetime of their apps. Otherwise, they run the risk of not generating any revenue. Because of the importance of the ad business model on the economical success of app developers, we performed an empirical study on half a million of free Android app versions, from 200K different apps to analyze the ad maintenance in mobile apps.

The major conclusions from this chapter are:

- **Ad libraries play a prominent role in free apps.** Up to 51% of the free to download apps have at least one ad library. We identified 72 different ad libraries.

- **The percentage of ad-supported apps varies depending on the app category.** Some categories are more likely to have a higher percentage of ad libraries than others. For example the free to download apps in the Photography and Games categories have the highest percentage of ad-supported apps (up to 78%). In the other

hand, only 14% of the apps in the Business category are ad-supported.

- **Ad developers are integrating multiple ad libraries.** Up to 35% of the ad-supported apps have more than one ad library. The motivation behind integrating more ad libraries is to have multiple source of ads.

- **App developers are frequently updating the ad libraries.** Almost one in every two app versions has a change in their ad libraries.

- **The number of changes related to the ad libraries depends on the ad serving model.** Using multiple ad network libraries leads to the highest number of ad-related changes.

- **The number of ad libraries is not related to the perceived quality of an app.** Having multiple ad libraries only increases the source of ads, does not seem to lead to lower ratings. If an app does not have an ad to display, an own ad (house ad) can be displayed. Hence, to display an ad is independent of the number of ad libraries.

- **Poor, or a lack of ad maintenance may result in app developers do not generate any revenue, and a negative rating in their apps.** App developers have to deal with constant updates of ad libraries by the advertising companies, dead ad libraries, and the intrusive behavior of certain ad libraries that can negatively affect the quality of their apps, and more importantly the total ad revenue of their apps.

Chapter 6

Conclusions and Future Work

In a market widely dominated by free to download apps, app developers can not depend on the price of their apps as factor of differentiation. App developers are competing with hundred of thousands of other apps. Hence, app developers must obtain a high rating that differentiates their apps from the rest of apps. They must also produce apps with as little effort as possible through code reuse. Furthermore, due to the free nature of the apps, developers must make good use of ads within their apps to generate revenue.

However, in spite of the accelerated growth of the mobile market, few research has been done to address these new challenges that software developers have to face. Hence, we empirically analyzed thousands of apps in order to gain a better understanding about the complexity of the software engineering challenges in mobile apps. Through the studies presented in this thesis we aimed to validate our research statement:

> New software engineering challenges have emerged in the current context of mobile apps. App developers, in order to compete in a massive market of apps, must take into consideration traditional aspects such as code reuse and novel aspects such as the approval of their users, and managing advertisement libraries embedded in their

apps. In depth empirical studies are needed in order to shape and direct future software engineering research for mobile apps.

Chapter 3 examines how the app market can have such an accelerated growth. In the following two chapters (4 and 5) we present two further studies on the rating system in the app market and the maintenance effort currently involved with ad libraries in apps respectively. In particular, from each study presented in this thesis, we find that:

Understanding Reuse in the Google Play App Store

Chapter 3 presents a study of software reuse among thousands of Android apps in five different categories along two dimensions: (a) reuse by inheritance, and (b) class reuse. Since app stores only distribute the byte code of the mobile apps, and not the source code, we used the concept of Software Bertillonage to track code across mobile apps. We found that almost 23% of the classes inherit from a base class in the Android API, and 27% of the classes inherit from a domain specific base class. Furthermore, on average 61% of all classes in each category of mobile apps exist in two or more apps, and 217 mobile apps are reused completely by another mobile app in the same category.

From this study, we conclude that software reuse is prevalent (compared to regular open source software) in mobile apps of the Google Play app store. App developers are reusing software through inheritance, libraries and frameworks. The practice of code reuse has highly contributed to the accelerated growth of the number of apps available.

A Large-scale Empirical Study on User Ratings of Mobile Apps

Chapter 4 shows an empirical analysis of hundred of thousands of free to download apps in order to gain a better understanding of the user rating system for apps. The most interesting findings in this chapter are: (1) The overall rating (*i.e.*, the rating advertised in

Google Play) is biased towards apps with very few raters (67% of the apps had less than 10 raters). (2) Once the biased apps are filtered from the case study, most apps have a high global-rating (median global-rating is 4), which is (3) consistent across categories in the app stores, and (4) resilient to fluctuations once an app has gathered a substantial number of raters. Looking at the rating for a particular version (version-rating), (5) apps have a high chance to increase their version-rating in their following version, *i.e.*, app developers often recover from bad ratings in previous app-versions, and (6) we can build models that can predict the probability of a version-rating increase before the release of a new version of an app. Such a prediction can be used by the owners of app-stores, and can help app developers decide whether they should release a new version of the app or not.

From this study, we conclude that app developers are dealing with various challenges in the rating system for their apps. Unlike traditional apps, mobile app developers need to understand the app rating behaviour in order to compete with the high number of apps available in the app stores.

A Large-scale Empirical Study on Ad Library Maintenance of Mobile Apps

Given the large number of free-to-download apps available in the app stores, and the large proportion of free-to-download apps downloaded compared to paid apps, app developers are depending on business models which allow them to distribute their apps for free. One of these most popular models is the ad business model. Hence, app developers are heavily depending on advertisement libraries to generate revenue from their apps. However, the use of ad libraries in apps demands that app developers perform tasks of ad library maintenance. The ad library maintenance involves changes to the embedded ad libraries in an app, instead of the features of the app. Examples of such changes are correcting problems with the ad libraries, modifying, adding, removing or updating an ad library in an app. The final goal of

a proper ad maintenance is to ensure that the ad libraries do not impact negatively on the app user experience, while app developers generate the maximum revenue from the ads in their apps.

Chapter 5 presents an analysis of ad library maintenance across hundred of thousands of Android apps in the Google Play app store. The more relevant findings are: (1) ad libraries play a prominent role in the free apps. At least 51% of the free-to-download apps have at least one ad library. (2) The percentage of ad-supported apps varies across the different app categories, with the Photography and Games categories having up to 78% of ad-supported apps. On the other hand, in the Business category only 14% of the free-to-download apps are ad supported. (3) Ad-supported apps can contain a large number of ad libraries. At least 35% of the ad-supported apps have two or more ad libraries. (4) App developers are frequently updating the ad libraries in their apps. Up to 48% of the versions of an ad-supported app have a change because of the ad libraries embedded in them. (5) The number of changes related to the ad libraries in an app depends on the type of ad library. (6) The number of ad libraries is not related to the rating of an app. However, (7) the intrusive behaviour of some particular ad libraries impacts negatively the rating of the apps that use them.

From this study, we conclude that app developers are facing a new challenge which we call ad maintenance.

6.1 Limitations and Future Work

The limitations for each of the three studies included in this thesis are included before the conclusions in their respective chapters (threats to validity). However, there is one limitation across the three different studies in this thesis that we detail in this section. Finally, we discuss possible future research directions.

- We chose to study only the free-to-download Android apps because we required access to the source code or bytecode, which is not possible for paid apps without actually paying. Hence, our results can suffer from generalizability. However, Android OS is the the most popular mobile user base (Kellogg, 2011), free-to-download apps are downloaded more frequently, up to 90% (Pettey and van der Meulen, 2012), and the number of free-to-download Android apps available in the Google Play is about three times the number of paid apps (AppBrain, 2013).

The previous limitation points out our object of study: Android apps. This same limitation invites us to consider performing similar studies in other mobile platforms such as BlackBerry, iOS, Windows Phone. For app developers, it is extremely important to identify different factors that affect the success of their apps. In our studies we did not consider elements on the graphical user interface. Do the graphical elements influence the rating of an app? If so, which graphical elements? Also, does other elements such as design, sound, themes, help to increase the rating and number of downloads?

During our analysis we found that app developers are using different type of libraries. Some of these libraries have not been designed specifically for mobile apps, given that apps differ from other type of software, we think that it is important to analyse how these libraries affect the apps, are they adding unnecessary work to app developers, or increasing the number of tasks of software maintenance? How other type of libraries specifically designed for mobile apps (*e.g.*, in-app-purchase, and analytic libraries) are affecting the apps, and the developing practice of app developers?

What other *traditional* software engineer factors (*e.g.*, software reuse) play an import role in the development practices of app developers? App stores are a rich source of information for app developers in order to help them to improve the quality of their apps. For example,

app users are constantly commenting about the different aspects of the apps, what can app developers learn from these comments? What other factors from the app stores are useful for app developers?

Bibliography

ABI (2012), 'In-app purchases to outpace pay-per-download revenues in 2012', http://www.abiresearch.com/press/in-app-purchases-to-outpace-pay-per-download-reven. Last visited on July 1st, 2013.

Adfonic (2012), 'A full and up to date mobile advertising glossary for terms used across the industry', http://adfonic.com/support/glossary/. Last visited on July 1st, 2013.

AdMarvel (2006), 'Admarvel', http://www.admarvel.com. Last visited on July 1st, 2013.

Admob (2013), 'Admob - about', http://www.admob.com/home/about. Last visited on July 1st, 2013.

Adobe Systems (2012), 'Click here: The state of online advertising', http://www.adobe.com/aboutadobe/pressroom/pdfs/Adobe_State_of_Online_Advertising_Study.pdf. Last visited on July 1st, 2013.

Adomavicius, G. and Tuzhilin, A. (2005), "Toward the next generation of recommender

systems: a survey of the state-of-the-art and possible extensions", *Knowledge and Data Engineering, IEEE Transactions on* , Vol. 17, pp. 734 – 749.

Adwhirl (2011), 'Issue 187 - adwhirl - Inmobi adapter failure can cause multiple rollovers - AdWhirl Open Source SDK and Server - Google Project Hosting', http://code.google.com/p/adwhirl/issues/detail?id=187&colspec=ID%20Type%20Component%20Stars%20Status%20Priority%20Opened%20Summary.

Adwhirl (2012), 'Issues adwhirl', http://code.google.com/p/adwhirl/issues/list. Last visited on July 1st, 2013.

Affpaying (2012), 'Leadbolt - content locking - cpa network reviews - affpaying', http://www.affpaying.com/leadbolt. Last visited on July 1st, 2013.

Airpush (2010), 'Airpush - mobile ad network', http://www.airpush.com/. Last visited on July 1st, 2013.

Alexandrou, M. (2013), 'Cpm vs. cpc vs. cpa', http://infolific.com/technology/internet/sem/cpm-cpc-cpa/. Last visited on July 1st, 2013.

Amazon (2011), 'Amazon Appstore', http://www.amazon.com/appstore. Last visited on July 1st, 2013.

Ambler, S. W. (1998), 'The various types of object-oriented reuse'.

Android (2011), 'Android - google+ - and a closer look at the 10 billion +android app downloads', https://plus.google.com/+android/posts/SctK4iLoHCo.

Android (2013*a*), 'Dashboards — android developers', http://developer.android.com/about/dashboards/index.html. Last visited on July 1st, 2013.

Android (2013*b*), 'Visibility for your apps', http://developer.android.com/distribute/googleplay/about/visibility.html. Last visited on July 1st, 2013.

AP (2012), 'Android market share q3 2012: Google's still beating apple, but will the iphone 5 change that?', http://www.huffingtonpost.com/2012/09/18/android-market-share-q3-2012_n_1893292.html. Last visited on July 1st, 2013.

AppBrain (2013), 'Comparison of free and paid android apps', http://www.appbrain.com/stats/free-and-paid-android-applications. Last visited on July 1st, 2013.

Apple (2008*a*), 'Apple's App Store', https://itunes.apple.com/. Last visited on July 1st, 2013.

Apple (2008*b*), 'iphone app store downloads top 10 million in first weekend', http://www.apple.com/pr/library/2008/07/14iPhone-App-Store-Downloads-Top-10-Million-in-First-Weekend.html. Last visited on July 1st, 2013.

Basic4Android (2011), 'Admob unusable error', http://www.basic4ppc.com/forum/basic4android-updates-questions/8743-admob-unusable-error.html. Last visited on July 1st, 2013.

Basili, V. R., Briand, L. C. and Melo, W. L. (1996), "How reuse influences productivity in object-oriented systems", *Commun. ACM* , Vol. 39, ACM, New York, NY, USA, pp. 104–116.

Basili, V. R. and Rombach, H. D. (1991), "Support for comprehensive reuse", *Softw. Eng. J.* , Vol. 6, Michael Faraday House, Herts, UK, UK, pp. 303–316.

Bateman, S. (2012), 'Online advertising models: Cpc, cpm or cpa?', `http://www.promisemedia.com/online-advertising-strategies/best-revenue-deals-cpm-cpc-or-cpa`. Last visited on July 1st, 2013.

Bhalgat, A. and Gollapudi, S. (2012), Ad allocation for browse sessions, *in* 'Proceedings of the 8th international conference on Internet and Network Economics', WINE'12, Springer-Verlag, Berlin, Heidelberg, pp. 475–481.

Blackberry (2009), 'Blackberry App World', `http://appworld.blackberry.com/`. Last visited on July 1st, 2013.

Blum, S. (2012), 'App permissions explained what do they really mean?', `http://www.androidpit.com/app-permissions-explained`. Last visited on July 1st, 2013.

Bryan (2011), 'How to optimize your mobile ads (part 1 of 3)', `http://www.mopub.com/2011/08/25/how-to-optimize-your-mobile-ads-part-1-of-3/`. Last visited on July 1st, 2013.

Burstly (2012), 'How can i increase my fill rate?', `http://support.burstly.com/customer/portal/articles/888632-how-can-i-increase-my-fill-rate-`. Last visited on July 1st, 2013.

Business Wire (2009), 'AdMob Announces Agreement to Acquire AdWhirl', http://www.businesswire.com/portal/site/home/permalink/?ndmViewId=news_view&newsId=20090827006007. Last visited on July 1st, 2013.

Canalys (2013), 'Top ios and android apps largely absent on windows phone and blackberry 10', http://www.canalys.com/newsroom/top-ios-and-android-apps-largely-absent-windows-phone-and-blackberry-10. Last visited on July 1st, 2013.

Candeias, H. (2011), 'Smaato releases q2 2011 mobile metrics report', http://www.smaato.com/metricsq22011-2/. Last visited on July 1st, 2013.

Cardino, G., Baruchelli, F. and Valerio, A. (1997), "The evaluation of framework reusability", *SIGAPP Appl. Comput. Rev.* , Vol. 5, ACM, New York, NY, USA, pp. 21–27.

Chen, D. Y.-C. (2009), Essays on Mobile Advertising and Commerce, PhD thesis, Science, Technology and Management, Harvard University.

Chen, M. and Singh, J. P. (2001), Computing and using reputations for internet ratings, *in* 'Proceedings of the 3rd ACM conference on Electronic Commerce', EC '01, ACM, New York, NY, USA, pp. 154–162.

Chevalier, J. A. and Mayzlin, D. (2003), The effect of word of mouth on sales: Online book reviews, Working Paper 10148, National Bureau of Economic Research.

Chien, E. (2012), 'Madware makers mixing up their code', http://www.mobilesecurity.com/articles/285-madware-makers-mixing-up-their-code. Last visited on July 1st, 2013.

Cleff, E. B. (2010), "Effective approaches to regulate mobile advertising: Moving towards a coordinated legal, self-regulatory and technical response", *Computer Law & Security Review* , Vol. 26, pp. 158–169.

Columbus, L. (2013), 'Roundup of mobile apps and app store forecasts, 2013', `http://www.forbes.com/sites/louiscolumbus/2013/06/09/roundup-of-mobile-apps-app-store-forecasts-2013/`. Last visited on July 1st, 2013.

Connolly, T. G. and Sluckin, W. (1953), *An Introduction to Statistics for the Social Sciences*, Cleaver-Hume Press.

Conte, M. T., Trick, A. R., Gyllenhaal, J. C. and Hwu, W.-m. W. (1998), A study of code reuse and sharing characteristics of java applications, *in* 'Proceedings of the Workload Characterization: Methodology and Case Studies', WWC '98, IEEE Computer Society, Washington, DC, USA, pp. 27–.

Corona Labs Inc. (2011), 'Amazon appstore issue', `http://developer.coronalabs.com/forum/2011/11/03/amazon-appstore-issue`. Last visited on July 1st, 2013.

Cui, G., kwong Lui, H. and Guo, X. (2010), Online reviews as a driver of new product sales, *in* 'Management of e-Commerce and e-Government (ICMeCG), 2010 Fourth International Conference on', pp. 20 –25.

Dave, V., Guha, S. and Zhang, Y. (2012), Measuring and fingerprinting click-spam in ad networks, *in* 'Proceedings of the ACM SIGCOMM 2012 conference on Applications,

technologies, architectures, and protocols for computer communication', SIGCOMM '12, ACM, New York, NY, USA, pp. 175–186.

Davidson, D. and Livshits, B. (2012), 'Morepriv: Mobile os support for application personalization and privacy'. Technical Report MSR-TR-2012-50, available at `http://research.microsoft.com/pubs/163596/MSR-TR.pdf`.

Davies, J., German, D. M., Godfrey, M. W. and Hindle, A. (2011), Software bertillonage: finding the provenance of an entity, *in* 'Proceedings of the 8th Working Conference on Mining Software Repositories', MSR '11, ACM, New York, NY, USA, pp. 183–192.

Daye, D. (2010), '10 principles of advertising - bill bernbach', `http://www.brandingstrategyinsider.com/2010/10/10-principles-of-advertising-bill-bernbach.html`. Last visited on July 1st, 2013.

de Souza, R. (2011), 'Part 2: Ad networks vs remnant ad exchanges: The publisher perspective', `http://www.zedo.com/part-2-ad-exchanges-vs-remnant-ad-networks-the-publisher-perspective/`. Last visited on July 1st, 2013.

Denier, S. and Guéhéneuc, Y.-G. (2008), Mendel: A model, metrics, and rules to understand class hierarchies, *in* 'Proceedings of the 2008 The 16th IEEE International Conference on Program Comprehension', ICPC '08, IEEE Computer Society, Washington, DC, USA, pp. 143–152.

Development, S. (2013), 'Average app store review times', `http://reviewtimes.shinydevelopment.com/`. Last visited on July 1st, 2013.

Dienst, S. and Berger, T. (2012), 'Static analysis of app dependencies in android bytecode'. Tech. Note, available at `http://www.informatik.uni-leipzig.de/~berger/tr/2012-dienst.pdf`.

DrMop (2011), 'The big list of android app stores where to sell and what to avoid', `http://www.drmop.com/index.php/2011/10/08/the-big-list-of-android-app-stores-where-to-sell-and-what-to-void/`. Last visited on July 1st, 2013.

eMarketer (2012), 'How millennial men react to mobile ads - emarketer', `http://www.emarketer.com/Article.aspx?R=1008834`. Last visited on July 1st, 2013.

Enck, W., Octeau, D., McDaniel, P. and Chaudhuri, S. (2011), A study of android application security, *in* 'Proceedings of the 20th USENIX conference on Security', SEC'11, USENIX Association, Berkeley, CA, USA, pp. 21–21.

Enck, W., Ongtang, M. and McDaniel, P. (2009), "Understanding android security", *Security and Privacy Magazine* , Vol. 7, pp. 50–57.

Fayad, M. and Schmidt, D. C. (1997), "Object-oriented application frameworks", *Commun. ACM* , Vol. 40, ACM, New York, NY, USA, pp. 32–38.

Federal Trade Commission (2005), 'Monitoring software on your pc: Spyware, adware, and other software', `http://www.ftc.gov/os/2005/03/050307spywarerpt.pdf`. Last visited on July 1st, 2013.

Ferrell, J. (2012), 'Why does my app have to be reviewed before it is released?', `http://www.digitalrelativity.com/why-does-my-app-have-to-be-reviewed-before-it-is-released/`. Last visited on July 1st, 2013.

Flurry (2012), 'Flurry - release notes for android', http://support.flurry.com/index.php?title=Analytics/Code/ReleaseNotes/Android. Last visited on July 1st, 2013.

Fortinet Inc. (2012), 'Riskware/spyboo!android', http://www.fortiguard.com/av/VID3734287. Last visited on July 1st, 2013.

Frakes, W. B. and Kang, K. (2005), "Software reuse research: Status and future", *IEEE Trans. Softw. Eng.* , Vol. 31, IEEE Press, Piscataway, NJ, USA, pp. 529–536.

Gartner (2011), 'Gartner says worldwide mobile application store revenue forecast to surpass $15 billion in 2011', http://www.gartner.com/it/page.jsp?id=1529214. Last visited on July 1st, 2013.

Gartner Inc. (2011), 'Worldwide mobile application store revenue forecast to surpass $15 billion in 2011', http://www.gartner.com/newsroom/id/1529214. Last visited on July 1st, 2013.

Gasimov, A., Tan, C.-H., Phang, C. W. and Sutanto, J. (2010), Visiting mobile application development: What, how and where, *in* 'Proceedings of the International Conference on Mobile Business and Global Mobility Roundtable (ICMB-GMR)', pp. 74–81.

Gold, H. (2007), 'Pros and cons of ad exchanges', http://www.clickz.com/clickz/column/1693572/pros-cons-ad-exchanges. Last visited on July 1st, 2013.

Google (2008), 'Google Play App Store', https://play.google.com/store/apps. Last visited on July 1st, 2013.

Google (2011), 'Error Reports of SDK 4.0.3', https://groups.google.com/

forum/?fromgroups=#!topic/google-admob-ads-sdk/UaTw_HjhSxs. Last visited on July 1st, 2013.

Google (2012*a*), 'Banners i - google admob ads sdk - google developers', https://developers.google.com/mobile-ads-sdk/docs/admob/fundamentals. Last visited on July 1st, 2013.

Google (2012*b*), 'Google analytics sdk for android - google analytics - google code', http://code.google.com/apis/analytics/docs/mobile/android.html. Last visited on July 1st, 2013.

Google (2012*c*), 'New sdk 5.0 from airpush to comply with google play new policies?', https://groups.google.com/forum/?fromgroups=#!topic/android-developers/-RXK0jP-4zs. Last visited on July 1st, 2013.

Google (2013*a*), 'Adwhirl', https://www.adwhirl.com/. Last visited on July 1st, 2013.

Google (2013*b*), 'Create a house ad', http://support.google.com/admob/answer/1619751. Last visited on July 1st, 2013.

Google (2013*c*), 'Mobile ads - google ads', http://www.google.ca/ads/mobile/. Last visited on July 1st, 2013.

Google (2013*d*), 'Ratings and comments', http://support.google.com/googleplay/android-developer/bin/answer.py?answer=138230. Last visited on July 1st, 2013.

Grace, M. C., Zhou, W., Jiang, X. and Sadeghi, A.-R. (2012), Unsafe exposure analysis of mobile in-app advertisements, *in* 'Proceedings of the fifth ACM conference on Security

and Privacy in Wireless and Mobile Networks', WISEC '12, ACM, New York, NY, USA, pp. 101–112.

Harbich, S. and Auer, S. (2005), Rater bias: The influence of hedonic quality on usability questionnaires, *in* M. Costabile and F. Patern, eds, 'Human-Computer Interaction - INTERACT 2005', Vol. 3585 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 1129–1133.

Harman, M., Jia, Y. and Zhang, Y. (2012), App store mining and analysis: Msr for app stores, *in* 'Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on', MSR'12.

Hartley, M. (2012), 'Blackberry jam 2012: Live coverage', `http://business.financialpost.com/2012/09/25/blackberry-jam-2012-live-coverage/`. Last visited on July 1st, 2013.

Heinemann, L., Deissenboeck, F., Gleirscher, M., Hummel, B. and Irlbeck, M. (2011), On the extent and nature of software reuse in open source java projects, *in* 'Proceedings of the 12th international conference on Top productivity through software reuse', ICSR'11, Springer-Verlag, Berlin, Heidelberg, pp. 207–222.

Hemel, A., Kalleberg, K. T., Vermaas, R. and Dolstra, E. (2011), Finding software license violations through binary code clone detection, *in* 'Proceedings of the 8th Working Conference on Mining Software Repositories', MSR '11, ACM, New York, NY, USA, pp. 63–72.

Herraiz, I., Shihab, E., Nguyen, T. H. D. and Hassan, A. E. (2011), Impact of installation

counts on perceived quality: A case study on debian, *in* 'Proc. of the 2011 18th Working Conf. on Reverse Engineering (WCRE)', pp. 219–228.

Holub, A. (2003), 'Why extends is evil', `http://www.javaworld.com/javaworld/jw-08-2003/jw-0801-toolbox.html`. Last visited on July 1st, 2013.

Ionescu, C. (2012), 'Airpush begins obfuscating ad modules', `http://www.symantec.com/connect/blogs/airpush-begins-obfuscating-ad-modules`. Last visited on July 1st, 2013.

ISO/IEC (2006), Software Engineering - Software Life Cycle Processes - Maintenance, Norm ISO/IEC 14764:2006.

Johnson, J. (2012), 'Fake google play apps pulled immediately after approval', `http://www.inquisitr.com/403255/fake-google-play-apps-pulled-immediately-after-approval/`. Last visited on July 1st, 2013.

Johnson, R. E. and Foote, B. (1988), "Designing reusable classes", *Object-Oriented Programming* , Vol. 1, pp. 22–35.

Kapser, C. J. and Godfrey, M. W. (2008), ""cloning considered harmful" considered harmful: patterns of cloning in software", *Empirical Softw. Engg.* , Vol. 13, Kluwer Academic Publishers, Hingham, MA, USA, pp. 645–692.

Kaufman, B. (2013), 'Mobile advertising and pricing trends for 2013', `http://www.doublepositive.com/blog/mobile-advertising-and-pricing-trends-2013/`. Last visited on July 1st, 2013.

Kellogg, D. (2011), '40 percent of u.s. mobile users own smartphones; 40 percent are android', `http://blog.nielsen.com/nielsenwire/online_mobile/`

40-percent-of-u-s-mobile-users-own-smartphones-40-percent-are-android/. Last visited on July 1st, 2013.

Kim, K. J., Kim, S. Y., Park, E., Sundar, S. and del Pobil, A. (2012), The more the better? effects of ad exposure frequency on online consumers with varying product knowledge, *in* 'Information Science and Digital Content Technology (ICIDT), 2012 8th International Conference on', Vol. 1, pp. 92 –96.

Kincaid, J. (2009), 'Google acquires admob for $750 million', http://techcrunch.com/2009/11/09/google-acquires-admob/. Last visited on July 1st, 2013.

Koetsier, J. (2013), 'Why developers choose the amazon app store: fewer apps, ease of porting, and pending global expansion', http://venturebeat.com/2013/04/25/why-developers-choose-the-amazon-app-store-fewer-apps-ease-of-porting-and-pending-global-expansion/. Last visited on July 1st, 2013.

Kumar Maji, A., Hao, K., Sultana, S. and Bagchi, S. (2010), Characterizing failures in mobile oses: A case study with android and symbian, *in* 'Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)', pp. 249–258.

Lawson, S. (2009), 'Android Market Needs More Filters, T-Mobile Says', http://www.pcworld.com/article/161410/article.html. Last visited on July 1st, 2013.

LeadBolt (2011), 'Earn more with push notifications', http://www.leadbolt.com/

blog/2011/10/earn-more-with-push-notifications/. Last visited on July 1st, 2013.

LeadBolt (2012), 'Boost revenues with leadbolt's new app icon', http://goo.gl/SsG94. Last visited on July 1st, 2013.

Lee, S. Y. (2004), To vary or not? the effects of ad variation on the web, PhD thesis. AAI3147645.

Leontiadis, I., Efstratiou, C., Picone, M. and Mascolo, C. (2012), Don't kill my ads!: balancing privacy in an ad-supported mobile application market, *in* 'Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications', HotMobile '12, ACM, New York, NY, USA, pp. 2:1–2:6.

Li, S. (2012), Juxtapp and dstruct: Detection of similarity among android applications, Master's thesis, EECS Department, University of California, Berkeley. Last visited on July 1st, 2013.

Li, Z., Zhang, K., Xie, Y., Yu, F. and Wang, X. (2012), Knowing your enemy: understanding and detecting malicious web advertising, *in* 'Proceedings of the 2012 ACM conference on Computer and communications security', CCS '12, ACM, New York, NY, USA, pp. 674–686.

Lohr, S. (2010), 'Google's do-it-yourself app creation software', http://www.nytimes.com/2010/07/12/technology/12google.html. Last visited on July 1st, 2013.

Lunden, I. (2012), 'Google tightens up app policy, gets stricter on naming/icon, payments, privacy, ads and spam rules [developer letter]', http:

//techcrunch.com/2012/08/01/google-tightens-up-app-policy-gets-stricter-on-namingicon-payments-privacy-ads-and-spam-rules-developer-letter/. Last visited on July 1st, 2013.

Mackenzie, T. (2012), 'Understanding cost-per-click and cost-per-action for mobile ads', http://www.techrepublic.com/blog/app-builder/understanding-cost-per-click-and-cost-per-action-for-mobile-ads/1268. Last visited on July 1st, 2013.

Malheiros, M., Jennett, C., Patel, S., Brostoff, S. and Sasse, M. A. (2012), Too close for comfort: a study of the effectiveness and acceptability of rich-media personalized advertising, *in* 'Proceedings of the SIGCHI Conference on Human Factors in Computing Systems', CHI '12, ACM, New York, NY, USA, pp. 579–588.

McIlroy, M. D. (1968), *Mass Produced Software Components*, Vol. 1, NATO Science Committee, pp. 138–155.

Megan (2012), 'Inside the android factory', http://blog.startapp.com/inside-the-android-factory/. Last visited on July 1st, 2013.

Michail, A. (1999), Data mining library reuse patterns in user-selected applications, *in* 'Proceedings of the 14th IEEE international conference on Automated software engineering', ASE '99, IEEE Computer Society, Washington, DC, USA, pp. 24–.

Michail, A. (2000), Data mining library reuse patterns using generalized association rules, *in* 'Proceedings of the 22nd international conference on Software engineering', ICSE '00, ACM, New York, NY, USA, pp. 167–176.

Microsoft (2010), 'Windows Phone App Store', http://www.windowsphone.com/en-US/store. Last visited on July 1st, 2013.

Miles, S. (2013), '5 ad exchanges for local publishers', http://streetfightmag.com/2013/01/22/5-ad-exchanges-for-local-publishers/. Last visited on July 1st, 2013.

Minelli, R. and Lanza, M. (2013), Software analytics for mobile applications - insights & lessons learned, *in* 'Proceedings of CSMR 2013 (17th European Conference on Software Maintenance and Reengineering)', IEEE CS Press, pp. 144–153.

Mobclix (2005), 'obclix - Payments', http://www.mobclix.com/faqs.html#faqs-3. Last visited on July 1st, 2013.

MobiDev (2013), 'Mobile software monetization models', http://mobidev.biz/blogs/mobile_software_monetization_models.html. Last visited on July 1st, 2013.

mobiThinking (2012), 'mobithinking guide to mobile advertising networks (2012)', http://mobithinking.com/mobile-ad-network-guide. Last visited on July 1st, 2013.

mobiThinking (2013), 'The mobithinking guide to mobile advertising networks 2013: Premium blind networks', http://mobithinking.com/mobile-ad-network-guide/premiumblind. Last visited on July 1st, 2013.

Moby affiliates (2012), 'World's top mobile advertising platforms', http://www.mobyaffiliates.com/blog/worlds-top-mobile-advertising-platforms/. Last visited on July 1st, 2013.

MobyAffiliates (2013), 'Wooboo mobile advertising', http://www.mobyaffiliates.com/mobile-advertising-networks/wooboo/. Last visited on July 1st, 2013.

Mockus, A. (2007), Large-scale code reuse in open source software, *in* 'Proceedings of the First International Workshop on Emerging Trends in FLOSS Research and Development', FLOSS '07, IEEE Computer Society, Washington, DC, USA, pp. 7–.

Mohamed, F. E., Schmidt, D. C. and Johnson, R. E. (1997), *ObjectOriented Application Frameworks: Problems and Perspectives*, Wiley.

Mui, L., Mohtashemi, M., Ang, C., Szolovits, P. and Halberstadt, A. (2001), Ratings in distributed systems: A bayesian approach, *in* 'Workshop on Information Technologies and Systems'.

Nagappan, N. and Ball, T. (2005), Use of relative code churn measures to predict system defect density, *in* 'Proceedings of the 27th international conference on Software engineering', ICSE '05, ACM, New York, NY, USA, pp. 284–292.

Nokia (2012), 'Nokia ad exchange - faq', http://www.developer.nokia.com/Distribute/NAX/FAQ.xhtml. Last visited on July 1st, 2013.

on Mobile, A. (2010), 'Mobile Advertising: CPC or CPM? CPA!', http://analyticsonmobile.com/2010/05/17/mobile-advertising-cpc-or-cpm-cpa.html. Last visited on July 1st, 2013.

O'Neill, N. (2010), '10 surprising app platform facts [infographic]', http://www.allfacebook.com/app-platform-facts-2010-09. Last visited on July 1st, 2013.

Opera (2013), 'The state of mobile advertising, q2 2012', http://business.opera.com/sma/2012/q2/. Last visited on July 1st, 2013.

Organized Shopping, LLC (2012), 'House ad', http://www.marketingterms.com/dictionary/house_ad/. Last visited on July 1st, 2013.

Ossher, J., Sajnani, H. and Lopes, C. (2011), File cloning in open source java projects: The good, the bad, and the ugly, *in* 'Software Maintenance (ICSM), 2011 27th IEEE International Conference on', pp. 283 –292.

Oswald, E. (2011), 'Pandora sends personal data en masse to advertisers, researcher says', http://betanews.com/2011/04/07/pandora-sends-personal-data-en-masse-to-advertisers-researcher-says. Last visited on July 1st, 2013.

Palumbo, P. A. (2010), 'Accustream research: Online video advertising forecast at $2.7 billion in '10; double-digit growth projected through '12', http://www.accustreamresearch.com/news/20100811.html. Last visited on July 1st, 2013.

Palumbo, P. A. (2012), 'Accustream research: Video, mobile ad networks, trafficking and clearing platforms combine for $13.2 billion in 2012 media spend', http://www.accustreamresearch.com/news/20120904.html. Last visited on July 1st, 2013.

Pathak, A., Hu, Y. C., Zhang, M., Bahl, P. and Wang, Y.-M. (2011), Fine-grained power

modeling for smartphones using system call tracing, *in* 'Proceedings of the sixth conference on Computer systems', EuroSys '11, ACM, New York, NY, USA, pp. 153–168.
URL: *http://doi.acm.org/10.1145/1966445.1966460*

Peebles, C. V. (2012), 'Why airpush's new sdk is huge for advertisers and developers', `http://blog.airpush.com/why-airpushs-new-sdk-is-huge-for-advertisers-and-developers/`. Last visited on July 1st, 2013.

Pettey, C. and van der Meulen, R. (2012), 'Gartner says free apps will account for nearly 90 percent of total mobile app store downloads in 2012', `http://www.gartner.com/it/page.jsp?id=2153215`. Last visited on July 1st, 2013.

Pettey, C. and van der Meulen, R. (2013), 'Gartner says worldwide mobile advertising revenue to reach $11.4 billion in 2013', `http://www.gartner.com/newsroom/id/2306215`. Last visited on July 1st, 2013.

PlacePlay (2012), 'How to Make Money with Apps: Part 1', `http://www.placeplay.com/how-to-make-money-with-apps-1`.

Prodhan, G. (2012), 'Apps become key to mobile advertising: report', `http://www.reuters.com/article/2012/04/20/net-us-mobile-advertising-apps-idUSBRE83J1EA20120420`. Last visited on July 1st, 2013.

Quirolgico, S., Voas, J. and Kuhn, R. (2011), "Vetting mobile apps", *IT Professional* , Vol. 13, IEEE Computer Society, Los Alamitos, CA, USA, pp. 9–11.

Resnick, P. and Zeckhauser, R. (2002), Trust among strangers in internet transactions: Empirical analysis of ebay's reputation system, *in* 'The Economics of the Internet and E-Commerce, M. R. Baye (ed.), Amsterdam: Elsevier Science', Vol. 11, pp. 127 – 157.

Rosen, K. (2012), 'Google play has 700,000 apps, tying apple's app store', http://mashable.com/2012/11/01/google-apps-tie-apple/. Last visited on July 1st, 2013.

Rowinski, D. (2012), 'History of mobile app stores'.

Roy, C. K. and Cordy, J. R. (2008), An empirical study of function clones in open source software, *in* 'Proceedings of the 2008 15th Working Conference on Reverse Engineering', WCRE '08, IEEE Computer Society, Washington, DC, USA, pp. 81–90.

Ruckman, R. (2012), 'Fraudulent mobile ad network mobgold closes permanently but they don't want you to know', http://www.imgrind.com/fraudulent-mobile-ad-network-mobgold-closes-permanently-but-they-dont-want-you-to-know/. Last visited on July 1st, 2013.

Saracino, A. (2012), 'Why you should integrate video into your marketing plan', http://www.clickz.com/clickz/column/2208509/why-you-should-integrate-video-into-your-marketing-plan. Last visited on July 1st, 2013.

Saverio (2010), 'Monetize: Ad network vs. mediator vs. exchange and what it means for you', http://blog.mobclixdev.com/2010/11/10/monetize-ad-network-vs-mediator-vs-exchange-and-what-it-means-for-you/. Last visited on July 1st, 2013.

Seungyeop Han, Jaeyeon Jung, D. W. (2011), 'A study of third-party tracking by mobile apps in the wild'. Technical Report UW-CSE-12-03-01, available at ftp://ftp.cs.washington.edu/tr/2012/03/UW-CSE-12-03-01.PDF.

Shabtai, A., Fledel, Y. and Elovici, Y. (2010), Automated static code analysis for classifying android applications using machine learning, *in* 'International Conference on Computational Intelligence and Security', CIS'10.

Sharma, C. (2010), 'Sizing up the global apps market', `http://www.chetansharma.com/mobileappseconomy.htm`. Last visited on July 1st, 2013.

Shekhar, S., Dietz, M. and Wallach, D. S. (2012), Adsplit: separating smartphone advertising from applications, *in* 'Proceedings of the 21st USENIX conference on Security symposium', Security'12, USENIX Association, Berkeley, CA, USA, pp. 28–28.

Siang, O. K. (2011), 'Get advertisement on your windows phone app', `http://ooiks.com/blog/tag/software-development`. Last visited on July 1st, 2013.

Slivka, E. (2012), 'Average wait time for mac app store app reviews rising significantly', `http://www.macrumors.com/2012/10/08/average-wait-time-for-mac-app-store-app-reviews-rising-significantly/`. Last visited on July 1st, 2013.

Spinellis, D. (2006), *Code Quality: The Open Source Perspective (Effective Software Development Series)*, Addison-Wesley Professional.

Spring, T. (2011), 'Sneaky mobile ads invade android phones', `http://www.techhive.com/article/245305/sneaky_mobile_ads_invade_android_phones.html`. Last visited on July 1st, 2013.

StackOverflow (2011), 'getting started with admob for android - confused about documentation', http://stackoverflow.com/questions/5320186/getting-started-with-admob-for-android-confused-about-documentation.

Stackpole, B. (2011), 'Your next job: Mobile app developer?', http://www.computerworld.com/s/article/9217885/Your_next_job_Mobile_app_developer_?taxonomyId=11&pageNumber=1. Last visited on July 1st, 2013.

Stevens, R., Gibler, C., Crussell, J., Erickson, J. and Chen, H. (2012), "Investigating user privacy in android ad libraries".

Stone-Gross, B., Stevens, R., Zarras, A., Kemmerer, R., Kruegel, C. and Vigna, G. (2011), Understanding fraudulent activities in online ad exchanges, *in* 'Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference', IMC '11, ACM, New York, NY, USA, pp. 279–294.

Strube, O. (2012), 'Cpc. cpm. cpa. what"s the best value for your ad campaign?', http://www.350media.com/2012/07/12/cpc-cpm-cpa-whats-the-best-value-for-your-ad-campaign/. Last visited on July 1st, 2013.

Syer, M. D. (2013), Empirical Studies of Mobile Apps and Their Dependence on Mobile Platforms, MSc thesis, School of Computing, Queen's University.

Syer, M. D., Adams, B., Zou, Y. and Hassan, A. E. (2011), Exploring the development of micro-apps: A case study on the blackberry and android platforms, *in* 'Proceedings of

the 2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation', SCAM '11, IEEE Computer Society, Washington, DC, USA, pp. 55–64.

Taivalsaari, A. (1996), "On the notion of inheritance", *ACM Comput. Surv.* , Vol. 28, ACM, New York, NY, USA, pp. 438–479.

Turow, J., King, J., Hoofnagle, C. J., Bleakley, A. and Hennessy, M. (2009), 'Americans reject tailored advertising and three activities that enable it'.

Ur, B., Leon, P. G., Cranor, L. F., Shay, R. and Wang, Y. (2012), Smart, useful, scary, creepy: perceptions of online behavioral advertising, *in* 'Proceedings of the Eighth Symposium on Usable Privacy and Security', SOUPS '12, ACM, New York, NY, USA, pp. 4:1–4:15.

Vallina-Rodriguez, N., Shah, J., Finamore, A., Grunenberger, Y., Papagiannaki, K., Haddadi, H. and Crowcroft, J. (2012), Breaking for commercials: characterizing mobile advertising, *in* 'Proceedings of the 2012 ACM conference on Internet measurement conference', IMC '12, ACM, New York, NY, USA, pp. 343–356.

Vascellaro, J. E. (2009), 'Google wagers on cellphone ads', `http://online.wsj.com/article/SB10001424052748704402404574525651367011402.html`. Last visited on July 1st, 2013.

Virki, T. (2012), 'Privacy risk from ads in apps rising: security firm', `http://www.reuters.com/article/2012/07/09/net-us-mobile-advertising-idUSBRE86807020120709`. Last visited on July 1st, 2013.

VisionMobile (2012), 'Developer economics 2012', `http://www.visionmobile.com/product/developer-economics-2012/`. Last visited on July 1st, 2013.

Wang, B.-C., Zhu, W.-Y. and Chen, L.-J. (2008), Improving the amazon review system by exploiting the credibility and time-decay of public reviews, *in* 'Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT '08. IEEE/WIC/ACM International Conference on', Vol. 3, pp. 123 –126.

Wasserman, A. I. (2010), Software engineering issues for mobile application development, *in* 'FSE/SDP workshop on Future of software engineering research', FoSER '10.

Wei, X., Gomez, L., Neamtiu, I. and Faloutsos, M. (2012), Profiledroid: multi-layer profiling of android applications, *in* 'Proceedings of the 18th annual international conference on Mobile computing and networking', Mobicom '12, ACM, New York, NY, USA, pp. 137–148.

Weintraub, S. (2010), 'How angry birds will soon make $1 million a month', `http://tech.fortune.cnn.com/2010/12/03/how-angry-birds-makes-a-million-a-month/`. Last visited on July 1st, 2013.

Wojcicki, S. (2010), 'We've officially acquired admob!', `http://googleblog.blogspot.ca/2010/05/weve-officially-acquired-admob.html`. Last visited on July 1st, 2013.

Zacharia, G., Moukas, A. and Maes, P. (1999), Collaborative reputation mechanisms in electronic marketplaces, *in* 'Proceedings of the Thirty-second Annual Hawaii International Conference on System Sciences-Volume 8 - Volume 8', HICSS '99, IEEE Computer Society, Washington, DC, USA, pp. 8026–.

Zhou, W., Zhou, Y., Jiang, X. and Ning, P. (2012), Detecting repackaged smartphone applications in third-party android marketplaces, *in* 'Proceedings of the second ACM

conference on Data and Application Security and Privacy', CODASPY '12, ACM, New York, NY, USA, pp. 317–326.

Appendix A

Factors Used to Study the Dynamics of User Ratings of Android Apps

Table A.1: Factors used to study the dynamics of user ratings of Android apps. For each app version, we calculated the absolute value of these factors as well as the value relative to the previous app version. All the factors are numeric type, with the exception of the *Marketing* factors that are boolean.

Dim.	Factor	Explanation	Rationale
Outcome	Version-rating	Mean rating of a specific app version, calculated as $\frac{\sum_{i=0}^{N} star_i}{N}$ with $star_i$ the number of stars (1, 2, 3, 4 or 5) assigned by a user to this version and N the rating count (see below)	Measure for quality of a specific app version as perceived by the users (Android, 2013*b*).
	Average-rating	Average (mean) rating of an app across the version-ratings of each app version: $\frac{\sum_{i=0}^{M} version_i * count_i}{\sum_{i=0}^{M} count_i}$ with $version_i$ and $count_i$ the rating count of an app version.	Measure for quality of specific app over time (across all versions) as perceived by the users (Android, 2013*b*).

Continued on next page

Table A.1 – *Continued from previous page*

Dim.	Factor	Explanation	Rationale
Base	Previous version-rating	Version-rating of the previous app version.	The quality or reputation of the previous version might influence the current version's quality. To improve a bad version from a bad version might be difficult.
	Rating count	Number of users who have rated an app version.	An app might be rated by one user with the highest rating, or by thousands of users with differing ratings.
Size	Total classes	Total number of classes per app version.	The code size of the app, as a measure of the amount of features in an app, might play an important role in user-perceived quality.

Continued on next page

Table A.1 – *Continued from previous page*

Dim.	**Factor**	**Explanation**	**Rationale**
	Churn	Number of classes added or deleted compared to the previous app version.	Churn is known to be a good predictor of software defects (Nagappan and Ball, 2005), which might affect the quality perceived by users.
	Binary size	Binary size in bytes.	Since mobile devices have memory constraints, and network access incurs data plan costs (Wei et al., 2012), users might be influenced by these constraints.

Continued on next page

Table A.1 – *Continued from previous page*

Dim.	Factor	Explanation	Rationale
OO Design	Weighted methods per class (WMC).	Median of the number of methods in each class of an app version.	Chidamber and Kemerer's Object Oriented (complexity) metrics along with CA and NPM are a measure of good design for an app (Spinellis, 2006).
	Depth of inheritance tree (DIT).	Median of the length of the inheritance chain across each class of an app version.	
	Number of children (NOC).	Median of the number of immediate subclasses of each class in the app version.	
	Coupling between object classes (CBO).	Median of the number of classes that a class is coupled to in an app version.	

Continued on next page

Table A.1 – *Continued from previous page*

Dim.	Factor	Explanation	Rationale
	Response for a class (RFC).	Median of the number of different methods that can be executed when an object in a class receives a message in the app version.	
	Lack of cohesion in methods (LCOM).	Median of the number of pairs of methods in a class that do not share at least one field in the app version.	
	Afferent coupling (CA).	Median of the number of other classes that use a class in an app version.	
	Number of public methods (NPM).	Median of the number of public methods in each class of an app version.	

Continued on next page

Table A.1 – *Continued from previous page*

Dim.	Factor	Explanation	Rationale
Reuse	Unique classes	Number of classes that are unique to this app.	This is code written possibly for the first time by the app developer and hence may not be as reliable
	Library reuse	Number of classes that are common with other apps, i.e., that are reused.	This is library code, which likely has been tested rigorously and should be more reliable.
Franchise	Previous versions	Number of previous app versions.	The maturity of an app might influence user-perceived quality.
	Mean previous average-rating	Mean of the average-rating of all previous app versions.	The reputation of an app might influence user-perceived quality.

Continued on next page

Table A.1 – *Continued from previous page*

Dim.	Factor	Explanation	Rationale
	Mean previous rating count	Mean of number of raters across previous app versions.	The ability of an app to engage users to rate the app might influence user-perceived quality.
Company	Other apps	Number of other apps by same company.	An established company might receive a better rating because of its experience and name.
	Mean churn	Mean churn across all versions of all other apps by the same company.	Degree of activity of a company might give an indication of the company's desire to improve quality.

Continued on next page

Table A.1 – *Continued from previous page*

Dim.	Factor	Explanation	Rationale
	Mean rating	Mean average-rating across all versions of all other apps by the same company.	The reputation of the other apps of a company might influence how users perceive this app version
	Mean rating count	Mean rating count across all versions of all other apps by the same company.	The ability of other apps of the company to engage user feedback might influence the ability of this app version to do so.
	Mean total classes	Mean number of total classes across all versions of all other apps by the same company.	The scope and feature set of other apps of the company might relate to how users perceive this app version

Continued on next page

Table A.1 – *Continued from previous page*

Dim.	Factor	Explanation	Rationale
Marketing	Contact website	Indicates whether the app version's app store page presents a link to the company's website.	Users might trust apps with a support website more than those without.
	Promotional video	Indicates whether the app version's app store page presents a link to a video.	A video promoting or explaining the app might yield more favourable ratings than other apps.