STUDYING THE INTEGRATION PRACTICES AND THE EVOLUTION OF AD LIBRARIES IN THE GOOGLE PLAY STORE

by

MD AHASANUZZAMAN

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

July 2019

Abstract

IN-APP advertisements have become a major revenue for app developers in the mobile app economy. Ad libraries play an integral part in this ecosystem as app developers integrate these libraries into their apps to display ads. However, little is known about how app developers integrate these libraries with their apps and how these libraries have evolved over time.

In this thesis, we study the ad library integration practices and the evolution of such libraries. To understand the integration practices of ad libraries, we manually study apps and derive a set of rules to automatically identify four strategies for integrating multiple ad libraries. We observe that integrating multiple ad libraries commonly occurs in apps with a large number of downloads and ones in categories with a high percentage of apps that display ads. We also observe that app developers prefer to manage their own integrations instead of using off the shelf features of ad libraries for integrating multiple ad libraries.

To study the evolution of ad libraries, we conduct a longitudinal study of the 8 most popular ad libraries. In particular, we look at their evolution in terms of size, the main drivers for releasing a new ad library version, and their architecture. We observe that ad libraries are continuously evolving with a median release interval of 34 days. Some ad libraries have grown exponentially in size (e.g., Facebook Audience Network ad library), while other libraries have worked to reduce their size. To study the main drivers for releasing an ad library version, we manually study the release notes of the eight studied ad libraries. We observe that ad library developers continuously update their ad libraries to support a wider range of Android versions (i.e., to ensure that more devices can use the libraries without errors). Finally, we derive a reference architecture for ad libraries and study how the studied ad libraries diverged from this architecture during our study period.

Our findings can assist ad library developers to understand the challenges for developing ad libraries and the desired features of these libraries.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to Allah (God) for his blessings and help to accomplish this work. I am incredibly thankful to my supervisor Prof. Ahmed E. Hassan for his guidance and support throughout this journey. He motivated me not only to set big goals but to carry out an actionable plan to achieve them in time. I consider myself very lucky to work under his supervision.

I would also like to thank my examination committee members Prof. Yuan Tian and Prof. Thomas Dean for their valuable feedback and useful comments.

I am greatly indebted to my colleague Dr. Safwat Mohamed Ibrahim Hassan. He is very talented. I learned a great many things from him which assist me to improve my knowledge and skill. I am very thankful to him for his guidance and help. I would also like to thank Prof. Cor-Paul Bezemer for his support.

I am fortunate to work with the brightest minds in SAIL (Software Analysis & Intelligence Lab) lab. My appreciation extends to my fellow labmates: Jiayuan Zhou, Md

Hasan Ibrahim, Dr. Heng Li, Filipe Roseiro Cô go, Dr. Gustavo Ansaldi Oliva, Shahnaz Shariff, Gopi Krishnan Rajbahadur, Dr. Shaowei Wang, Daniel Lee, Haoxiang Zhang, and Abdullah Ahmad Zarir for their support. I would like to offer my special thanks to Dr. Fahim Imam, Dr. Mohamed Sami Rakha, and Dr. Prashant Agrawal for their great help.

I owe my deepest gratitude to my mother (Ayesha Akhtara Begum) for her continuous blessings, support, and help. I would like to dedicate my work to my father (Late. Md. Abdul Jalil Miah). I want to thank my brother Dr. Muhammad Asaduzzaman who always keeps me motivated and shows me a good example. A special thanks to my maternal uncles, my maternal aunts, my sister-in-law (Nushrat Jahan), and my nephew (Nubaid Ilham). I believe that this work could not be done without their endless love and their continuous praying for me.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

THE mobile app market is continuously evolving at a tremendous rate with billions of mobile app downloads every year (Statista (2019)). The majority of the apps in app stores are free-to-download (95% of the apps in the Google Play Store are free-to-download (AppBrain (2019))). To earn revenue from these free-to-download apps, app developers primarily use an *in-app advertising* model. In this model, app developers display advertisements (*ads*) to app users and earn revenue based on the number of displayed ads and user interactions with these ads. The in-app advertising model is a growing market with a forecasted revenue of $201 billion by 2021 (AppAnnie (2017)). Figure 1.1 presents an overview of the in-app advertising

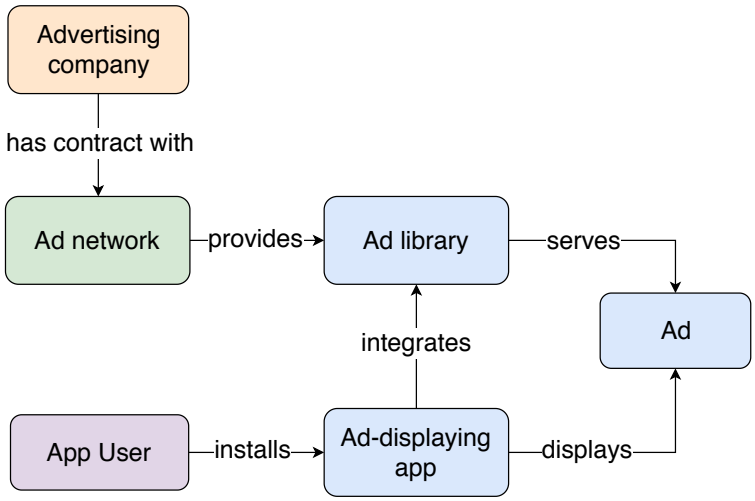

Figure 1.1: An overview of the in-app advertising model.

model. The in-app advertising model consists of four main components: (1) *advertising companies* that pay for the display of ads for promoting their products, (2) *ad-displaying app* that displays ads and earns revenue from the displayed ads, (3) *mobile ad networks* which act as a bridge between advertising companies and ad-displaying apps, and (4) *users* who use the ad-displaying apps and interact with the displayed ads.

To display ads, app developers need to register with an ad network (e.g., Facebook Audience Network) and integrate into their app a library that is offered by the ad network. This library is known as an *ad library*. The main objective of an *ad library* is to manage the communication with an ad network, and to track user's interaction with the displayed ads.

In the competitive app market, many ad networks continue to emerge and each ad network offer its own ad libraries. To maximize app revenue, app developers often integrate ad libraries from several ad networks (Davidson et al. (2014); Grace et al. (2012)). For example, Ruiz et al. (2014) observed that the number of ad libraries that are integrated into an app could be as large as 28. Although ad libraries are an integral part for

app revenue, prior studies show that ad libraries can increase the development effort and can have a negative impact on the integrating app (e.g., ad libraries can increase the energy consumption of the app (Gui et al. (2015)), or they can negatively affect the user-perceived quality (Hassan et al. (2018))). Since ad libraries play an integral role in mobile app ecosystem, developers who wish to build their own ad libraries or the developers who have their ad libraries need to understand the challenges of integrating of such ad libraries within apps and learn how these ad libraries have evolved.

1.1 Thesis Statement

> Studying the integration practices of ad libraries and the evolution of these libraries can help ad library developers understand the current challenges of developing such libraries and possible improvements for such libraries.

1.2 Thesis Overview

In this section, we provide an outline of our thesis.

1.2.1 Chapter 2: Related work

In this chapter, we provide an overview of prior research that is related to ad libraries.

1.2.2 Chapter 3: Studying Ad Library Integration Practices of Top Free-to-Download Apps

In this chapter, our aim is to understand how app developers integrate ad libraries into their apps as prior research shows that app developers could integrate more than one ad library. First, we study the characteristics of integrating multiple ad libraries. Then, we manually analyze a statistically representative random sample of the studied apps and derive a set of rules to automatically identify the strategies for integrating multiple ad libraries.

We observe that the integration of multiple ad libraries commonly occurs in apps with a large number of downloads and ones in categories with a high percentage of apps that integrate ad libraries for displaying ads. Our study of the strategies for integrating multiple ad libraries shows that app developers prefer to customize and manage their own integrations instead of just using off the shelf features of ad libraries for integrating multiple ad libraries. Our findings are valuable for ad library developers who wish to gain firsthand knowledge about the challenges of integrating ad libraries.

1.2.3 Chapter 4: A Longitudinal Study of Popular Ad Libraries in the Google Play Store

In this chapter, we study the evolution of the 8 most popular ad libraries. First, we look at their evolution in terms of size and understand how ad library developers manage the size of their ad libraries. Then, we manually analyze the release notes of the studied ad libraries and identify the main drivers for releasing a new version. Knowing such drivers can help ad library developers understand the challenges of evolving ad

libraries. Finally, we derive a reference architecture from the studied ad libraries and study how these libraries deviated from this architecture in the study period. Our findings and the derived reference architecture for ad libraries are valuable for ad library developers who wish to learn how other developers built and evolved their successful ad libraries.

1.3 Thesis Contributions

In this thesis, we study how app developers integrate ad libraries with their apps and how these libraries have evolved. We demonstrate that our in-depth analysis of ad libraries can help ad library developers to identify the challenges and possible improvements for their libraries. In particular, our main contributions are as follows:

1. This is the first work to study how app developers integrate ad libraries with their apps. We derive a set of rules to automatically identify the strategies that app developers use when integrating multiple ad libraries. Our in-depth analysis of each identified strategy can help ad library developers identify the challenges and possible improvements to facilitate the integration of ad libraries.

2. Our longitudinal analysis of ad libraries provides valuable insights into the evolution of ad libraries.

3. We study the main drivers for ad library developers to release a new version. Such drivers help ad library developers to understand the challenges of developing ad libraries.

4. We are the first to propose a reference architecture for ad libraries. This reference architecture helps developers who wish to build their own ad libraries understand the important components of ad libraries and how they interact with each others.

CHAPTER 2

Related Work

There have been several studies on ad libraries in the mobile app analysis research area. Prior work mainly focused on the updates of ad libraries, the cost of ad libraries and the security issues of ad libraries. All of the prior work focuses on the impact of ad libraries. However, our thesis is the first to investigate the integration practices and the evolution of the ad libraries themselves. We discuss the related work below.

2.1 The Updates of Ad Libraries

Ruiz et al. (2016) performed an empirical study on the frequency of ad library updates in mobile apps. The authors analyzed 120,981 free-to-download apps from the Google Play Store. To determine ad library updates, Ruiz et al. generated class signatures and

compared the signatures between two consecutive updates of classes using the software bertillonage approach. Their result showed that app developers actively update their ad libraries, as Ruiz et al. found that ad libraries were updated in 48% of the apps.

Derr et al. (2017) studied what drives app developers to update third-party libraries (including their ad libraries) in Android apps. The authors first surveyed 203 app developers to better understand third-party library usage in apps. The authors also performed a large-scale updatability analysis on 1.2M apps from the Google Play Store. Derr et al. concluded from the survey that bug-fixes and security fixes would motivate developers to update a third-party library. The result of the updatability analysis showed that 60% of the app developers regularly update their third-party libraries.

While prior research focuses on analyzing the updatability of ad libraries (e.g., how frequent app developers update their ad libraries), the objective of our thesis is to understand how ad libraries evolve over time from the perspective of the developers of such libraries. In particular, we analyze the frequency of ad library releases and their size. We also investigate what drives ad library developers to release a new version of ad libraries. Finally, we study how the architecture of ad libraries evolves over time. Our study is important for ad library developers and researchers as it provides the first in-depth analysis on how successful ad libraries have evolved during the study period.

2.2 The Cost of Ad Libraries

Ruiz et al. (2014) analyzed the impact of ad libraries on the rating of mobile Android apps. Ruiz et al. mined 236K mobile apps and 519K updates of these mobile apps to study the relationship between the number of ad libraries that are integrated into an app and the app's user rating. The result showed that the number of integrated ad

libraries is not related to the app's rating. However, using certain ad libraries could result in poor app ratings. Ruiz et al. suggested that developers need to be careful and selective about the ad libraries that they choose to integrate.

Gui et al. (2015) investigated the hidden costs of mobile advertising by analyzing 21 real-world apps from the Google Play Store. The result showed that hidden cost of ads manifests itself in performance, memory usage, network usage, maintenance of ad-related code and the app rating.

Gao et al. (2018) investigated 104 popular Android apps and identified 12 ad schemes for which the authors studied the cost of using ads. In particular, Gao et al. measured the performance cost of these identified ad schemes in terms of memory, network traffic, and battery consumption. Based on the study, the authors suggested that app developers should use the Google AdMob ad library as it consumes less CPU overhead than the Mopub ad library and developers should use a full-banner scheme to display ads due to its low-performance cost and its association with a higher rating of the integrating apps.

2.3 The Security of Ad Libraries

Prior work by Calciati and Gorla (2017); Calciati et al. (2018); Felt et al. (2011); Au et al. (2012); Backes et al. (2016); Wang et al. (2019) shows that privacy and security are emerging issues in mobile apps. Since ad libraries are widely integrated in mobile apps, researchers study the impact of using ad libraries on app security. For example, Book et al. (2013) analyzed 1,14,000 apps to understand the evolution of the requested permissions of ad libraries. They observed that the use of permissions has increased over time, and they conclude that most of the permissions that are requested by ad libraries

are risky in terms of user privacy and security.

Kim et al. (2016) analyzed the protective measurements of the Google AdMob, MoPub, AirPush, and AdMarvel ad libraries against malicious advertising. They found that these ad libraries require permissions, such as the *WRITE_EXTERNAL_STORAGE* and *READ_EXTERNAL_STORAGE* permissions, that could make apps users vulnerable to attacks.

Li et al. (2016) investigated 1.5 million apps using 1,113 third-party libraries and 240 ad libraries to investigate which libraries are commonly used in Android apps. The study showed that the most used library is Google's ad library (AdMob). Li et al. also observed that a significant portion of apps that used ad libraries are apps that are flagged by virus scanners.

Dong et al. (2018) conducted an exploratory study on ad fraud (e.g., cheating advertisers with fake ad clicks) in mobile apps and proposed an automated approach for detecting these ad frauds in mobile apps. Their automated approach achieves 92% recall and 93% precision on the manually validated data set of 100 apps. To further study ad frauds in mobile apps, they analyzed 12,000 ad-supported apps that use 20 unique ad libraries. They observed that no ad libraries were exempt from fraudulent behaviours and that the AppBrain ad library is the most targeted ad library for ad frauds.

CHAPTER 3

Studying Ad Library Integration Practices of Top Free-to-Download Apps

IN-APP advertisements have become a major revenue source for app developers in the mobile app ecosystem. As a result, ad libraries play an integral part in this ecosystem. App developers integrate these libraries into their apps to display ads and gain revenue based on user interactions with the displayed ads. However, prior work has never explored – how app developers integrate ad libraries into their apps.

In this chapter, we study ad library integration practices by analyzing 35,462 updates of 1,840 top free-to-download apps of the Google Play Store. We observe that ad libraries (e.g., Google AdMob) are not always used for serving ads – 22.3% of the apps that integrate Google AdMob ad library do not display ads. They instead depend on Google AdMob ad library for analytical purposes. Among the apps that display ads,

we observe that 59.5% of them integrate multiple ad libraries. We observe that such integration of multiple ad libraries occurs commonly in apps with a large number of downloads and ones in app categories with a high proportion of ad-displaying apps.

We manually analyze a sample of apps and derive a set of rules to automatically identify the common strategies for integrating multiple ad libraries. We identify four such strategies: *(1) external-mediation strategy* (app developers use an external-ad-mediator package that is provided by an ad library and do not write their own code to integrate other ad libraries), *(2) self-mediation strategy* (app developers write their own centralized code (self-mediator) to integrate ad libraries), *(3) scattered strategy* (app developers scatter their code across the different app screens), and *(4) mixed strategy* (app developers use both the external-mediation strategy and the scattered strategy). We observe that the mixed and the self-mediation strategies are the dominant ones for integrating multiple ad libraries – showing that app developers prefer to manage their own integrations instead of using off the shelf external-ad-mediators.

Our findings are valuable for ad library developers who wish to learn first hand about the challenges of integrating ad libraries in mobile apps.

3.1 Introduction

In-app advertising is a growing market, many ad networks are emerging in this market with their own ad library. In this competitive market, app developers select an ad network that maximizes their revenue (e.g., offering a high fill rate[1]) (Quora (2016)). To earn more app revenue, app developers integrate multiple ad libraries with their apps to increase the fill rate (Ruiz et al. (2014)).

[1]Fill rate is the ratio of the number of displayed ads over the number of requested ads.

Despite the integral role of ad libraries in the mobile app ecosystem, there have been no prior studies that analyzed how these libraries are integrated int mobile apps and how app developers handle multiple ad libraries. In this paper, we perform an in-depth study of the common practices of integrating such ad libraries in the top free-to-download apps in the Google Play Store. Our study can help ad library developers understand the common challenges of integrating multiple libraries into mobile apps. Hence, ad library developers can improve the design and the offered features of their ad libraries to the ease the ad library integration process.

To study ad library integration practices, we analyzed 35,462 updates of 1,840 top free-to-download apps from the Google Play Store. In particular, we studied such practices along the following research questions (RQs):

RQ1: *What are the characteristics of integrating multiple ad libraries?* App developers integrate multiple ad libraries to display ads on their apps. The integration of multiple ad libraries occurs commonly in the apps with a large number of downloads and ones in app categories where a high proportion of apps integrate ad libraries.

RQ2: *How do app developers integrate multiple ad libraries with their apps?* We manually analyze a statistically representative random sample of ad-displaying apps (62) that integrate multiple ad libraries and derive a set of rules to automatically identify (four) strategies that app developers use for integrating multiple ad libraries: *(1) external-mediation strategy* (app developers use an external-ad-mediator package that is provided by an ad library and do not write their own code to integrate other ad libraries), *(2) self-mediation strategy* (app developers

write their own centralized code (self-mediator) to integrate ad libraries), *(3) scattered strategy* (app developers scatter their code across the different app screens), and *(4) mixed strategy* (app developers use both the external-mediation strategy and the scattered strategy).

We document the definition, example app, the benefits, and drawbacks of each identified strategy for integrating multiple ad libraries. Developers of ad libraries can leverage our strategies to ensure that their ad libraries can support the varying needs of ad-displaying apps.

The rest of the chapter is organized as follows. Section 3.2 describes our data collection process. Section 3.4 presents the results of our study. Section 3.5 discusses how app developers maintain their integrated ad libraries over time. Section 3.6 describes the implications of our work. Section 3.7 describes threats to the validity of our observations, and Section 3.8 concludes the paper.

3.2 Data collection

In this section, we describe our process for collecting ad library data. Figure 3.1 represents an overview of our data collection process. As shown in Figure 3.1, first, we collected the updates of top free-to-download apps in the Google Play Store. Then, we identified ad libraries that are integrated by the apps in these updates. Finally, we identify the updates that display ads. We briefly highlight each step below.

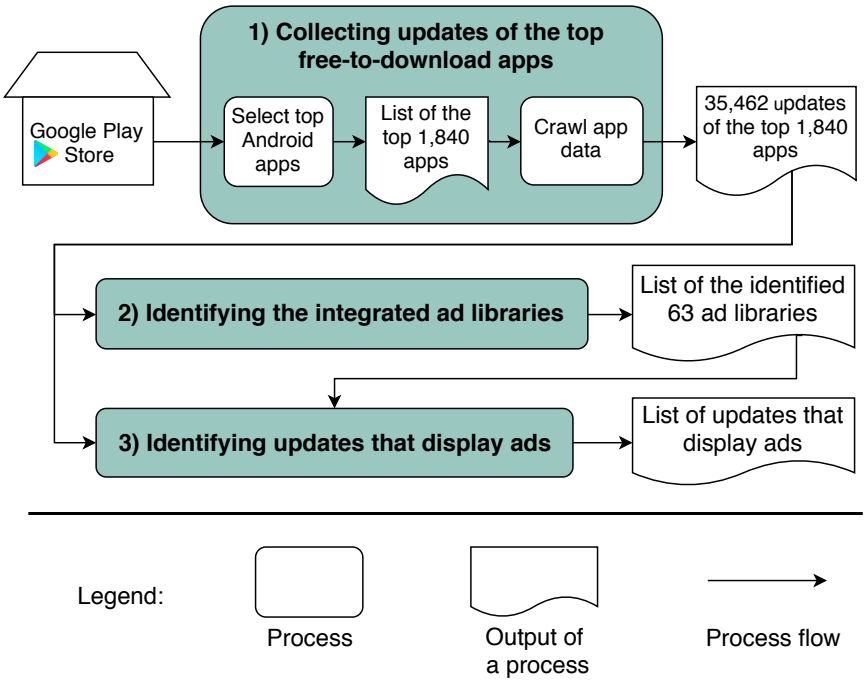

Figure 3.1: An overview of our data collection process.

3.2.1 Collecting updates of the top free-to-download apps

Step 1: Select top Android apps. In our study, we focus on the top free-to-download apps as these apps have a large user-base. Hence, these apps are likely to follow the in-app advertising model to earn revenue. Moreover, these apps are likely to maintain their ad integration code well in order to ensure that they do not lose any ad revenue. To obtain the list of popular apps, we used the App Annie's report (AppAnnie (2018)) that lists the popular apps across the 28 categories (e.g., Games) in the Google Play Store in 2016. Then, we selected the top 100 apps in each app category so that our study does not have any bias due to variances across the different app categories. During the app selection process, we found that 746 apps were removed from the Google Play Store at the start of our study period and 214 apps were repeated across the app categories. In total, we selected 1,840 apps and downloaded all their deployed updates for our study.

Step 2: Crawl app data. We ran a custom crawler (based on the Akdeniz's (Akdeniz (2013)) Google Play crawler) for 18 months from April 20^{th} 2016 to September 20^{th} 2017 to collect the deployed updates of our studied apps. At the end of this step, we collected 35,462 updates for the 1,840 top free-to-download apps.

3.2.2 Identifying the integrated ad libraries

App developers integrate many third-party libraries and identifying an ad library package from these third-party libraries is a non-trivial task. To identify an ad library package, we followed a similar approach to the exhaustive one that is presented by Ruiz et al. (2016). We detail our process below.

First, we converted the APKs of the collected updates to JARs using the dex2jar tool (dex2jar (2016)). Then, we used the BCEL tool Apache BCEL (2018) to extract the fully qualified class names (i.e., the class name and the package name) of all classes in the generated JARs. Since prior studies showed that an ad library's packages or class names contain the term "ad" or "Ad" (Li et al. (2016)), we filtered the fully qualified class names using the regular expression "[aA][dD]". However, this exhaustive regular expression matches many class names that are not related to ad libraries (e.g., `com.fbox.load.ImageLoad`). Hence, to identify ad libraries, we followed the approach of (Ruiz et al. (2016)) and manually verified online the package name of each of the matched classes. We manually verified 303 packages on the web. In total, we identified 63 ad libraries. The Appendix describes the list of 303 packages that we manually analyzed online and the list of identified 63 ad libraries.

3.2.3 Identifying updates that display ads

In the previous step, we identified the list of the integrated ad libraries. However, integrating an ad library in an update does not necessarily imply that the update displays ads (e.g., ad libraries can be used for analytical purposes as we discovered in our study). To identify the updates that display ads, first, we identified the app screens (as ads need to be displayed through the app screens). Then, we identified the screens that display ads. The details of our approach are as follows.

Step 1: Identify app screens. To create a single app screen, app developers write the required functionality of the screen in a java class which is known as an *Activity*. Then, app developers define the app screens (i.e, activities) in the *AndroidManifest.xml* file using the "$< activity >$" tag (Google (2019a)). Hence, to identify the app screens, we parsed the *AndroidManifest.xml* file and listed all the defined activities (using the "$< activity >$" tag) and their corresponding classes.

Step 2: Identify the screens that display ads. First, we identified the integrated libraries in every screen using the BCEL tool (Apache BCEL (2018)). Then, we identified screens that display ads if the screen code invokes the display method in the integrated ad library (e.g., calling the showAd() method). Finally, we flagged an update as an ad-displaying update if the update contains at least one screen that displays an ad.

At the end of this step, we identified all updates that display ads.

3.3 Data characteristics

In this section, we describe the characteristics of our dataset in terms of (1) ad-displaying functionality, (2) app category, and (3) integrated ad libraries.

Ad libraries are not only used for serving ads but also for analytical purposes. In our dataset, the studied apps can be classified into two main categories: (1) *ad-displaying* apps (i.e., apps that integrate ad libraries to display ads) and (2) *non-ad-displaying* apps (i.e., apps that do not display ads). Table 3.1 describes our dataset. As shown in the Table 3.1, we observe that 22.3% of the non-ad-displaying apps (154 apps) integrate the Google AdMob ad library. In particular, we observe that these apps use third-party analytics libraries (e.g., AppsFlyer analytics) that depend on the Google AdMob ad library to uniquely identify a user's device.

Analytics libraries need to track in-app behavior (e.g., how long users use an app) for each user. Therefore, analytics libraries need to uniquely identify a user's device. Table 3.2 shows the top ten used third-party libraries that depend on the Google AdMob ad library (for the studied 154 apps) to identify a user's device. For example, we observe that the Google Analytics library depends on the package "com.google.android.gms.ads.identifier" (Google (2019)) of the Google AdMob ad library. This package provides the functionality to generate an Android Advertising ID (AAID) which is a recommended practice to identify a user's device instead of using a user's personal information (e.g., IMEI number or device MAC address) (Google (2019); Terkki et al. (2017)).

Given our abovementioned observation that ad libraries are not used only for serving ads, researchers who study ad library integration practices need to be careful that the analyzed apps are ad-displaying apps (i.e., the integrated ad libraries are used for serving ads).

Table 3.1: Statistics of the studied apps

App category	Category definition	# of apps	% of apps
Ad-displaying	Apps that integrate ad libraries and display ads	1,145	62.3%
Non-ad-displaying	Apps that **do not integrate** ad libraries	538	29.2%
	Apps that integrate ad libraries but **do not** display ads	154	8.4%

Table 3.2: Top ten third party libraries that depend on Google AdMob ad library.

Package name	Library name	# of apps using the package	% of apps using the package
com.google.android.gms.analytics (Google (2019d))	Google Analytics	151	98.1%
com.appsflyer (AppsFlyer (2019))	AppsFlyer Analytics	23	14.9%
com.flurry.sdk (Flurry (2019))	Flurry Analytics	14	9.1%
com.kochava.android.tracker (Kochava (2019))	Kochava Analytics	13	8.4%
com.localytics.android (Localitics (2019))	Android Location Tracker	10	6.5%
com.life360.android.location (Life360 (2019))	Life 350 Location Tracker	4	2.6%
com.mologiq.analytics (NinthDecimal (2019))	MoLogiq Analytic	4	2.6%
com.quantcast.measurement.service (Quantcast (2019))	Quantcast Measure	4	2.6%
com.urbanairship.analytics (Analytics (2019))	Urban Airship Analytics	3	1.9%
com.moat.analytics.mobile.ovi (Moat (2017))	Moat Analytics	2	1.3%

Although *Google AdMob* and *Facebook Audience Network* are the most integrated ad libraries throughout the studied ad-displaying apps, other ad libraries are popular within certain app categories. Table 3.3 presents the top ten integrated ad libraries of the studied ad-displaying apps. As shown in the Table 3.3, Google AdMob is the most widely integrated ad library (96.4% of the ad-displaying apps integrate the Google AdMob ad library). To understand the popularity of an ad library in every app category, we measured the percentage of apps that integrate every ad library in each app category. Table 3.4 shows the top five integrated ad libraries in each app category.

Table 3.3: Statistics for the top ten integrated ad libraries.

Ad library	**# of ad-displaying apps**	**% of ad-displaying apps**
Google AdMob	1,104	96.4%
Facebook Audience Network	506	44.2%
MoPub	312	27.2%
Amazon Mobile Ad	130	11.4%
Flurry	117	10.2%
Millennialmedia	117	10.2%
AdColony	117	10.2%
InMobi	112	9.8%
Applovin	105	9.2%
Unity Ads	95	8.3%

As shown in the Table 3.4, we observe that Google AdMob and Facebook Audience Network are the most integrated ad libraries in each app category. However, other ad libraries are popular within certain app categories. For example, we observe that Unity Ads is the second most integrated ad library in the Game category (52% of the ad-displaying apps in the Game category integrate Unity Ads ad libarary). One possible reason for the popularity of the Unity Ads ad library in the Game category is that the library provides easy integration to the apps that are built on the Unity framework (a popular framework for building games). In addition, the Unity Ads ad library offers features for displaying rewarded video ads (e.g., users earn an extra life or coins if they watch a video ad) which have become popular among video gaming apps as these ads improve user engagement with the app (BusinessOfApps (2017); MobileMarketing (2018)).

We also observe that the MoPub (MP) ad library is the second most popular ad library in four app categories (i.e., the Weather, Medical, Travel and local, and Libraries

Table 3.4: Top five ranked ad libraries in each app category.

App category	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5
Music and audio	GA(96%)	FAN(37%)	MP(22%)	MM(13%)	IM(11%)
Weather	GA(100%)	**MP(53%)**	FAN(45%)	AMA(38%)	MM(30%)
Personalization	GA(96%)	FAN(85%)	MP(37%)	FL(9%)	UA(8%)
Entertainment	GA(65%)	FAN(43%)	MP(28%)	AV(23%)	UA(21%)
Photography	GA(94%)	FAN(59%)	MP(23%)	MV(19%)	AV(10%)
Game	GA(89%)	**UA(52%)**	AC(50%)	VL(42%)	AV(41%)
News and magazines	GA(95%)	FAN(31%)	MP(28%)	FH(15%)	IA(11%)
Tools	GA(98%)	FAN(69%)	MP(45%)	FL(18%)	DAP(18%)
Video players	GA(95%)	FAN(28%)	MP(15%)	IM(8%)	AV(6%)
Auto and vehicles	GA(100%)	FAN(14%)	MP(14%)	–	–
Sports	GA(90%)	FAN(20%)	FH(15%)	MP(13%)	MM(11%)
Social	GA(91%)	FAN(65%)	MP(40%)	FL(31%)	AC(23%)
Comics	GA(88%)	FAN(30%)	AMA(22%)	AC(19%)	IM(13%)
Books and reference	GA(88%)	FAN(24%)	MP(13%)	AMA(13%)	AB(11%)
Health and fitness	GA(100%)	FAN(35%)	MP(20%)	AMA(17%)	MV(10%)
Productivity	GA(95%)	FAN(64%)	MP(26%)	FL(16%)	DAP(9%)
Lifestyle	GA(100%)	FAN(45%)	MP(31%)	AMA(18%)	FL(13%)
Communication	GA(89%)	FAN(54%)	MP(35%)	FL(21%)	IM(18%)
Medical	GA(100%)	**MP(30%)**	FAN(23%)	AM(15%)	AC(11%)
Shopping	GA(90%)	FAN(18%)	TJ(4%)	MP(4%)	VL(4%)
Finance	GA(100%)	FAN(13%)	MP(6%)	FY(6%)	MM(6%)
Maps and navigation	GA(92%)	FAN(7%)	MP(7%)	AS(3%)	–
Travel and local	GA(78%)	**MP(21%)**	MM(14%)	FL(7%)	AOL(7%)
Education	GA(96%)	FAN(33%)	MP(22%)	FL(7%)	–
Libraries and demo	GA(65%)	**MP(11%)**	IM(11%)	FL(11%)	MP(11%)
Business	GA(91%)	FAN(35%)	MP(13%)	AMA(8%)	DAP(4%)

The abbreviations for ad libraries are as follows: AdColony *(AC)*, AdMarvel *(AM)*, AerServ *(AS)*, Amazon Mobile Ad *(AMA)*, AppBrain *(AB)*, Du Ad Platform *(DAP)*, Facebook Audience Network *(FAN)*, Flurry *(FL)*, FreeWheel *(FH)*, Google AdMob *(GA)*, InMobi *(IM)*, MillennialMedia *(MM)*, MobVista *(MV)*, MoPub *(MP)*, TapJoy *(TJ)*, Unity Ads *(UA)*, andVungle *(VL)*.

* The bold text highlights ad libraries (in Rank 2) other than Facebook Audience Network *(FAN)* ad library.

and demo app categories). One possible reason for MoPub's popularity in these categories is that the MoPub ad library offers an external-ad-mediator. The external-ad-mediator is an ad library feature that facilitates app developers to display ads from

different ad networks by integrating a single ad library.

In particular, we observe that 76% of the ad-displaying apps in the Weather category integrate multiple ad libraries with the external-ad-mediator of the MoPub ad library which is the most used external-ad-mediator in this app category. We also observe that the external-ad-mediator of MoPub ad library is the most used external-ad-mediator in other app categories (i.e., Medical, Travel and local, and Libraries and demo app categories).

Summary

While ad libraries are commonly integrated for serving ads, they are often integrated for analytical purposes. While the mobile ad market is heavily dominated by the Google AdMob and Facebook Audience Network ad libraries, other ad libraries still play a leading role in some particular app categories.

3.4 A Study of the Integration Practices of Ad Libraries

We now present our study of the integration practices of ad libraries. For each research question, we discuss the motivation, approach and results.

3.4.1 RQ1: What are the characteristics of integrating multiple ad libraries?

Motivation: Ad networks decide whether to serve ads for a requesting app based on different factors (e.g., the characteristics of the user-base of that app). For example, ad libraries tend to serve ads for apps of countries that contain a large user-base (Terkki et al. (2017)). Hence, apps most often integrate more than one ad library.

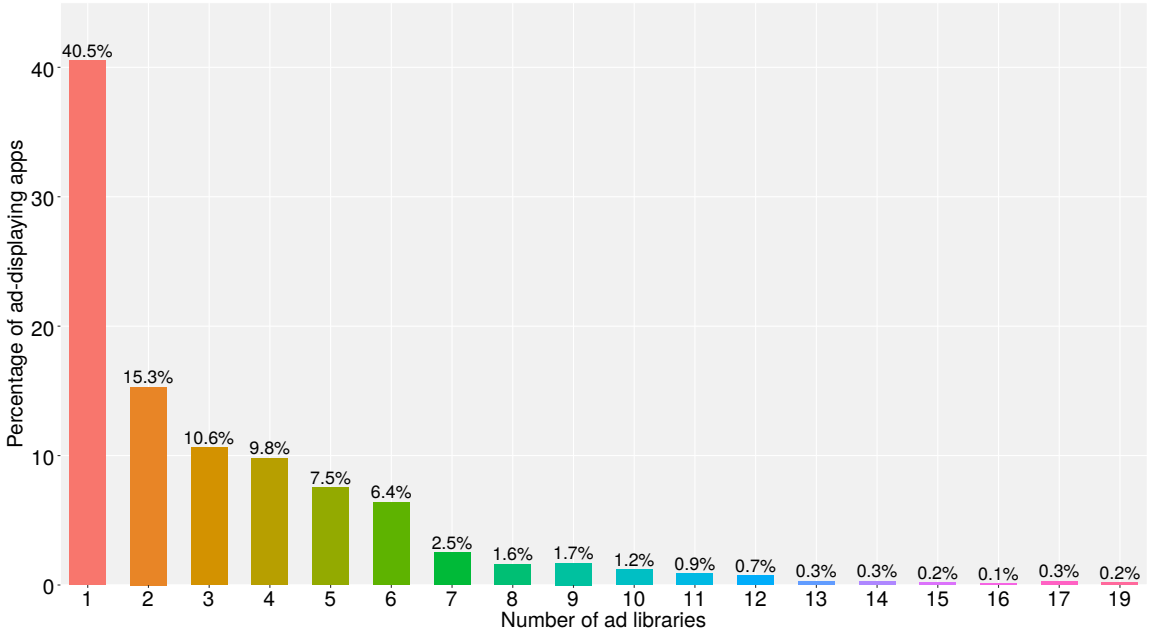

Figure 3.2: The percentage of ad-displaying apps that integrate a specific number of ad libraries.

A good understanding of the apps that integrate multiple ad libraries would help the developers of ad libraries better understand how apps use their ad libraries and how their libraries co-exist with other competing ad libraries.

Approach: For this study, we calculated the percentage of ad-displaying apps that integrate a specific number of ad libraries. Then, we calculated the *multiple-ads* ratio (as the ratio of apps that integrate *multiple ad libraries* to apps that integrate *a single ad library*) across every download range. A multiple-ads ratio that is higher than one indicates that the number of apps that integrate multiple ad libraries is higher than the number of apps that integrate a single ad library (for a certain download range).

Findings: **59.5% of the ad-displaying apps integrate multiple ad libraries.** Figure 3.2 shows the percentage of ad-displaying apps along with the number of integrated ad libraries. We observe that the number of integrated ad libraries could reach up to 19

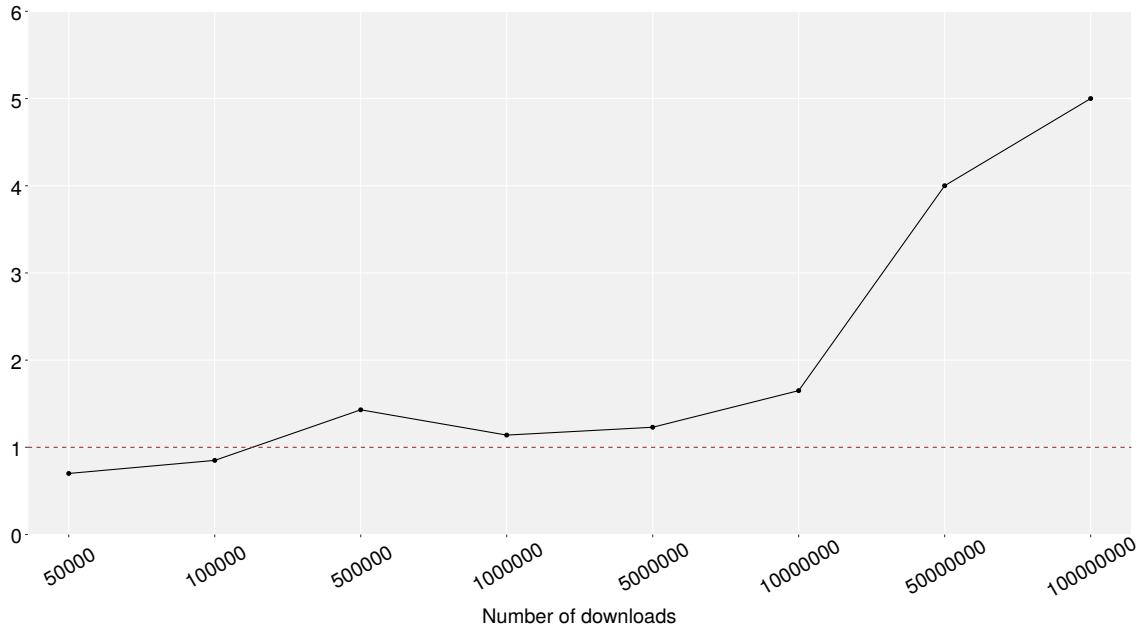

Figure 3.3: A line plot shows the ratio of the number of apps that integrate more than one ad library over the number of apps that integrate a single ad library across the number of app downloads. The **red** dotted line in the figure shows the ratio value 1.

ad libraries. For instance, the "FreeTone Free Calls & Texting"[2] app integrates 19 ad libraries; these ad libraries represent 32% of the binary size of the app. App developers integrate multiple ad libraries to increase the ad fill rate (i.e., to ensure that their apps can always display an ad) (Ruiz et al. (2014)).

Apps with a large number of downloads are more likely to integrate multiple ad libraries. Figure 3.3 presents a line plot of the multiple-ads ratio along with the number of downloads. We observe that the ratio value increases as the number of downloads increases. The increase in the ratio value from one to five indicates that apps having a high number of downloads tend to integrate multiple libraries.

To statistically measure the relationship between the number of downloads and the integration of multiple ad libraries, we performed a correlation analysis between

[2]https://play.google.com/store/apps/details?id=com.textmeinc.freetone

Table 3.5: Distribution of apps that integrate ad libraries in each app category.

App category	# of studied apps	# of ad-displaying apps	% of ad-displaying apps	Median # of integrated ad libraries	Maximum # of integrated ad libraries
Music and audio	73	69	94%	2	11
Weather	76	71	93%	5	13
Personalization	89	82	92%	3	10
Entertainment	56	46	82%	3	15
Photography	94	77	81%	2	10
Game	104	82	78%	5	17
News and magazines	79	60	76%	2	7
Tools	97	72	74%	3	10
Video players	63	45	71%	1	7
Auto and vehicles	10	7	70%	1	2
Sports	76	53	70%	2	11
Social	87	60	69%	3	19
Comics	56	36	64%	2	7
Books and reference	77	45	58%	1	9
Health and fitness	73	39	53%	2	6
Productivity	80	42	52%	2	10
Lifestyle	45	22	49%	2	7
Communication	78	37	47%	3	12
Medical	57	26	46%	1	6
Shopping	53	22	42%	1	4
Finance	38	15	39%	1	4
Maps and navigation	73	27	37%	1	3
Travel and local	77	28	36%	1	8
Education	76	27	35%	1	8
Libraries and demo	27	9	33%	1	5
Business	81	23	28%	1	5

the number of downloads and the multiple-ads ratio using the Spearman's rank-order correlation. We find that the correlation coefficient $\rho = 0.92$ (with a $p-value$ less than 0.05) which indicates that the probability of integrating multiple ad libraries increases

with the growth in the number of downloads.

Ad-displaying apps are distributed across app categories – with apps in categories having a high proportion of ad-displaying apps integrating multiple ad libraries. Table 3.5 presents the distribution of ad-displaying apps, the median number of integrated ad libraries, and the maximum number of integrated ad libraries in each app category. We observe that ad-displaying apps are distributed across app categories with some categories having a higher penetration of ads. For example, more than 90% of the studied apps in the *Music and audio*, the *Weather*, and the *Personalization* app categories integrate ad libraries.

The median number of integrated ad libraries are greater than one for 57.6% of the app categories. The Spearman's rank-order correlation between the percentage of ad-displaying apps and the number of integrated ad libraries (median) for every app category is $\rho = 0.7$ ($p - value$ is less than 0.05) which indicates that the number of integrated ad libraries increases with the growth in the proportion of ad-displaying apps within a category.

One of the possible explanations for integrating multiple ad libraries is that as the number of downloads of an app increases or the proportion of ad-displaying apps increases in a category increases the competition for ads to display from ad libraries leading to a lower fill rate. Hence, integrating multiple ad libraries increases the chance of having an ad to display and the potential ad revenue for ad-displaying apps (Ruiz et al. (2014)).

Summary of RQ1

The probability of integrating multiple ad libraries increases as the number of downloads of an app increases. App in categories with a high proportion of ad-displaying apps are more likely to integrate multiple ad libraries. We hypothesize that the integration of multiple ad libraries is a mechanism to cope with the high demand for ads in an attempt to improve the ad fill rate.

3.4.2 RQ2: How do app developers integrate multiple ad libraries?

Motivation: Integrating multiple ad libraries is a common practice in ad-displaying apps. A good understanding of how app developers integrate multiple ad libraries can help ad library developers identify the challenges and the possible improvements for their ad libraries.

Approach: To identify the strategies for integrating multiple ad libraries, two researchers (including myself and a collaborator) manually analyze several ad-displaying apps where app developers integrate multiple ad libraries as follows.

Step 1: Selecting a statistically representative sample of ad-displaying apps. Our data set has 680 ad-displaying apps that integrate multiple ad libraries. Analyzing all these ad-displaying apps manually is both difficult and time consuming. Therefore, for our manual study, we selected a statistically representative random sample of 62 apps (out of the 680 ad-displaying apps) providing us with a confidence level of 90% and a confidence interval of 10%.

Step 2: Generating a static call graph for each selected app. To understand how app developers integrate multiple ad libraries, we need to analyze the ***call-site*** source code (i.e, the packages, classes and methods that are needed to communicate with

an ad library). Hence, we decompiled the generated JARs (in Section 3.1) into Java source code files using the Class File Reader (CFR) tool (CFR (2013)). Then, we used the Understand tool (Understand (2015)) to generate and visualize the dependency call graph of each studied ad-displaying app.

Step 3: Identifying the strategies of integrating multiple ad libraries. We start our manual analysis with an open ended question "How does an app integrate multiple ad libraries?". We observe that apps differ in the way which they integrate ad libraries with respect to two practices: (1) whether the app code uses a centralized component (i.e., an ad **mediator component**) that handles the access to the multiple ad libraries and (2) whether the centralized component is written by the app developer or by the library designer.

Hence, we manually investigate every selected ad-displaying app based on the following two questions: "Does the app code call a centralized component that handles the access to the multiple ad libraries?" and "Is that centralized component written by the app developer or by the library developer?". Then, we grouped apps with similar integration behaviour (regarding the aforementioned investigated questions) as an **integration strategy**. Finally, we derived a set of rules to automatically identify the integration strategy for any unseen app.

Step 4: Analyzing the characteristics of the identified integration strategies. We ran the derived rules on the studied 680 ad-displaying apps and identified apps that belong to every integration strategy. Then, we studied the characteristics (i.e., the number of call-cite classes) of the apps that belong to every integration strategy. Finally, based on our analysis of the apps, we describe the description, an example, the benefits, and the drawbacks of each identified strategy for integrating multiple ad libraries.

Table 3.6: Distribution of apps across the different integration strategies.

Ad library integration strategy	# of apps	% of apps
Mixed strategy	329	48.3%
Self-mediation strategy	179	26.3%
External-mediation strategy	95	14.0%
Scattered strategy	77	11.3%

Findings: We identified four strategies for integrating multiple ad libraries. Table 3.6 shows the distribution of apps across the identified strategies. In the next section, we explain the identified integration strategies.

External-mediation strategy

Description of the external-mediation strategy:

In this strategy, app developers write code to integrate only one ad library that offers a mediator package. This mediator package is responsible for serving ads from other ad networks which are supported by the ad library. Since the mediator is not written by the app developers, we call this an external-mediation strategy.

Figure 3.4 shows an overview of how app developers integrate multiple ad libraries using an external mediation strategy. As shown in the Figure 3.4, app developers integrate an ad library (Ad Library 1) that has an external-ad-mediator. Every app screen that displays ads communicates only with the external-ad-mediator of the Ad Library 1. The external-ad-mediator communicates with the integrated ad libraries and serves ads from these libraries.

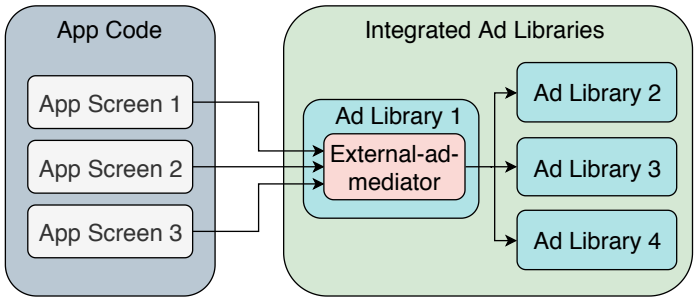

Figure 3.4: An overview of the external-mediation strategy

Rules for automatically identifying apps that use the external-mediation strategy: We determine that an ad-displaying app is using the external-mediation strategy if the following two rules are met:

1. The number of accessed ad libraries by the app code is one and the number of integrated ad libraries in the app is more than one.

2. The package of the accessed ad library contains an external-ad-mediator package that is accessed by the app code.

Example of an app that uses the external-mediaion strategy: The *"Ringtones & Wallpapers for Me"*[3] app, a popular app in the Personalization category, displays ads from ten ad libraries. The app code (i.e., the code that is written by app developers) of this app contains the code for integrating only one ad library (Google AdMob). This app uses the external-ad-mediator of the Google AdMob ad library (com.google.andorid.gms.ads.mediation) which communicates with the other nine ad libraries as well as the Google AdMob ad library for displaying ads.

[3]https://play.google.com/store/apps/details?id=com.apalon.ringtones

Benefits of using the external-mediation strategy:

- The ease of integrating multiple ad libraries is one of the main benefits of this strategy. The external-ad-mediator implements the required logic for serving ads from multiple ad libraries.

- The external-ad-mediator selects an ad library for serving ads from the supported ad libraries based on dynamically estimated measures such as the eCPM which captures the ad monetization performance of an ad library at run-time; leading to much more dynamic and accurate estimates of the revenue for a served ad (IronSource (2019)).

Drawbacks of using the external-mediation strategy:

- The external-ad-mediator of an ad library may not support all the existing ad libraries that are available in the app market. Therefore, app developers can not serve ads from the unsupported ad libraries.

  For example, we observe that only 5 out of the identified 63 ad libraries offer an external-ad-mediator. The external-ad-mediators of these five ad libraries (Google AdMob, MoPub, AerServ, Fyber and HeyZap) offer support for serving ads from only 13 ad libraries (20% of the identified ad libraries). Hence, apps that use the external-mediation strategy cannot serve ads from other ad libraries that are not supported by the external-ad-mediators.

- The entire process of serving an ad is not transparent as app developers have less control over the exact ad library from which an ad is to be served. For example,

the external-ad-mediator might provide preferential serving of ads from its network over other ad networks. This lack of transparency might cause a mistrust issue leading app developers to avoid the use of the external-ad-mediators of ad libraries (Quora (2017)).

Self-mediation strategy

Description of the self-mediation strategy:

In this strategy, app developers write their own centralized package (i.e., self-mediator) which communicates with the integrated ad libraries and manages the process of serving ads.

Figure 3.5 presents an overview of the self-mediation strategy. As shown in the Figure 3.5, all three app screens communicate with the self-mediator and the self-mediator communicates with the integrated ad libraries to serve ads from them.

Rules for automatically identifying apps that use the self-mediation strategy:

We determine that an ad-displaying app is using the self-mediation strategy if the following rules are met:

1. The number of integrated ad libraries is more than one.
2. The number of accessed ad libraries by the app code and the number of integrated ad libraries are equal.
3. The app code contains a centralized package that communicates with the integrated ad libraries.

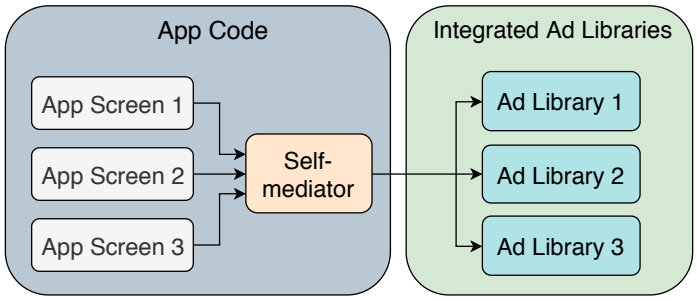

Figure 3.5: An overview of the self-mediation strategy

Example of an app that uses the self-mediation strategy:

The "*Calculator Plus Free*"[4] app, a popular app in the Tool category, integrates seven ad libraries using the self mediation strategy. The developers of the app wrote their own self-mediator (com.digitalchemy.foundation.advertising) which communicates with the integrated ad libraries to serve ads.

Benefits of using the self-mediation strategy:

- A self mediation strategy provides a good encapsulation of the code because app developers write a centralized package that manages the selection and serving of ads from multiple ad libraries.
- App developers are free to integrate any ad library instead of being limited to a handful of supported ad libraries like in the case of external mediation strategy.
- App developers have more control over selecting the ad library from which to serve an ad. We observe that app developers mainly use the following three approaches:

  A round-robin approach without a preferred list of ad libraries. In this approach, a random list of the integrated ad libraries is generated. If the first ad

[4]https://play.google.com/store/apps/details?id=com.digitalchemy.calculator.freedecimal

library in the list fails to serve an ad, the self-mediator requests an ad from the next ad library. This process continues in a circular order until an ad is served from an ad library.

A round-robin approach with a preferred list of ad libraries. In this approach, app developers set a preferred list of the integrated ad libraries based on some measures (e.g., the popularity of the ad library in a country or the offered feature of the ad library). The self-mediator selects the best preferred ad library for requesting an ad. If that library cannot serve an ad, then the mediator selects an ad library in a circular fashion from the preferred list until an ad is served from an ad library.

A custom event-based approach. In this approach, the self-mediator of an app selects an ad library based on custom events (e.g., when a user clicks a particular menu of an app or when a user earns a reward in the app). This custom event-based approach allows app developers to select the most suitable ad library for increasing user-engagement for that particular event.

Drawbacks of using the self-mediation strategy:

- App developers must write and maintain the code for the self-mediator. The self-mediator represents 8% (median) of the total number of classes of an app (in our studied apps).

  For example, the "*High-Powered Flashlight*"[5] app, a popular app in the Tool category, serves ads from 10 ad libraries which are integrated using the self-mediation

[5]https://play.google.com/store/apps/details?id=com.ihandysoft.ledflashlight.mini

strategy. The self-mediator ("com.ihandysoft.ad") consists of 22% of all the classes of this app.

- The ordering of ad libraries is static in nature. In contrast, the ordering of the external-ad-mediator is much more dynamic as it can order ad libraries based on the dynamically estimated eCPM value which is calculated at run-time based on the buying and selling of ads as conducted through real-time auctions that are facilitated by digital marketplaces (i.e., ad exchanges). Hence, app developers may not select the best ad library based on the current market conditions which in turn could prevent app developers from maximizing their ad revenue.

Scattered strategy

Description of the scattered strategy:

In this strategy, app developers neither write their own mediator nor use the external-ad-mediator of an ad library to serve ads from the integrated ad libraries. Rather, developers write code individually for each app screen to integrate each ad library for that particular screen.

Figure 3.6 shows an overview of the scattered strategy. As shown in the Figure 3.6, each app screen communicates with the integrated ad libraries directly. The integration code for the Ad Library 2 is written by the app developers for every app screen that displays an ad.

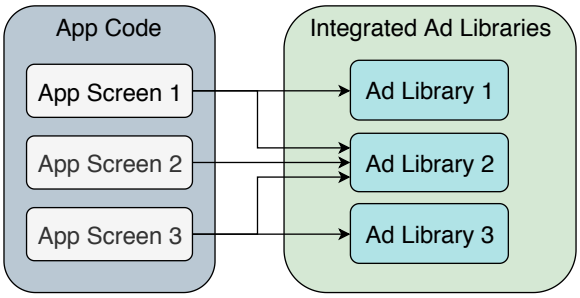

Figure 3.6: An overview of the scattered strategy

Rules for automatically identifying apps that use the scattered strategy:

We determine that an ad-displaying app is using the scattered strategy if the following rules are met:

1. The number of accessed ad library by the app code and the number of integrated ad libraries are equal.

2. The app code does not contain any centralized package that communicates with the integrated ad libraries.

Example of an app that uses the scattered strategy:

The "*Audiomack – Download New Music*"[6] app, a popular app in Music & audio category, integrates four ad libraries using the scattered strategy. The app displays ads in two screens. The developers of this app wrote code in two app screens for displaying ads from ad libraries individually.

Benefits of using the scattered strategy:

- App developers can quickly integrate several ad libraries as developers do not need to write a centralized package (e.g., self-mediator). Designing a flexible and

[6]https://play.google.com/store/apps/details?id=com.audiomack

reusable self-mediator requires effort and time which might delay the release cycle of the app.

- App developers can select ads of different ad formats (e.g., banner or native ad format) from different ad libraries based on the custom event of an app. For example the "*The Coupons App*"[7] app, a popular app in the Shopping category, integrates Google AdMob and Facebook Audience Network ad libraries in the same app screen. The app selects the Google ad library to serve banner ad format or the Facebook Audience Network to serve native ads based on the custom events of the app (e.g., clicking of a different buttons of that particular screen).

Drawbacks of using the scattered strategy:

- App developers need more effort to maintain their code because they need to write the same integration code (i.e, copy and paste) for an ad library if the ad library is integrated for displaying ads in different app screens. Therefore, the scalability and code management of an app may become difficult for app developers in a scattered strategy.
- The scattered code fetches ads from a single ad library and is not able to deal with low fill rate issues that might arise. Hence, the use of a scattered strategy may lead to a lower ad revenue.

---

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

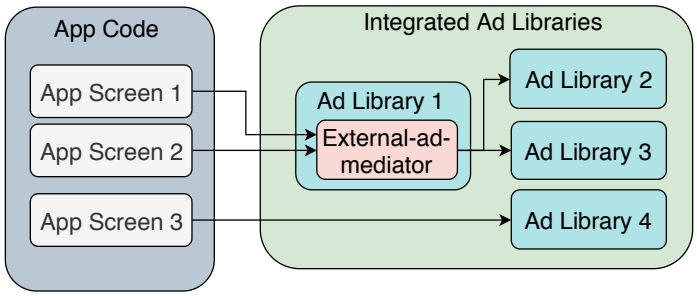

Figure 3.7: An overview of the mixed strategy

Mixed strategy

Description of the mixed strategy:

In this strategy, app developers combine both the external mediation strategy and the scattered strategy to serve ads.

Figure 3.7 shows an overview of the mixed strategy. As shown in Figure 3.7, App Screen 1 and App Screen 2 communicate with ad libraries using an external mediation strategy, whereas App Screen 3 communicates with the Ad Library 4 using a scattered strategy.

Rules for automatically identifying apps that uses the mixed strategy:

We determine that an ad-displaying app is using a scattered strategy if the following rules are met:

1. The number of accessed ad libraries by the app is less than the number of integrated ad libraries.

2. The app contains an external-ad-mediator package which communicates to many of the integrated ad libraries.

Example of an app that uses the mixed strategy:

The "*Real Guitar Free - Chords, Tabs & Simulator Games*"[8] app, a popular app in the Music category, integrates 14 ad libraries to serve ads. The developers of the app use the external-ad-mediator of the Google AdMob (com.google. android.gms.ads.mediation) ad library to integrate 11 ad libraries. To integrate the remaining three ad libraries, the app developers write code in the activities of some specific screens using the scattered strategy.

Benefits of using the mixed strategy:

- In the mixed strategy, app developers can leverage the external-ad-mediator of an ad library to serve ads from different ad libraries and can also write their integration code for other ad libraries which are not supported by the external-ad-mediator. We observe that 73.9% of ad-displaying apps with the mixed integration strategy call at least one ad library that is not supported by the currently available external-ad-mediators.

- Apps that use the mixed strategy integrate more ad libraries than the apps that use other strategies. Table 3.7 shows the mean and five-number summary of the integrated ad libraries for each of the four identified strategies. As shown in the Table 3.7, app developers integrate a maximum of 19 ad libraries using the mixed integration strategy. We find two apps from the *"TextMe, Inc"* company that integrate 19 ad libraries. We observe that they use the external-ad-mediator of the Google AdMob ad library which supports 13 ad libraries, the rest of the ad libraries are integrated using a scattered strategy since they are not supported by the external-ad-mediator of the Google AdMob ad library.

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

Table 3.7: Mean and five-number summary of each strategy for integrating multiple ad libraries.

Ad library integration strategy	Mean	Min.	1st Qu.	Median	3rd Qu.	Max.
External-mediation strategy	4.4	2	2	4	5.5	10
Mixed strategy	5.5	2	4	5	6	19
Self-mediation strategy	3.4	2	2	3	4	12
Scattered strategy	2.9	2	2	2	4	5

Drawbacks of using the mixed strategy:

Since the mixed strategy is the combination of the external-mediation strategy and the scattered strategy, all the drawbacks of these two strategies are exist in the mixed strategy. In addition, we wish highlight one additional drawback for the mixed strategy.

- An app that uses the mixed strategy can integrate the same ad library using the external-mediation and the scattered strategy (based on the needs of the app). Such an integration process might cause race conditions and ad synchronization issues.

Summary of RQ2

App developers dominantly use the mixed and the self-mediation strategy to integrate multiple ad libraries. This might be due to the currently available external-ad-mediators not satisfying their needs. To have more control on selecting ad libraries for displaying ads, app developers write their own centralized packages (self-mediator) based on preferred metrics (e.g., location information) or custom app events in the self-mediation strategy.

3.5 Discussion of the maintenance overhead of the integrated ad libraries for each integration strategy.

In this section, we discuss how app developers maintain their integrated ad libraries over time across the different ad library integration strategies. In particular, we discuss the modifiability of the ad-call-site code (i.e., the classes of the app code that invoke the method for integrating an ad library) and the flexibility of integrating ad libraries for each integration strategy.

3.5.1 The modifiability of ad-call-site code

In this section, we discuss the modifiability of the ad-call-site code along two aspects: (1) how frequently (in terms of the proportion of the updates of an app) do app developers modify the ad-call-site code, and (2) what is the proportion of the ad-call-site code that is modified across the integration strategies.

To determine if an ad-call-site code is modified, we follow the same approach that is presented by Ruiz et al. (2016). In this approach, for each update of an ad-displaying app that integrates multiple ad libraries, we generate the class signatures for all ad-call-site code (we consider only the statements that invoke ad library methods) of the integrated ad libraries. The ad-call-site code is modified in the app update (U_{i+1}) if the signature of the app update (U_{i+1}) is different than the signature of the app update (U_i).

The probability of modifying the ad-call-site code is 20% (median) across all integration strategies, with that probability increasing considerably to 37% (an increase of 60%) for ad-displaying apps which use the mixed strategy.

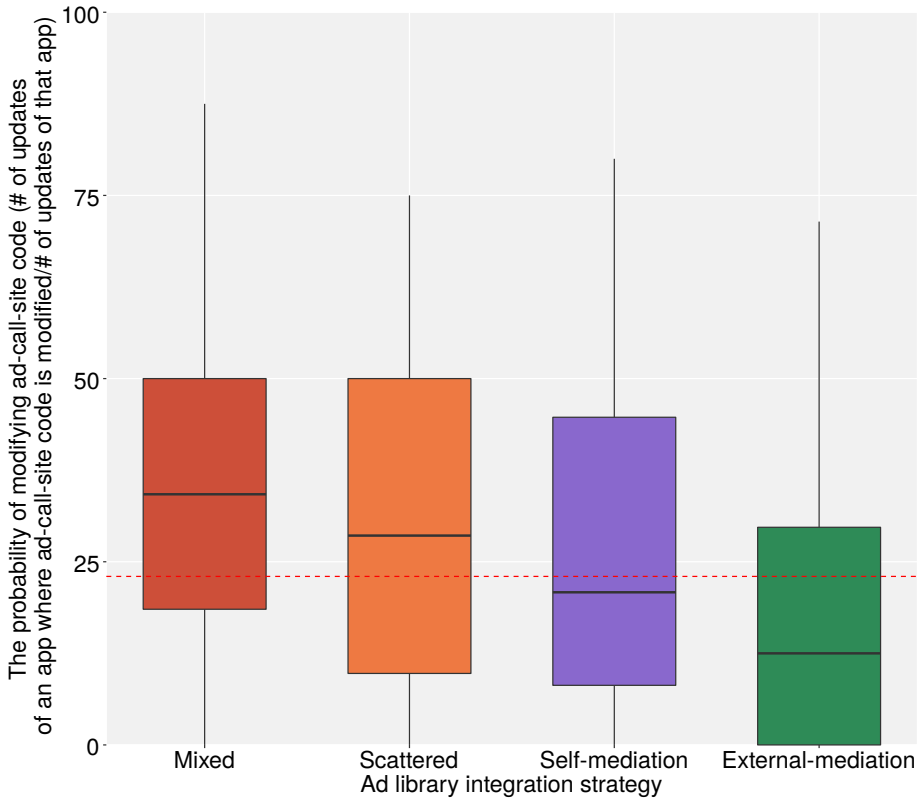

Figure 3.8: The probability of modifying the ad-call-site code in a update for each ad library integration strategy. The red dotted line shows the median probability of modifying the ad-call-site code.

Figure 3.8 shows the probability of modifying ad-call-site code for every integration strategy. As shown in the Figure 3.8, we observe that the probability of modifying the ad-call-site code for ad-displaying apps that use the mixed or the scattered strategies well above the median. We also observe that the median probability of modifying the ad-call-site code for apps that use the mixed strategy is twice than the median probability of modifying ad-call-site code for apps that use the external mediation. This result indicates one of the drawbacks of the scattered strategy that app developers who use the scattered strategy need to modify the ad-call-site code in a much larger proportions of the deployed updates of their apps.

Table 3.8: The percentage of the modified ad call-site-code when an ad library is updated and when an ad library is not updated.

Ad library integration strategy	**% of the modified ad-call-site code (median)**	
	when the integrated ad library is updated	**when the integrated ad library is not updated**
Mixed strategy	12.5	0.0
Scattered strategy	8.0	0.0
Self-mediation strategy	3.3	0.0
External-mediation strategy	0.0	0.0

The proportion of the modified ad-call-site code (# of ad-call-site code that is modified / # of total ad-call-site code) is highest in the apps that use the mixed strategy. We observe that app developers mostly modify ad-call-site code when they update their integrated ad libraries. Table 3.8 shows the proportion of the modified ad call-site-code in two cases: when the integrated ad library is updated and when the integrated ad library is not updated. As shown in the Table 3.8, we observe that the proportion (median) of the modified ad-call-site code is zero for each integration strategy (when the ad library is not updated) indicating that app developers usually do not optimize or change their ad library integration code. We also observe that in the case when an ad library is updated, the proportion of the modified ad-call-site code for the apps that use the mixed strategy is the highest whereas the proportion is almost zero for the apps that use the external-mediation strategy. This result is another indication that the mixed strategy may require more effort to maintain the ad-call-site code over time.

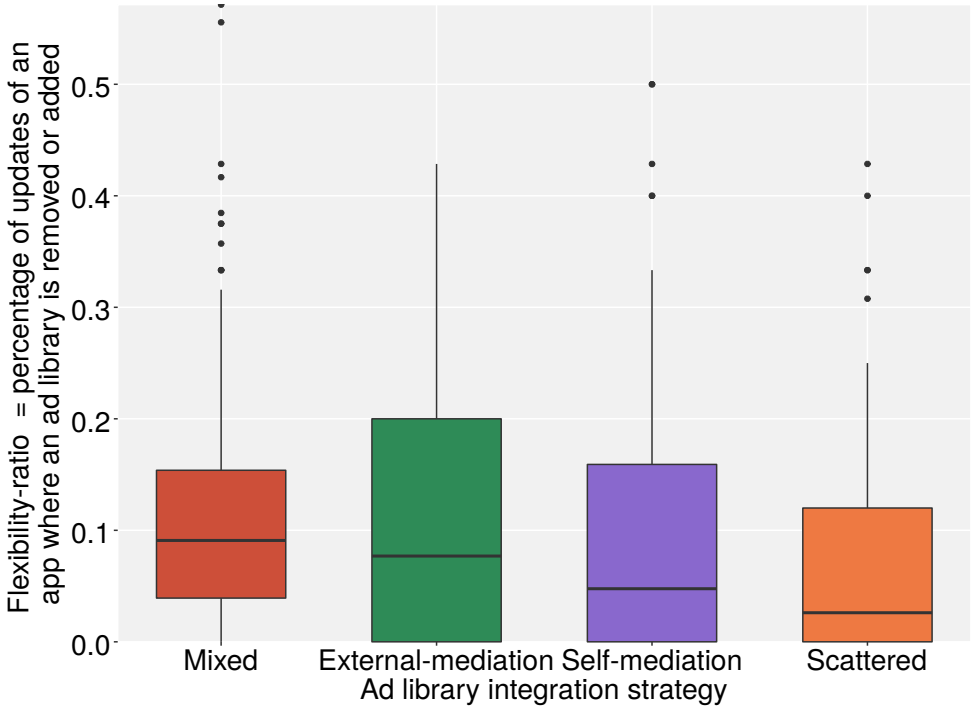

Figure 3.9: The flexibility-ratio for each of the integration strategies.

3.5.2 The flexibility of integrating ad libraries.

To understand which ad integration strategy is more flexible for modifying (adding or removing) ad libraries, we calculate a *flexibility-ratio* for each integration strategy. A *flexitibility-ratio* is the ratio of the number of updates of an app in which the app developer adds or removes an ad library to the total number updates of the app.

The mixed strategy has the highest flexibility-ratio indicating that this strategy provides developers with the highest flexibility. Figure 3.9 shows the *flexibility-ratio* for each identified strategy. As shown in the Figure 3.9 the mixed strategy has the highest *flexibility-ratio*. We identify ad libraries that are added or removed in the cases of mixed strategy and observe that all these ad libraries are supported by the currently available external mediators that are currently in use by these apps. One explanation of

this result is that developers do not need to write or update any code to add or remove ad libraries. This hypothesis explains as well the high *flexibility-ratio* for the external mediation strategy.

3.6 Implications

In this section, we describe the implications of our study of ad library integration practices for ad library developers.

The developers of the Google AdMob should spin out their functionality for uniquely identifying a user's device out of their ad library. As described in Section 3.4.1, analytics libraries have a dependency on the Google AdMob ad library. These analytics libraries depend on one of the packages of the Google AdMob ad library named "com.google.android.gms. ads.identifier" for the unique identification of a user's device. These analytics libraries need this functionality to track a user's in-app behavior. However, the main purpose of an ad library is to serve ads on a user's device. This unusual dependency on the Google AdMob increases the size of many apps that use these analytical libraries. Hence, we recommend that developers of the Google AdMob ad library should rethink their design and offer a separate library for uniquely identifying a user's device.

Ad library developers should improve their external-ad-mediators by (1) enabling the integration of new ad libraries at run-time and (2) increasing the supported ad libraries. In Section 3.5, we observed that app developers use the mixed strategy to achieve the highest flexibility (i.e., continuously adding or removing an add library). To improve the flexibility of the external-ad-mediators, ad library developers need to provide some standardized interfaces to enable the integration of new ad libraries for app

developers at run-time instead of only at design time. For example, the Google AdMob ad library has started offer an SDK-less mediation feature that enables app developers to add or delete any new ad library by re-configuring their Google ads account (without a need for deploying an update that adds/removes the required ad libraries) (Google (2016)).

Another dimension for improving the external-ad-mediators is to add support for a large number of ad libraries. For 3.4.2, we observed that 73.9% of the ad-displaying apps that use the mixed integration strategy call at least one ad library that is not supported by any external-ad-mediators. Hence, we recommend ad library developers who offer external-ad-mediators to support more ad libraries so that app developers can choose different ad libraries and maximize ad revenue. For example, ad libraries should offer standardized interfaces to their mediators to support for additional ad libraries can be added by app developers instead of app developers being locked to a limited number of supported ad libraries.

Ad library developers should offer more feature control over the selection of ad libraries. In Section 3.4.2, we observed that the external-ad-mediator selects an ad library from the integrated ad libraries based on dynamically calculated eCPM value. Although this selection process is useful for accurately estimating the revenue for a served ad, it might prevent an app from achieving an improved user-engagement as the process is not customizable. In addition, this process is not transparent to app developers (as eCPM is calculated dynamically) and does not allow app developers to control the selection of ad libraries. We observe that app developers write code (8% (median) of the total number of classes of the app code) for their self-mediator which offers them custom control when selecting ad libraries based on a preferred list of ad

libraries (e.g., a list of ad libraries based on the popularity of ad libraries in a country) or custom app events. Therefore, we recommend ad library developers to offer a more configurable interface for their external-ad-mediators so that app developers can have more control in selecting ad libraries when they desire.

3.7 Threats to Validity

External Validity: In this study, we only focused on the top free-to-download Android apps from the Google Play Store. Future studies should broaden the scope of our study and investigate how our findings apply to ad libraries that are integrated in other types of apps, such as Windows apps or iOS apps.

Internal Validity: In our analysis for identifying strategies for integrating multiple ad libraries, we manually investigated apps that integrate more than one ad library. In this analysis, we cannot deny the possibility of misinterpreting of the identified strategies for integrating multiple ad libraries since we are not the original developers of the studied apps. To mitigate this threat, the first and the second author leveraged the Understand tool to analyze the call graph of the apps, carefully investigated each of the sampled apps and consolidated their results.

3.8 Chapter Summary

In the mobile app economy, ad libraries play an integral role. App developers integrate one or many ad libraries to display ads and gain revenue based on user interactions with the displayed ads. Even though ad libraries play an essential role in the app ecosystem, there has been no prior studies of how these libraries are integrated by app

developers. In this paper, we analyze 1,840 top free-to-download apps to study the ad library integration practices. The most important findings of our study are:

1. While the ad market is heavily saturated by the Google AdMob and Facebook Audience Network ad libraries, there remain opportunities for other players in this market with ad-displaying apps opting to integrate several libraries.

2. The integration of multiple ad libraries is common in ad-displaying apps that have a large number of downloads and in app categories that have a high proportion of ad-displaying apps.

3. App developers use four strategies to integrate multiple ad libraries: *(1) external-mediation strategy* (app developers use an external-ad-mediator that is provided by an ad library and do not write their own code), *(2) self-mediation strategy* (app developers write their own centralized package (self-mediator), *(3) scattered strategy* (App developers do not write a mediator package instead they write separate code for each of the integrated ad libraries) and *(4) mixed strategy* (app developers use both the external-mediation strategy and the scattered strategy).

4. The current mediation offerings of ad libraries are not satisfying the needs of developers as we observe that 73.9% of the apps with the mixed integration strategy call at least one ad library that is not supported by the external mediator. Hence, we recommend that ad library developers should offer mechanisms to support a larger number ad libraries in their external mediators.

5. App developers write their centralized package (self-mediator) where they use several approaches (e.g., a round-robin or a custom-based approach) for selecting an ad library from which to serve ads. The self-mediation strategy allows app

developers to have more control on selecting their ad libraries than the dynamic selection of ad libraries in the external mediation. Hence, we recommend that ad library developers to offer app developers more control over the selection of ad libraries.

6. By analyzing the flexibility of integrating ad libraries, we observe that app developers use the mixed strategy to achieve the highest flexibility (i.e., continuously adding or removing an ad library). Hence, we recommend that ad library developers need to provide some standardized interfaces to enable the integration of new ad libraries for app developers at run-time instead of only at design time.

CHAPTER 4

A Longitudinal Study of Popular Ad Libraries in the Google Play Store

In this chapter, we study the evolution of the 8 most popular ad libraries (e.g., Google AdMob and Facebook Audience Network) over a period of 33 months (from April 2016 until December 2018). In particular, we look at their evolution in terms of size, the main drivers for releasing a new version, and their architecture. To identify popular ad libraries, we collect 35,462 updates of 1,840 top free-to-download apps in the Google Play Store. Then, we identify 63 ad libraries that are integrated into the studied popular apps. We observe that an ad library represents 10% of the binary size of mobile apps, and that the proportion of the ad library size compared to the app size has grown by 10% over our study period. By taking a closer look at the 8 most popular ad libraries, we find that ad libraries are continuously evolving with a median

release interval of 34 days. In addition, we observe that some libraries have grown exponentially in size (e.g, Facebook Audience Network ad library), while other libraries have attempted to reduce their size as they evolved. The libraries that reduced their size have done so through: (1) creating a lighter version of the ad library, (2) removing parts of the ad library, and (3) redesigning their architecture into a more modular one.

To identify the main drivers for releasing a new version, we manually analyze the release notes of the eight studied ad libraries. We observe that fixing issues that are related to displaying video ads is the main driver for releasing new versions. We also observe that ad library developers are constantly updating their libraries to support a wider range of Android versions (i.e., to ensure that more devices can use the libraries without errors). Finally, we derive a reference architecture from the studied eight ad libraries, and we study how these libraries deviated from this architecture in the study period.

Our study is important for ad library developers as it provides the first in-depth look into how the important mobile app market segment of ad libraries has evolved. Our findings and the reference architecture are valuable for ad library developers who wish to learn about how other developers built and evolved their successful ad libraries. For example, our reference architecture provides a new ad library developer with a foundation for understanding the interactions between the most important components of an ad library.

4.1 Introduction

In-app mobile advertising is a growing market with a forecasted revenue of $201 billion by 2021 (AppAnnie (2017)). Since the majority of the apps are free-to-download (AppBrain (2019)), app developers use in-app advertising as their primary revenue model (De La Iglesia and Gayo (2009)). In this model, app developers display advertisements (*ads*) to app users and gain revenue based on the number of displayed ads and the user's interactions with these ads.

To display advertisements, every ad network provides an ad library that needs to be integrated into the integrating apps (i.e., apps that integrate ad libraries). The main functionality of these ad libraries is to take care of the communication with the ad network and to display ads to app users.

Although ad libraries are an integral part for app revenue, prior studies show that ad libraries can add to the development effort for app developers and can have a negative impact on the integrating app (e.g., they can increase the energy consumption of the app (Gui et al. (2015)), or they can negatively affect the user-perceived quality (Hassan et al. (2018))).

Despite the integral role of ad libraries in the mobile app ecosystem, there have been no prior studies that analyze how these libraries evolve over time. Understanding this evolution is essential for developers who wish to build or evolve their own ad libraries.

To motivate our work, we conducted an initial study on the evolution of the size of 1,840 popular app binaries and their integrated ad libraries as follows. First, we identified the list of 63 ad libraries that are integrated into the studied apps (Section 4.2.2

describes our process for identifying the integrated ad libraries). Second, for each update of an app, we calculated the app binary size and the combined size of the integrated ad libraries in such update. Then, we calculated the app size in every month as the average of the app binary size for all app updates that were deployed within that month. Similarly, we calculated the monthly ad library size of an app as the average of the combined size of the integrated ad libraries for all app updates that were deployed within a month. Figure 4.1 shows the size of the studied app binaries and the combined size of their integrated ad libraries during 18 months (from April 20^{th} 2016 to September 20^{th} 2017). As shown in Figure 4.1, both the app binary and the combined ad library size increased over time.

An interesting observation is that the proportion of an app that consists of ad libraries increased in the studied 18 months. We calculated the growth ratio of this proportion for an app (A) as follows:

$$Ad\ library\ growth\ ratio\ (A) = \frac{PAL_{end}(A)}{PAL_{start}(A)} \tag{4.1}$$

Where $PAL_{start}(A)$ and $PAL_{end}(A)$ are the proportion of ad libraries of app A (i.e., the ratio of the ad libraries size and the app binary size) at the start and the end of the studied 18 months. A growth ratio that is larger than one indicates that the proportion of ad libraries increased for app A during the studied period. A growth ratio that is smaller than one indicates that the proportion of ad libraries decreased for app A. We find that the median growth ratio is 1.1, which indicates that the proportion of an app that consists of ad libraries increased by 10% during the studied 18 months.

To learn more about the interesting phenomenon of ad library evolution, and to investigate why the proportion of ad libraries in an app is increasing, in this chapter we

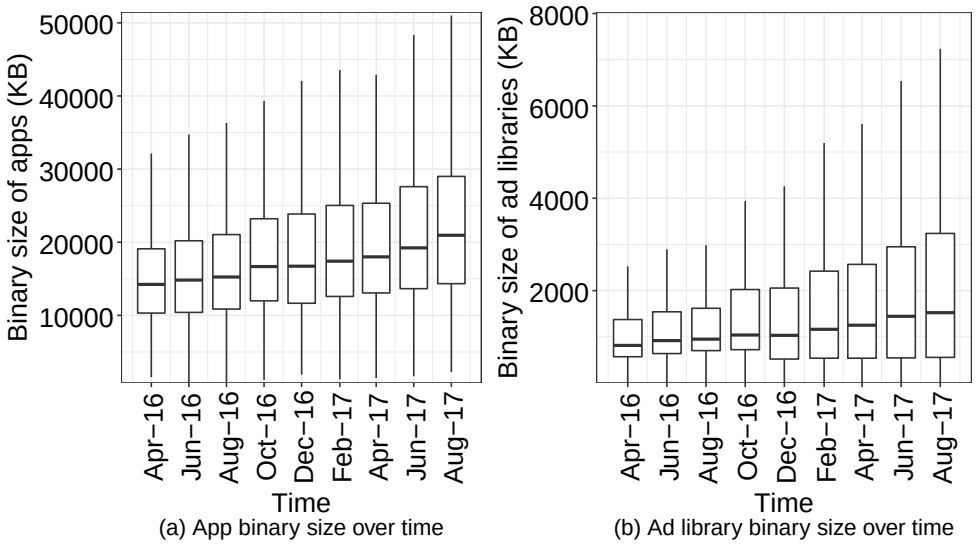

Figure 4.1: The size of apps and ad libraries during a 1.5 year period.

conduct a longitudinal study of the eight most popular ad libraries that are integrated by popular free-to-download apps in the Google Play Store. In particular, we study the evolution of ad libraries over a period of 33 months (from April 2016 until December 2018) by addressing the following three research questions (RQs):

RQ1: *How often are ad libraries released, and how large are these releases?*

Ad libraries have a median of one release per month. While the size of the ad libraries is increasing, a few ad library developers are taking measures to constrain the library growth since larger apps are less likely to be installed by users (Reinhardt (2016); Google (2018)). The followings are some of the measures that we observed: (1) releasing a lighter version of the ad library, (2) redesigning their architecture into a more modular architecture, and (3) removing components from the ad library.

RQ2: *What drives ad library developers to release a new version?*

We manually read the release notes of the released versions of popular ad libraries during our study period to identify the main drivers for each version. We find that fixing issues that are related to displaying video ads is the main driver to release new versions during our study period. In addition, a common driver for releasing is to add support for a new Android version, thereby increasing the range of users to which an app can display ads using that particular library.

RQ3: *How did the architecture of ad libraries evolve over time?*

In order to avoid common pitfalls, it is essential for developers who wish to build or evolve their own ad libraries to understand how other ad libraries were designed. Therefore, we derived a reference architecture from the studied ad libraries. We show that as ad libraries evolve, their architectures tend to converge towards this reference architecture. In addition, we observe that ad libraries use different display components to support the display of ads of various media types.

The main contributions of our study are as follows:

1. We are the first to conduct a longitudinal study of ad libraries. Our work provides valuable insights into the evolution of ad libraries.

2. We provide an in-depth analysis of the main drivers for ad library developers to release a new version. These drivers can help researchers and software developers better understand the challenges of developing ad libraries.

3. We propose the first reference architecture for ad libraries. This architecture is helpful for developers who wish to build their own ad libraries to understand

the interactions of the most important components of an ad library. In addition, as noted by prior work (Grosskurth and Godfrey (2005); Roy et al. (2017); Addo et al. (2014); Medvidovic and Taylor (2000); Dueñas et al. (1998); Hassan and Holt (2002); Medvidovic and Jakobac (2006); Hassan and Holt (2000)), a reference architecture for a domain provides a common vocabulary for a domain, enabling developers and others (e.g., researchers) to discuss concepts and concerns at a much higher level of abstraction (i.e., domain wide) instead of being fixated with the peculiarities of particular implementations (i.e., the naming of a package in a particular library).

The rest of the chapter is organized as follows. Section 4.2 describes our data collection process. Section 4.3 presents the results of our longitudinal study of ad libraries. Section 4.4 describes the quality attributes of our derived reference architecture for ad libraries. Section 4.5 discusses the implications of our work. Section 4.6 describes the threats to validity of our findings, and Section 4.7 concludes the chapter.

4.2 Data Collection

In this section, we describe our data collection process. Our data collection process contains three main steps. First, we collected data of the most popular free-to-download apps in the Google Play Store. Then, we used the collected data to identify the eight most popular ad libraries that are integrated into apps. Finally, we downloaded the release notes and the bytecode (JAR files) for all the versions of the identified popular ad libraries. Figure 4.2 gives an overview of the main steps of our data collection process. We detail each step below.

4.2.1 Collecting Updates of the Top Free-to-Download Apps

In this step, we collect the deployed updates (i.e., the APK files) of the top free-to-download apps in the Google Play store. We focus on the top free-to-download apps because these apps have a large number of active users, and are therefore a good example of how ad libraries are integrated in successful apps (as opposed to malicious apps which may overwhelm a user with ads).

We used App Annie's report of popular apps in 2016 (AppAnnie (2018)) to identify popular apps. We selected the top 100 apps in each of the available 28 app categories (e.g., the communication and game categories) in the Google Play store. We found that 214 apps were repeated across app categories and 746 apps were already removed from the store at the start of our study period. In total, we selected 1,840 apps for our study. Then, we ran a custom crawler that is based on Akdeniz' Google Play crawler (Akdeniz (2013)) for 18 months from April 20^{th} 2016 to September 20^{th} 2017 to collect all deployed updates of the selected apps. At the end of this step, we collected 35,462 updates of 1,840 apps that were deployed during our study period.

4.2.2 Identifying Popular Ad Libraries

To identify popular ad libraries, we followed a similar approach to the one presented by Ruiz et al. (2016):

Step 1: Identify the integrated ad libraries in every update. First, we converted the collected APKs to JARs using the dex2jar tool (dex2jar (2016)). Then, we used the BCEL tool (Apache BCEL (2018)) to extract the fully qualified class names (i.e., the class name and the package name) of all classes in the generated JARs. Since prior studies showed that an ad library's packages or class names contain the term "ad" or "Ad" (Li

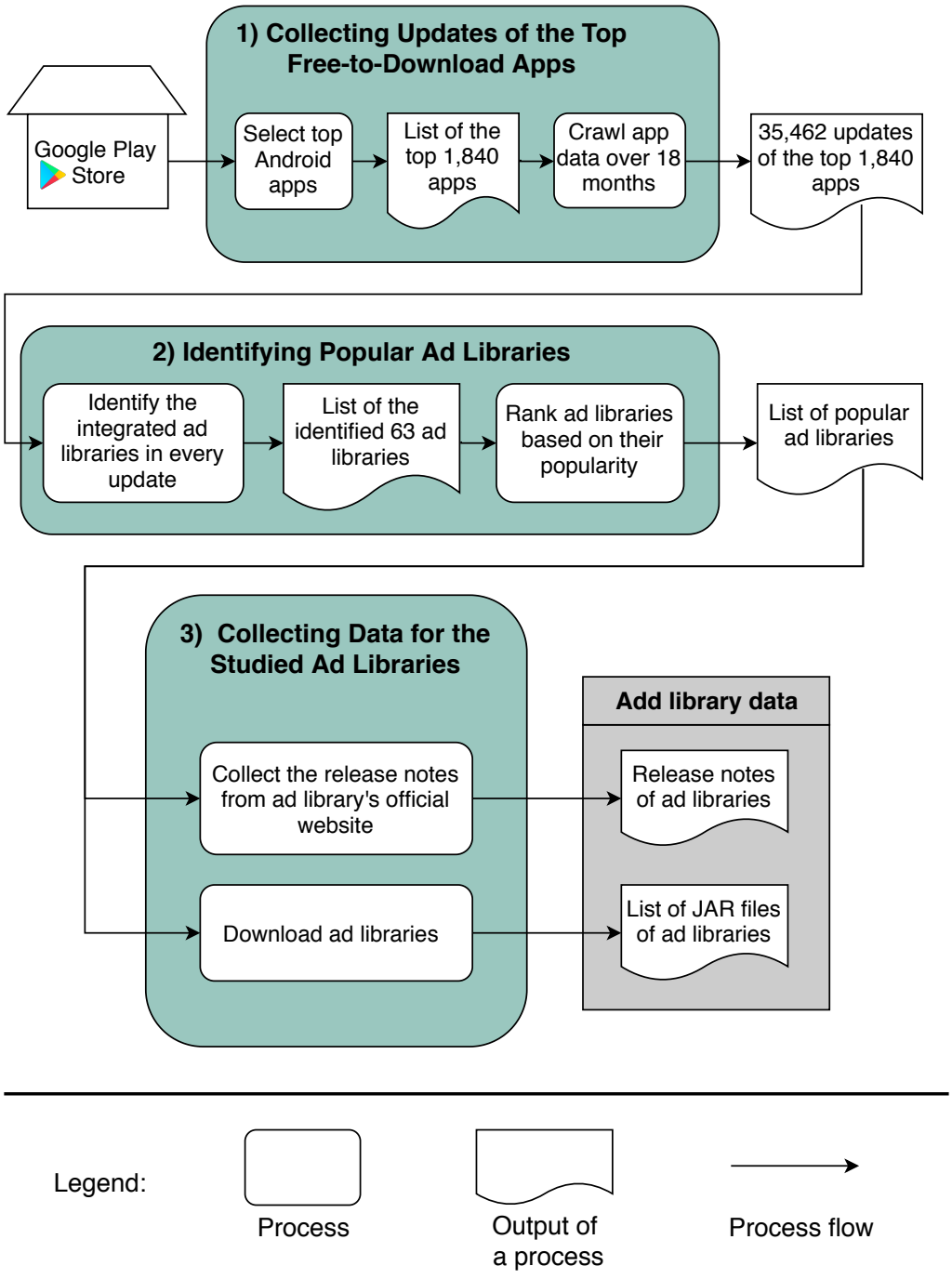

Figure 4.2: An overview of our data collection process.

Table 4.1: Statistics for the identified top ten popular ad libraries (sorted by the percentage of integrating apps).

Ad library	# of apps integrating this library	% of apps integrating this library
Google AdMob	1,310	71.2%
Facebook Audience Network	513	27.9%
MoPub	292	15.9%
Amazon Mobile Ad	129	7.0%
Millennial Media*	118	6.4%
AdColony	111	6.0%
InMobi*	106	5.8%
Unity Ads	104	5.6%
Vungle	82	4.5%
Flurry	80	4.3%

*These ad libraries were not included in our study because we could not locate a JAR file for each of the releases of these libraries.

et al. (2016)), we filtered the fully qualified class names using the regular expression "[aA][dD]". However, this regular expression also matches class names that are not related to ad libraries (e.g., `com.fbox.load.ImageLoad`). Hence, to identify ad libraries, we followed the same approach of Ruiz et al. (2016) by manually verifying the package name of the matching classes on the web. We manually verified 303 packages on the web. In total, we identified 63 ad libraries. The Appendix describes the list of 303 packages that we manually analyzed on the web and the list of identified 63 ad libraries.

Step 2: Rank ad libraries based on their popularity. For each ad library, we calculated the number of apps that integrates this particular library to represent its popularity. Then, we ranked the 63 ad libraries based on their popularity. We focused our study on the top ten popular ad libraries as these are the most integrated libraries by the studied top free-to-download apps: 96% of the studied apps that display ads integrate one or more of these libraries. Table 4.1 shows the top ten popular ad libraries.

4.2.3 Collecting Data for the Studied Ad Libraries

To analyze the evolution of the identified popular ad libraries, we downloaded the release notes and the bytecode (JAR files) of the libraries. We collected the release notes and JAR files of all versions that were released during our study period (from April 2016 to December 2018) from the official library website. We could not find JAR files for all releases of the InMobi and Millennial Media ad libraries. Therefore, we removed these two ad libraries from our analysis. In total, we collected the release notes and the JAR files of 163 released versions of the 8 most popular ad libraries during our study period.

In the following section, we describe the results of our analysis of the collected data.

4.3 Case Study Results

In this section, we present our longitudinal study of the evolution of ad libraries. For each research question, we discuss the motivation, approach, and results.

4.3.1 RQ1: How often are ad libraries released, and how large are these releases?

Motivation: In Section 4.1, we observed that the median size of the ad libraries increased during our study period. The size of an app (and hence the libraries it uses) is important, as prior studies show that larger apps are less likely to be installed by users (Reinhardt (2016); Google (2018)). Hence, we study the increase in size and the frequency of ad library releases. Understanding how ad library developers manage the size of their ad libraries can help developers who wish to develop or evolve their own ad libraries to manage the size of their own libraries.

Table 4.2: Median release interval in days, and the number of versions of each ad library (sorted alphabetically by ad library name).

Ad library	**Median release interval (in days)**	**Total # of versions**	**# of major versions**	**# of minor versions**	**# of patch versions**
AdColony	30	18	0	4	14
Amazon Mobile Ad	81	5	0	1	4
Facebook Audience Network	34	31	1	18	12
Flurry	21	32	5	15	12
Google AdMob	30	21	7	9	5
MoPub	37	26	2	19	5
Unity Ads	23	20	1	3	16
Vungle	67	10	3	4	3
Overall	34	163	19	73	71

Approach: For every ad library version, we measured the number of days between releasing it and the following version (the *release interval*). Then, for every ad library, we calculated the median release interval of all released versions of that library. Using release versioning rules (Preston-Werner (2013)), we also calculated the number of major, minor, and patch versions of the studied ad libraries. Finally, to analyze the change in size of an ad library, we measured the size in kilobytes (KB) of the released versions of that library.

Findings: **The studied ad libraries had a median release interval of approximately one month.** Table 4.2 shows the number of released version, the median release interval (in days), and the number of released major/minor/patch versions for the studied ad libraries. As shown in Table 4.2, all studied libraries had at least five versions during the study period with a median release interval of 34 days.

We observe that the majority of the released versions are minor and patch versions. However, 75% of the studied libraries had at least one major version during the study period.

The size of all the studied ad libraries (except Google AdMob and Vungle) increases over time. Figure 4.3 shows the size of the studied ad libraries during the study period. We identified four trends in the size evolution of the studied ad libraries:

1. *Explosive growth:* Facebook Audience Network, Unity Ads
2. *Stable growth:* MoPub, Amazon Mobile Ad
3. *Shrinkage:* Google AdMob, Vungle
4. *Fluctuating size:* AdColony, Flurry

As shown in Figure 4.3, the size of the Facebook Audience Network library increased by 297% during our study period. To understand the rationale for the explosive growth, we studied the release notes of the Facebook Audience Network library. We observed that the Facebook Audience Network library added several video streaming features during the study period, which contributed to its size increase. The other ad libraries also have video streaming functionality; however, because the Facebook Audience Network ad library started out relatively small compared to the other ad libraries, the impact of the new features on the growth rate of this library was more prominent.

Table 4.3 shows the median size (in KB) of the ad libraries. Interestingly, the Vungle ad library is considerably larger than most of the other ad libraries. For example, the size of Vungle versions *4.0.3* (2,327 KB) and *5.1.0* (2,229 KB) is twice the median size of the other ad libraries (1,048 KB).

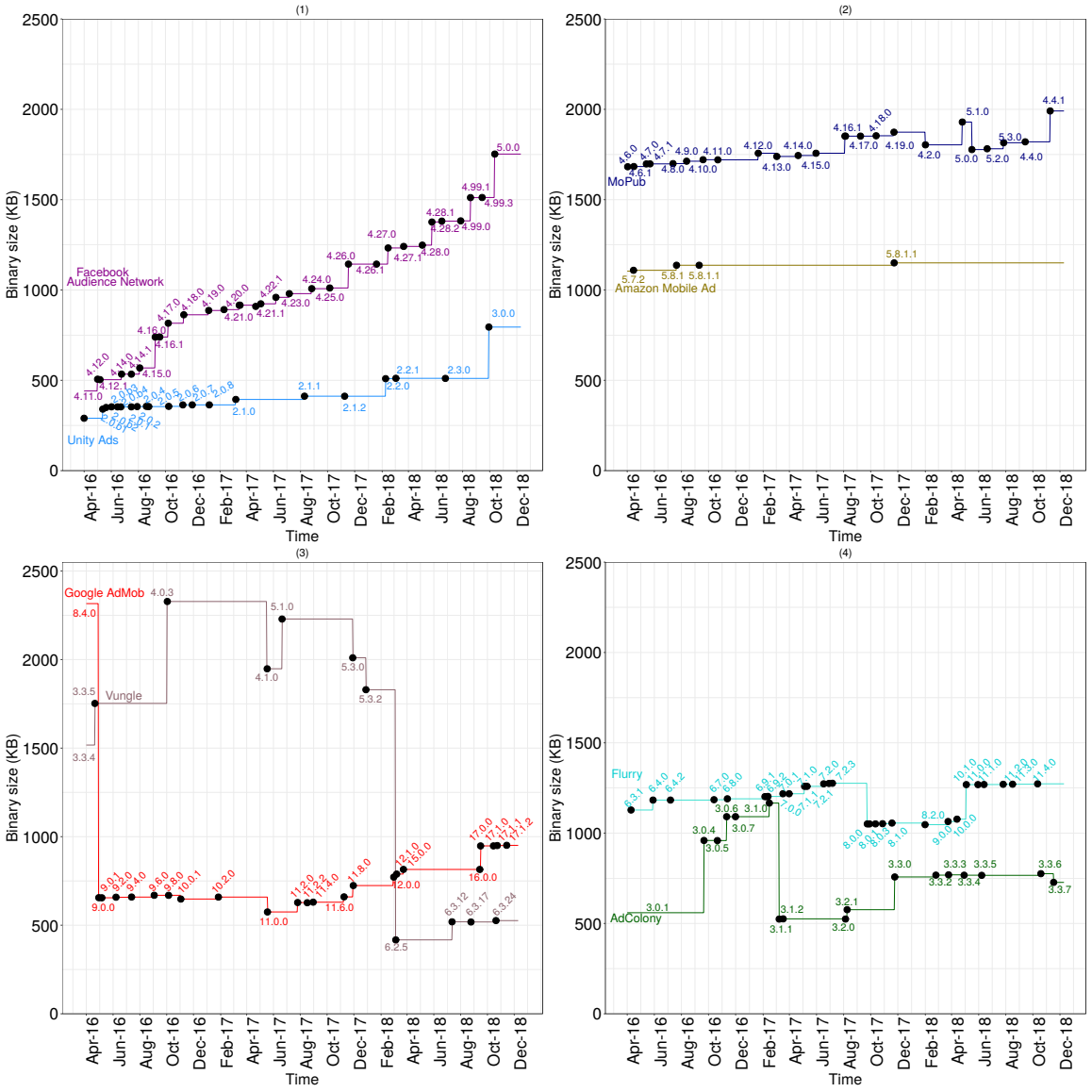

Figure 4.3: The identified trends in the size evolution of the studied ad libraries: (1) *Explosive growth*, (2) *Stable growth*, (3) *Shrinkage*, and (4) *Fluctuating size*.

We analyzed the release notes and the source code of these Vungle versions. We observe that the major increase in the release size occurred because Vungle integrated the RxJava library (RxJava (2013)) whose size of 924KB, is almost half of the Vungle library

Table 4.3: The median size of the studied ad libraries (sorted by the median ad library size).

Ad library	**Median ad library size (KB)**
Vungle	1,792
MoPub	1,767
Flurry	1,202
Amazon Mobile Ad	1,137
Facebook Audience Network	941
AdColony	768
Google AdMob	664
Unity Ads	360

size. The RxJava library is useful for applications that are designed for reactive programming (ReactiveX (2013)) (e.g., mobile apps that need to respond to user clicks). We observe that Vungle leveraged the RxJava library to improve the communication between the ad library and the Vungle ad network.

Although the size of most ad libraries increased, ad library developers took measures to limit the growth in size. Figure 4.3 shows that some ad library releases (such as version *9.0.0* of Google AdMob and version *3.1.1* of AdColony) are considerably smaller than their predecessors. To further understand this decrease in size, we investigated the code changes (i.e., the changed classes and packages) and the release notes of the releases that were smaller than their predecessors. We identified three approaches that were used by ad library developers to reduce the size of their library:

1. **Creating a lighter version of the ad library.** Google provides a set of APIs called the Google Play Services APK which are installed by default on user devices. This APK contains the common features that are needed to communicate with the Google Play Store. Starting from Google AdMob version *9.0.0*, Google introduced

Google Ads Lite (Google (2019a)). This lite version does not contain the code that communicates with the store to fetch ads. Rather, the Google Ads Lite library depends on the installed APK to communicate with the Google Play Store which reduces the library size (Google (2019a)). In our analysis, we observed that the newly-released lightweight version decreased the library size by 70%. On the other hand, removing parts of the ad library code creates a dependency on the installed APK when retrieving advertisements. Hence, creating a lighter version of an ad library may introduce new risks for integrators, as the lighter version adds a layer of complexity to the integration of the ad library.

2. **Extracting the functionality of an ad library into independent modules.** In version *4.9.0*, MoPub introduced a modular architecture that separates the functionality of the ad library into the following five different modules: (1) the *"banner ads"* module that displays banner (i.e., image-based) ads, (2) the *"native ads"* module that displays ads with the same look and feel as the integrating apps, (3) the *"video ads"* module that displays video ads, (4) the *"interstitial ads"* module that displays full-screen ads, and (5) the *"reward video ads"* module that allows users to receive rewards based on the displayed video ads.

   The size of each module is smaller than the original size of the full MoPub ad library. Hence, integrating apps can reduce their size by integrating only the necessary modules. For example, if an integrating app displays only banner advertisements, the app only has to integrate the banner ads module. According to the MoPub website, their modular architecture allows app developers to save up to 60% of the library size by including only the needed modules (MoPub (2016)).

3. **Removing components from an ad library.** The AdColony library (version *3.0.2*) introduced an additional component called *Compass* which resulted in a 108% increase in the overall library size. To increase user's engagement with the displayed ads, *Compass* provides the following features: (1) promoting other apps of the app developers, (2) engaging users with in-app notifications, and (3) rewarding users for in-app purchases. Later, the AdColony developers removed Compass from their ad library which reduced the library size by 55%. Although we have no evidence that Compass was removed to reduce the ad library size, the fact that the developers did not opt to disable Compass (rather than remove it completely) could be an indication that size played a role in the removal.

   In another example, we observe that the design of the Vungle library (version *6.2.5*) was revised to handle ad events (e.g., an event to initialize an ad) without using the RxJava features which reduced the size of ad library by 77%.

Summary of RQ1

The studied ad libraries had a median release interval of a month during the studied 33 months. Although the size of most ad libraries keeps on increasing, ad library developers take measures to reduce the size of their libraries, such as: (1) creating a lighter version of the ad library, (2) extracting the functionality of an ad library into independent modules, and (3) removing components from an ad library.

Table 4.4: The identified drivers for releasing ad library versions.

Driver category	**Driver name**	**Description (*D*) - Example (*E*)**
Internal code fixing	Fix a crash or exception	***D:*** The version fixes a crash or exception in an ad library. ***E:*** *"Fixed crash when interacting with the screen after rewarded video finishes and before showing the end card."*
	API refactoring	***D:*** The version refactors (e.g., adds, removes, or modifies) methods or classes that are related to the ad library APIs. ***E:*** *"Removed the deprecated EventListener.onVideoView() API."*
	Improve code obfus--cation	***D:*** The version obfuscates the ad library code (e.g., using Proguard (ProGuard (2013))). ***E:*** *"Resolves a ProGuard issue introduced in 9.0.0."*
	Improve logging	***D:*** The version improves the logged information. ***E:*** *"Improved logging when attempting to show an ad that is not ready."*
Managing the displayed content	Manage orientation and layout	***D:*** The version improves the layout of the displayed ads. ***E:*** *"Support for vertical ads and improved ad orientation controls."*
	Display video streaming ad	***D:*** The version adds new features for video streaming or fixes issues that are related to video ads. ***E:*** *"Added support for rewarded video"*
Accessing user data	Fix privacy issues	***D:*** The version fixes users' privacy issues. ***E:*** *"Removed collection of IMEI as per Google Play Content Developer Policy","Removed MAC address tracking."*
	Improve analytics	***D:*** The version improves ads analytics features ***E:*** *"Improvements on analytics."*

Driver category	Driver name	Description (*D*) - Example (*E*)
Compatibility	Support user device models	***D:*** The version fixes issues that are related to the interaction with user' devices. ***E:*** *"Bug preventing MediaPlayer from resuming playback on certain devices."*
	Support Android platforms	***D:*** The version provides support for new Android platforms or fixes issues in the supported Android platforms ***E:*** *"Added support for Android Nougat (Android v7.0)"*
Resource optimization	Optimize network resources	***D:*** The version improves the communication with ad networks. ***E:*** *"Replaced usage of NSURLConnection with NSURLSession for optimizing ad server communication protocols."*
	Optimize memory resources	***D:*** The version improves the memory management (e.g., caching mechanism) of ad libraries. ***E:*** *"Fix Memory leak caused by LocalBroadcastReceiver holding onto MediaView reference."*
	Optimize energy resources	***D:*** The version improves the energy consumption (e.g., battery drain) of ad libraries. ***E:*** *"Improvements to reduce battery drain."*
	Optimize device storage resources	***D:*** The version fixes issues that are related to using device storage. ***E:*** *"Fixed storage overuse issue reported by a small number of publishers upgrading from 2.x ->3.x."*
General features	Integrate with other ad networks	***D:*** The version supports the communication with other ad networks. ***E:*** *"Added and updated mediated network versions."*
	Unspecified	***D:*** The release note does not contain detailed information about the fixed issues or the added features. ***E:*** *"bug fixes"*

4.3.2 RQ2: What drives ad library developers to release a new version?

Motivation: In Section 4.3.1, we observed that ad libraries release new versions quite frequently. In this section, we conduct a qualitative study to identify what drives ad library developers to release a new version. Knowing such drivers can help ad library developers understand the challenges of evolving ad libraries.

Approach: To identify what drives ad library developers to release a new version, we conducted a manual analysis of the release notes of the studied ad libraries as follows.

Step 1: Two researchers (including myself and a collaborator) (as two *coders*) independently followed an iterative approach that is similar to the open coding method (Khandkar (2009)). Each researcher manually read the release notes of every ad library version and identified the drivers for releasing this version. For example, a version with the release notes *"Improved logging when attempting to show an ad that is not ready"* has a driver *Improve logging*. We identified multiple drivers for a version where applicable. When a new driver is identified during the analysis of the release notes, it is added to the list of drivers, and all release notes were reanalyzed using the new list of identified drivers. During this process, we conducted 15 revisits of all release notes to identify all drivers. This process terminated when all versions were analyzed and the list of the identified drivers was finalized (i.e., the researchers did not find any new drivers).

Step 2: For every studied release note, we compared the two lists of identified drivers. Conflicts were discussed until the coders agreed on the identified drivers. We also calculated the agreement between both coders using Cohen's Kappa inter-rater agreement (Cohen (1960)). Cohen's Kappa value ranges from -1 to +1. A Cohen's Kappa value of +1 means that both coders identified the same drivers for all analyzed releases. To

calculate the Cohen's Kappa value, we used the "psych" (CRAN (2017)) library in R. In our study, the Cohen's Kappa value is 0.83 which is an almost perfect agreement according to the interpretation of the Cohen's Kappa value which is proposed by Landis and Koch (1977). At the end of this step, we identified 16 drivers for releasing ad library versions. Table 4.4 shows the list of the identified drivers along with the description and an example of each driver.

Table 4.5: Statistics for the identified drivers for releasing an ad library version (grouped by the driver category).

Driver category	**Driver name**	**# of ad libraries**	**# of versions**
Internal code fixing	Fix a crash or exception	6	38
	API refactoring	6	33
	Improve code obfuscation	4	10
	Improve logging	4	6
Managing the displayed content	Display video streaming ad	8	39
	Manage orientation and layout	7	23
Accessing user data	Improve analytics	6	16
	Fix privacy issues	5	13
Resource optimization	Optimize memory resources	6	24
	Optimize network resources	3	7
	Optimize device storage resources	3	3
	Optimize energy resources	1	2
Compatibility	Support Android platforms	8	23
	Support user device models	5	8
General features	Integrate with other ad networks	4	5

Findings: **Adding and improving the streaming video ad functionality is the most occurring driver for releasing an ad library version.** Table 4.5 shows that we found 39

versions that add or improve the video streaming ad functionality. All studied ad libraries release at least one version that improved the displayed video ads. By carefully analyzing the versions that concern the video streaming functionality, we identified the following four main features that are related to video streaming ads: (1) offering reward videos, (2) adding video controls, (3) handling native video ads, and (4) prefetching video ads. Table 4.6 shows the description and an example of the identified video streaming features.

Table 4.6: The main identified features for video streaming ads.

Category	**Description (*D*) - Example (*E*)**
Offering reward videos	***D:*** Ad library developers implement or improve reward videos features that give users rewards (e.g., points in game apps) after watching a video ad. ***E:*** *"Rewarded video support from the MoPub Marketplace (Beta)."*
Adding video controls	***D:*** Ad library developers allow users to control (e.g., pause, resume, mute, and replay) the displayed video ads. ***E:*** *"Added new design for play/pause button in Rewarded Video."*
Handling native video ads	***D:*** Ad library developers improve or add features that display video ads that have the same look and feel as the integrating app. ***E:*** *"Added the setAdChoicesPlacement() method to the NativeAdOptions.Builder class, which app publishers can now use to specify the location of their AdChoices in native ads."*
Prefetching video ads	***D:*** Ad library developers implement or improve prefetching techniques to obtain video ads from ad networks and store these ads into a user's device to be displayed later. ***E:*** *"Video cache limit updated to 64mb for prefetching."*

We observe that ad library developers were constantly improving the video streaming features of their libraries. For example, the Facebook Audience Network ad library started supporting reward videos (*"Added new design for play/pause button in*

Rewarded Video") in June 2017 (Facebook (2017)). The reason for the large number of versions that improve the video streaming ad functionality is probably that video ads lead to a better user engagement than static ad images (Belanche et al. (2017)), and are therefore a popular feature amongst ad publishers.

Ad library developers tend to provide support for the latest version of the Android platform. The Android platform is updated approximately every six months (Android version history (2019)). Because most new Android versions offer new features, app developers are keen to migrate to the latest version of Android to make use of these new features (McDonnell et al. (2013)). As a result, ad library developers need to keep up and make sure that their libraries support new versions of Android as well.

To understand how fast ad libraries add support for the new version of the Android platform, we calculated the difference between the release date of a new Android version and the release date of the ad library version that implements support for the latest Android version. We find that the median number of days required to add support for the newer Android platform is less than two months (49 days). Interestingly, we also observe that over time many ad libraries lower their minimum supported Android version to support older Android versions (e.g., *"Lowered our library's minimum SDK version to fix build issues with apps that support earlier versions."*).

In addition to supporting new Android versions, ad library developers had to perform maintenance on their libraries as well to ensure that they work properly in the supported Android versions. For example, in 10 out of 23 release notes that mention the Android platform, the ad library developers mention that they fix issues and bugs in the supported Android versions (e.g., *"Fixed a crash if the app starts when a WebView update is in progress for Android 5.0"*). If ad library developers do not update

their libraries to support the newer Android version, their ad libraries still can work on the devices with the newer versions because of backward compatibility of Android versions. While updating their ad libraries to add the updated features of the newer Android versions, ad library developers need to be careful that their ad libraries do not break because of the added features.

Ad libraries leverage users' information to provide analytics features for integrating apps. As shown in Table 4.5, 75% of the studied ad libraries mention ad analytics in their release notes. We observed that ad libraries collect user data (e.g., a user's location) to offer two main features: (1) to provide metrics (e.g., the number of clicked ads) about the performance of the displayed ads, and (2) to select the most suitable ads for an app user. Table 4.7 shows the analytics features that were added or improved during the studied 18 months by the studied ad libraries, along with the collected user data.

As shown in Table 4.7, ad analytics offer insights about the displayed ads. The collected ad analytics metrics are useful for developers of integrating apps who wish to increase their ads' revenue. For example, the analytics provide information about which ads, or which screen positions are the most successful in terms of user engagement. Prior work showed that users mainly complain about the size and the location of the displayed ads (Gui et al. (2017)). By leveraging the ad analytics insights, integrators can improve the frequency, size, and location of the displayed ads.

Ad library developers stop collecting user information to adhere to policies and Google's best practice guidelines. As shown in Table 4.7, ad libraries collect user data to tailor the displayed ads to the app user. For example, ad libraries leverage the user's location to display ads for nearby stores. However, some user data may reveal too much

about the user's identity. For example, the IMEI and the MAC address of the user's device can be used to identify the physical user device, and therefore Google's best practice guidelines discourage developers from collecting this data (Google (2019b)). We observed during the study period that the MoPub, Vungle, and Flurry ad libraries reduced the amount of user information that they collect.

An additional privacy-related concern for ad library developers is the General Data Protection Regulation (GDPR), which is a privacy-related law for individuals in the European Union. While the GDPR was adopted after our study period, we observed through manual inspection that ad library developers released versions to adhere to the GDPR law. For example, the Google AdMob library added a form that requests a user's consent for sharing user information with the ad network.

Memory leaks are the most resolved resource handling-related issues in the studied ad library versions. We observe that fixes for memory leak issues were mentioned in 8 out of 32 release notes that discuss resource handling. Since ad libraries continuously fetch ad contents, failure to release the collected contents after displaying them can cause a memory leak. A memory leak can cause a crash, but also an energy issue as it may lead to unnecessary garbage collection calls (Guo et al. (2013)). Therefore, ad library developers should follow the proper guidelines (Google (2019b)) to avoid memory leaks.

Table 4.7: The main identified features that are offered in ad analytics.

Analytics main features	**Collected Data**	**Description (*D*) - Example (*E*)**
Provide metrics about the displayed ads	Ad session data	***D:*** Collect the duration of user engagement with an ad (i.e., how long users watch a video ad before closing it). ***E:*** MoPub has a `ExternalViewabilitySessionManager` class that provides methods (e.g., `createVideoSession`, `recordVideoEvent`) to capture the session information.
	Ad revenue	***D:*** Collect metrics (e.g., ad viewability ratio (Google (2019c)), click through ratio (AdEspresso (2018))) that are useful for estimating ad revenue.
	metrics	***E:*** *"Support for Moat 3rd party video viewability."* The viewability metric captures how many of the displayed ads are actually viewed by a user.
Select the most suitable ads for an app user	User identifier	***D:*** Collect the unique user identifier to tailor ads to a user. The Google Play Store provides a unique identifier for every app user (the advertising ID) that is useful for fine-tuning the displayed ads for every user. ***E:*** *Unity Ads collect advertising id with the method named fetchAdvertisingId*
	Device information	***D:*** Collect data related to a user's device information (e.g., device model) to improve the displayed ads. ***E:*** *"Reporting more device stats to serve better and better ads."*
	Demographic data	***D:*** Collect a user's demographic data (e.g., language, country, and location information) to display ads that are suitable for that particular demographic. ***E:*** *"Added auto-population of location information for apps that explicitly grant the location permission."*

Summary of RQ2

Ad library developers are constantly updating their ad libraries to support the latest version of the Android platform and even to support older versions of the platform enabling their libraries to work on as many devices as possible. The most occurring driver for releasing an ad library version is to add or improve the video streaming ad feature. Memory leaks are the most resolved resource handling-related issue in ad library versions.

4.3.3 RQ3: How did the architecture of ad libraries evolve over time?

Motivation: To capture the evolution of an ad library at the architectural level, we first need to derive a reference architecture for ad libraries. A reference architecture for a domain captures the fundamental components and their relationships that are present in existing systems in the domain (Hassan et al. (2017); Grosskurth and Godfrey (2005)). Identifying a reference architecture for ad libraries does not only help understand the system, but also can serve as a template for creating a new or evolving an existing ad library by reusing components at the design and implementation level (Medvidovic and Taylor (2000); Roy et al. (2017); Hassan and Holt (2002)).

Approach: To derive the reference architecture, we followed an approach that is similar to the one proposed by Hassan and Holt (2000). In particular, we used the source code and API documentation of the ad libraries to derive the reference architecture as follows.

Step 1: Generating a conceptual architecture of each ad library. In this step, we built a conceptual architecture for each ad library based on our domain knowledge and the available documentation for that library.

Table 4.8: The identified components of the reference architecture of ad libraries.

Component name	**Component definition**
Ad Lifecycle Manager	The *Ad Lifecycle Manager* is the main entry point of an ad library. It delegates all steps of an ad's lifecycle to the appropriate components. An ad's lifecycle consists of four steps: (1) fetching the ad, (2) storing the ad, (3) displaying the ad, and (4) deleting the ad. The *Ad Lifecycle Manager* performs its functionality using two subcomponents (*Interactive Ad* and *Non-interactive Ad*). The *Interactive Ad* component displays an interactive ad format (e.g., Augmented Reality ad or Playable ad) on a user's device. This ad format increases the engagement of users with the displayed ads (Applift (2017); Martin (2017)). For example, the Playable ads promote other Android apps of the Google Play Store, and users can play a demo of the promoted app (within the displayed ads) without downloading the app. In the *Non-interactive Ad*, users are allowed only to control (e.g., play or close) the displayed ads. This component consists of the following four subcomponents: (1) *Banner*, (2) *Native*, (3) *Interstitial* (full-screen), and (4) *Rewarded Video*.
Ad Retrieval	The *Ad Retrieval* component provides functionality to communicate with ad networks and download ad data. In particular, the *Ad Lifecycle Manager* sends messages to the *Ad Retrieval* component for fetching ads from an ad network. Then, the *Ad Retrieval* component fetches ad data (lightweight format such as JSON) from the ad networks by using HTTP requests.
Ad Serving	The main functionality of the *Ad Serving* component is to construct displayable ad content (e.g., image or video content) by processing the fetched ad data (e.g., JSON objects).
Analytics	The *Analytics* component collects information about users (e.g., their location) and leverages the collected data to: (1) select the most suitable ads for every user, and (2) provide statistics about the displayed ads to integrators. This component sends the collected information of a user to the *Ad Lifecycle Manager* for fetching and displaying appropriate ads for every user.
Util	The *Util* component provides functionality related to cache management, handles all configuration setup and provides logging functionality with different log levels (e.g., debug, warning and error log levels). This component contains three subcomponents: (1) *Cache Manager*, (2) *Logger*, and (3) *Configuration/Properties*.

Step 2: Generating a concrete architecture of each ad library. In this step, we used the Understand tool (Understand (2015)) to generate and visualize the dependency call graph of each version of each studied ad library. Then, we analyzed the packages and classes of each ad library in the call graph. We identified the packages that offer similar functionalities and grouped these packages into a single architectural component. For example, in the Unity ad library, we observed that the request package (com.unity3d.ads. request), the broadcast package (com.unity3d.ads.broadcast), and the connectivity package (com .unity3d.ads.connectivity) perform a similar functionality of communicating with the ad networks through the HTTP protocol. Therefore, we grouped these packages into one architectural component which we named *Ad Network Connectivity*. At the end of this step, we identified the concrete architecture of the studied ad libraries.

Step 3: Refining the conceptual architectures of each ad library. We analyzed the concrete architecture and refined the conceptual architecture of every ad library.

Step 4: Deriving the reference architecture of the ad libraries. We derived a reference architecture that is based on the commonalities between the refined conceptual architectures of the studied ad libraries as proposed by Hassan and Holt (Hassan and Holt (2000)). Figure 4.4 shows our proposed reference architecture, and Table 4.8 gives a short description of each of the components of the architecture.

To study the architectural changes among ad libraries, we compared the conceptual architecture of every studied ad library with our derived reference architecture. In addition, we studied the differences between the architecture of the studied ad libraries, and we analyzed the architectural evolution of every ad library during our study period.

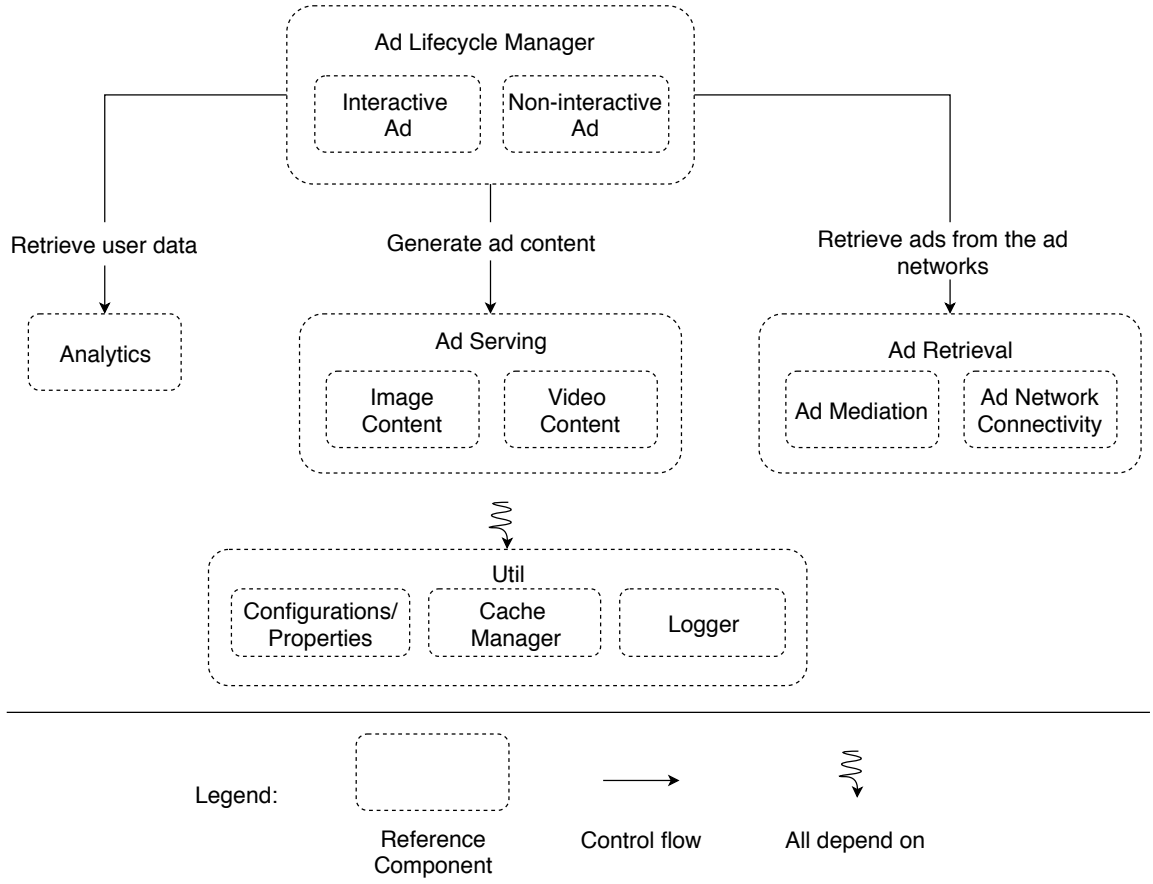

Figure 4.4: Ad library reference architecture. A line between two components indicates that there is a relationship between the components.

Findings: **7 out of 8 ad libraries offer an ad mediation component that enables integrators to communicate with several ad networks through a unified interface.** Ruiz et al. (2014) showed that integrators often integrate more than one ad library to increase their potential revenue. Hence, to display ads from different ad networks, integrators need to write code to interact with several ad libraries which increases their app maintenance effort.

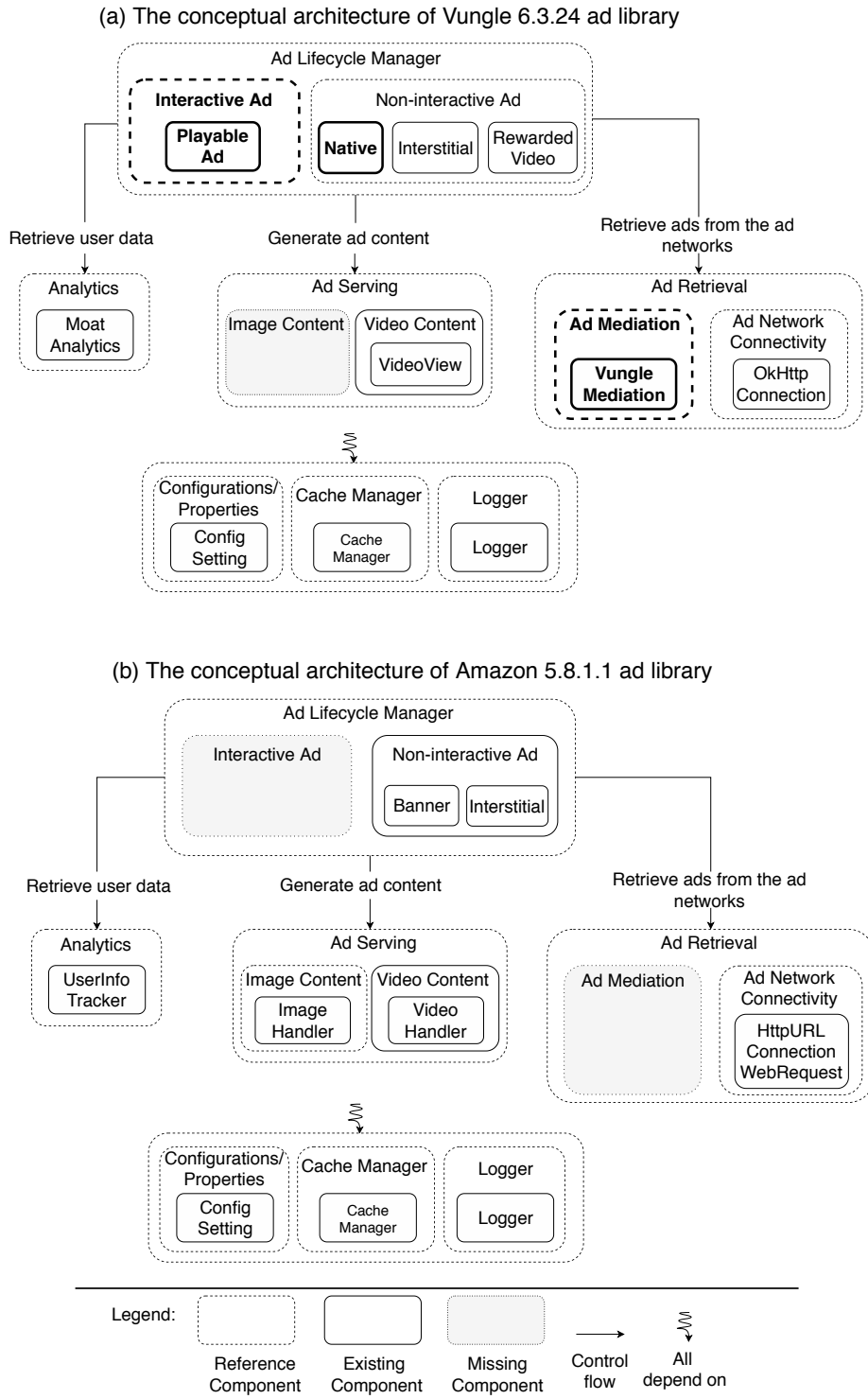

Figure 4.5: The differences between the ad library architectures of the Vungle and Amazon ad libraries. A bold box with a bold font shows a new component that appears in the version of the ad library.

To reduce the needed effort for serving ads from multiple ad networks, ad libraries nowadays commonly offer an ad mediation component. This component allows integrators to serve ads from several ad networks using a unified interface.

While the ad mediation component reduces the required effort to serve ads from multiple ad networks, integrators need to include all the dependent libraries of these ad networks into their apps. Hence, the overall app size of the integrating app increases considerably. To reduce the app size while using ad mediation, Google AdMob offers an *SDK-less* mediation feature (Google (2016)), for which integrators do not need to include the dependent libraries of other ad networks. Google AdMob's SDK-less mediation feature automatically communicates with the supported ad networks through Google's ad network servers. Therefore, the app size remains small (Google (2016)). Hence, we recommend that other ad library developers provide solutions to reduce the size of the integrating apps as well, e.g., by handling communications with the other ad networks on the ad network server.

Ad libraries have several interactive and non-interactive subcomponents as they provide different ad media formats. We observe that the non-interactive component supports four main formats of ads: (1) *banner* ads, (2) *native* ads, (3) *interstitial* ads, and (4) *rewarded video* ads. Note that image-based ads can be of the *banner*, *native* or *interstitial* format, and that video-based ads can be of the *native* or *interstitial* format. A *rewarded video* is a special format of a video ad.

On the other hand, the interactive component supports two main formats of ads: (1) *Playable* ads and (2) *Augmented Reality* ads. Unlike non-interactive ads, playable ads and augmented reality ads increase user-engagement by enabling users to directly interact with the displayed ads (Applift (2017); Martin (2017)).

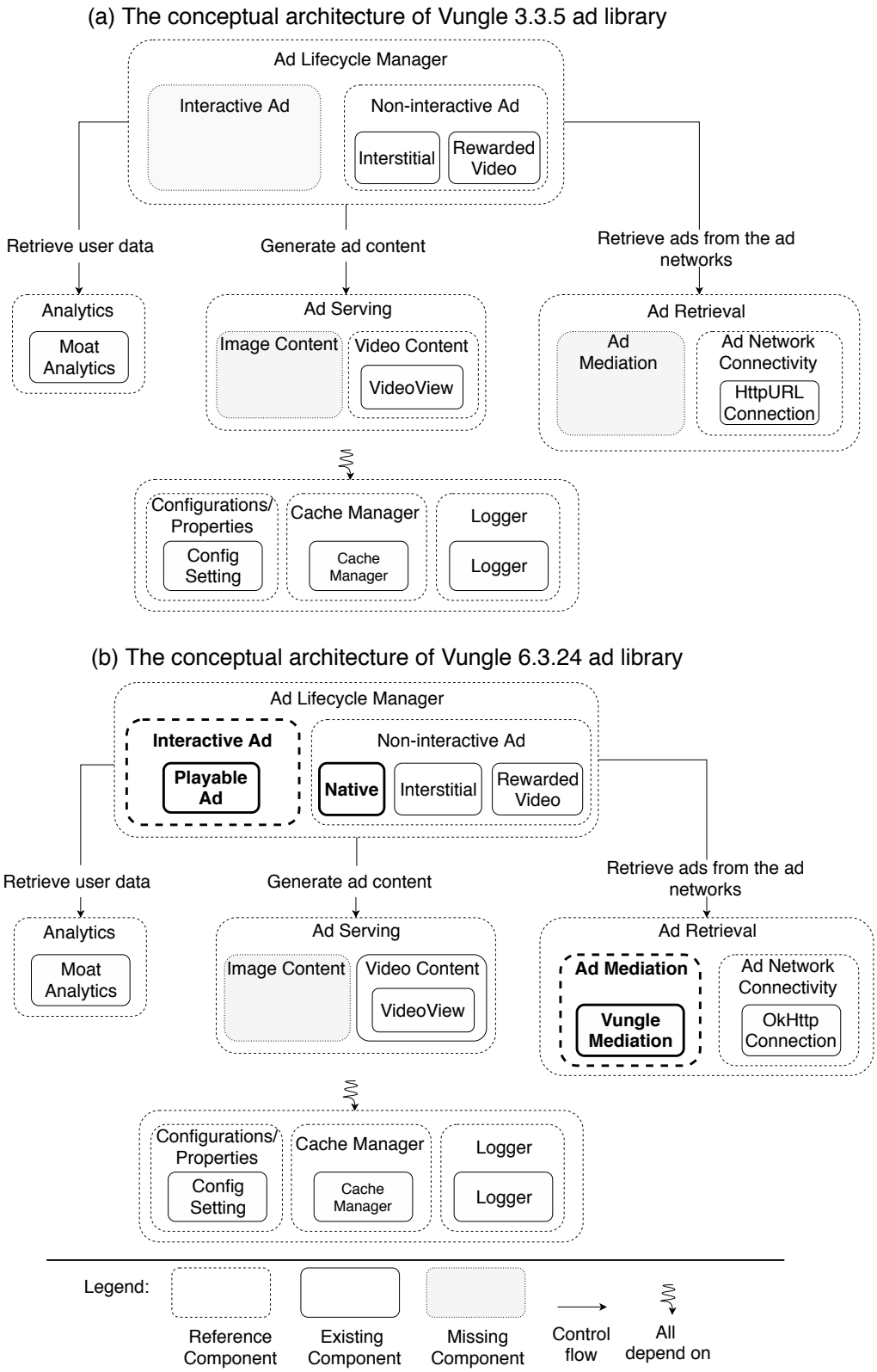

Figure 4.6: Evolution of the architecture of the Vungle ad library during our study period. A bold box with a bold font shows a new component that appears in the version of the ad library.

We can use these components to identify the differences between ad libraries in terms of their supported ad formats. For example, Figure 4.5 shows the architecture of the Vungle and the Amazon Mobile Ad library. In Figure 4.5, we highlight all components that exist in the reference architecture but not in the particular ad library. As shown in Figure 4.5, the Vungle ad library supports both interactive and non-interactive ads whereas the Amazon mobile Ad library only supports non-interactive ads. In addition, we observe that the Vungle ad library focuses only on video ads (e.g., native video, full-screen and rewarded video ads) (AppBrain (2016); Vungle (2019)). Figure 4.5 also shows that the Amazon Mobile Ad library focuses on image and video ad contents that are provided through banner and interstitial ads.

All ad libraries support the automatic resizing of the displayed ads based on the device model of a user. In our architectural analysis, we observe that all ad libraries leverage the user's device model information (e.g., the screen size and resolution) to automatically adjust the layout of the displayed ads. This information is collected by the Ad Lifecycle Manager, which also manages the layout of the displayed ads. In addition, the Google AdMob, Facebook Audience Network, and Amazon Mobile Ad ad libraries allow integrators to adjust the size of the displayed ads. Prior research showed that full-screen and frequently displayed ads can lead to negative reviews of the integrating apps (Gui et al. (2017)). Hence, integrators and ad library developers should be careful while adjusting the size and the frequency of the displayed ads.

Ad library developers are actively evolving the architecture of their libraries. For example, Figure 4.6 shows the difference between the conceptual architecture of the Vungle ad library at the start and at the end of our study period. We find that the Vungle added the following new components to its architecture:

1. **Adding ads mediation component.** The Vungle ad library added ad mediation support for eight ad networks (e.g., MoPub and Google AdMob).

2. **Adding native ads component.** The developers of the Vungle ad library added support for native video ads. Such ads are rendered and displayed with the same look and feel as the integrating apps. As native ads are less obtrusive to users, they can improve the clickthrough rate of the displayed ads (Manic (2015)).

3. **Adding interactive ad component.** The Vungle ad library added the *Playable Ad* as an interactive ad component. The playable ads are displayed on a device to promote other Android Apps. In particular, these ads allow users to play a demo version of the promoted apps (within the displayed ads) without installing the apps on the user's device. Such interaction with the ads improves user-engagement (Applift (2017)).

We observed that the other studied ad libraries were all converging towards our reference architecture during the study period. This convergence suggests that our derived reference architecture is capturing shared aspects across the domain.

Summary of RQ3

We propose a reference architecture for ad libraries. During our study period, ad library developers actively evolved the architecture of their libraries, as they added new functionality to the libraries. All ad libraries appear to be slowly converging to offer similar features with their architectures mapping well to our derived reference architecture.

4.4 An Exploration of Quality Attributes of the Derived Reference Architecture of Ad Libraries

An ad library architecture needs to satisfy several quality attributes to overcome common challenges (e.g., continuous improvements of the offered ad formats) across ad libraries. We briefly discuss below some quality attributes and how our derived reference architecture satisfies.

4.4.1 Evolvability of the Supported Ad Formats

Developers of ad libraries are always in search of ways to evolve their displayed ad formats to make them interesting and intricate in an effort to attract users into interacting with such ads. In particular, we observe that 31% of the studied library versions note improvements in the display of ads.

For instance, we observed that ad library developers have recently introduced more interactive ad formats (e.g., Playable ad and Augmented Reality ad) to improve user-engagement with the displayed ads. Our derived reference architecture enables such an evolutionary pattern where the other components of the reference architecture remain relatively stable over the years, with the internals of the *Interactive Ad* and *Non-interactive Ad* components exhibiting a large amount of changes while the impact of such changes being localized to the *Ad Lifecycle Manager* component. In the future, we envision ad libraries supporting voice-over ads (e.g., users can listen to streaming audio ads and interact with an ad using their voice commands); our current reference architecture would support such ads as well.

4.4.2 Flexibility of Integrating Multiple Ad Libraries

App developers usually prefer to integrate more than one ad library to ensure that they can always display an ad (since ad libraries do not guarantee that they would always provide an ad to display when an ad is requested by an app) (Ruiz et al. (2014)). To facilitate the ease of integrating multiple ad libraries, our reference architecture offers the *Ad Mediation* component. This component allows integrators to serve ads from different ad networks through a unified interface.

For example, an app that integrates only Google AdMob to serve ads can easily serve ads from other ad networks (e.g., Facebook and MoPub) that are supported by Google AdMob through the *Ad Mediation* component of the Google AdMob library. Therefore, it is possible to retrieve ads from the other ad networks with the Google AdMob library.

4.4.3 Efficiency of Ads at Run-time

Ads need to provide an interactive user experience while minimizing their resource consumption. The *Cache Manager* component in the reference architecture pre-fetches ads, which improves the responsiveness of ads. However, this caching mechanism brings into play its own slew of challenges, as we observe that eight versions of the studied ad libraries mention memory leak issues.

4.4.4 Minimizing the Size of the Ad Library

The size of an app is negatively associated with the number of installations of an app (Reinhardt (2016)). Hence, app developers aim to keep the size of their apps as small as possible. In return, we observe the same phenomena being reflected at the architecture level for the ad libraries that we studied. Over the years, we note that several instances of ad libraries replaced components of their reference architecture with external components (either ones provided by the Android platform, or by other companies) instead of including all the components. For example, we observed that Google gutted the Analytics component, replacing it with a thin façade that interfaces directly with the Google Analytics that is already offered by the Android platform. While on the other hand, we observed that Vungle removed the same component from its code base and points developers to where to download that component, enabling developers to avoid including the same components twice if they need its functionality for their own app.

Furthermore, we note that the developers of some ad libraries have worked on modularizing the different ad formats so integrator apps would only include the required modules. For example, if an integrating app displays only banner ads, the app only has to integrate the banner ads module. According to the MoPub website (MoPub (2016)), their modular architecture allows app developers to save up to 60% of the library size by including only the needed modules.

4.5 Implications

In this section, we describe the implications of our longitudinal analysis of ad libraries for ad library developers and ad library integrators.

4.5.1 Implications for Ad Library Developers

To reduce the size of an ad library, ad library developers should offer a modular version of their libraries. The size of an app is negatively associated with the number of installations of an app (Reinhardt (2016)). Hence, it is necessary to keep app size (and therefore the size of all its libraries) as small as possible. As described in Section 4.3.1, refactoring the MoPub architecture into a more modular one reduced the library size by 60%. Hence, we recommend that ad library developers rethink their designs and offer a modular version of their libraries to integrators who do not require the full functionality of the library. One additional practice that can reduce the size of ad libraries is to offer an SDK-less mediation feature. As described in Section 4.3.3, the SDK-less mediation feature of Google AdMob allows integrators to communicate with several ad networks without integrating the dependent ad libraries of these ad networks. Hence, we recommend that ad library developers offer an SDK-less mediation feature in their libraries.

Ad library developers should be careful about memory leaks in their libraries. In section 4.3.2, we observed that memory leaks are the most often fixed resource-related issue. Because ad libraries continuously fetch and display ads, these libraries are vulnerable to memory leaks when the ads are not correctly released. As memory leaks may cause performance and energy issues for app users (Guo et al. (2013)), ad library developers should be extra careful and follow proper guidelines (Google (2019b)) to avoid memory leaks. For example, ad library developers should leverage existing memory profiling tools, such as Memory Profiler (Google (2019)) to identify memory leaks.

4.5.2 Implications for Ad Library Integrators

Integrators should be aware that there is a median delay of 49 days after a new version of the Android platform before an ad library supports that version. In Section 4.3.2, we show that ad libraries have a release interval of approximately one month. However, providing support for a new Android version takes a median of 49 days. Hence, integrators should be cautious when they support a new version of Android before the ad library supports that version, as it is possible that their app will not be able to display ads correctly during that period leading to lost revenue.

4.6 Threats to Validity

External Validity: In our study, we analyzed the evolution of eight of the most popular ad libraries. Future studies should investigate whether our findings hold for other ad libraries. In addition, we focused our study on ad libraries that are integrated in free-to-download Android apps. Future studies should broaden the scope of our study and investigate how our findings apply to ad libraries that are integrated in other types of apps, such as paid or iOS apps.

Internal Validity: In our study of what drives ad library developers to release a new version of an ad library, we manually analyzed the release notes and identified the drivers in every release note. As we are not the ad library developers, it is possible that we misinterpreted the drivers for releasing a new version. To mitigate this threat, two researchers (including myself and a collaborator) individually identified the drivers from the release notes and consolidated their result. However, future studies should consider consulting ad library developers to identify the drivers for releasing a new version

of the ad library.

Our assumption that apps that integrate an ad library actually display ads is another threat to validity. However, since the size of an app is a major concern for app developers (as larger apps are less likely to be installed by users) and the main objective of integrating ad libraries is to earn revenue based on the displayed ads, we assume that an app only integrates an ad library if it actually displays ads.

One of the threats to validate in our study is that we only consider the release notes to identify the drivers for releasing a new version of ad libraries. There might be possibility that ad library developers do not document all their changes in the release notes. However, we study the top integrated ad libraries which are popular in the market and they should maintain a good documentation in their release notes so that app developers can understand the newly added features of ad libraries which can inspire them to integrate the ad libraries.

4.7 Chapter Summary

Ad libraries have become an integral part of the mobile app economy. Even though ad libraries play an essential role in the app ecosystem, there has been no prior study on how these ad libraries evolve over time. In this chapter, we conduct a longitudinal analysis of the 8 most popular ad libraries over a period of 33 months (from April 2016 until December 2018). The most important findings of our study are:

1. Ad libraries are continuously evolving and have a median release interval of 34 days.

2. As a large app size is negatively associated with the number of installations of an

app, it is essential that third-party libraries are as small as possible. Ad library developers are reducing their library sizes by: (1) creating a lighter version of the ad library, (2) removing functionality from the ad library, and (3) redesigning the ad library into a more modular architecture.

3. Ad library developers tend to integrate support for new Android platforms. However, there is median delay of 49 days after a new Android version. Therefore, ad library integrators should be cautious when updating their own apps to the latest Android version, as their integrated ad libraries may not yet support that version (leading to lose ad revenue).

4. Memory leaks are the most often resolved resource handling-related issues in the studied ad libraries. Hence, ad library developers should carefully examine memory leak issues in their libraries.

5. We derived a reference architecture for ad libraries. We observed that during our study period, all ad libraries were slowly converging towards this reference architecture.

Our study is useful for developers who wish to build and evolve their own ad libraries. In particular, these developers can leverage our derived reference architecture as a starting point and blueprint for building and evolving their own ad libraries.

CHAPTER 5

Conclusions and Future Work

THIS chapter summarizes our work and presents potential opportunities for further work.

Ad libraries have become an integral part in mobile economy. App developers integrate one or multiple ad libraries into their apps to display ads and earn revenue based on the displayed ads and user interactions with such ads. Even though ad libraries play an essential role in mobile economy, there has been no prior studies of how these ad libraries are integrated by app developers and how these libraries evolve.

In this thesis, we study the integration practices of ad libraries and the evolution of such libraries. Our findings can help ad library developers to understand the challenges of developing ad libraries and possible improvements of their offered features

so that ad library developers can build robust ad libraries to satisfy the need of app developers (e.g., increase ad revenue and improve user engagements).

5.1 Thesis Contributions

The main contributions of this thesis are as follows:

1. **Identifying the strategies for integrating multiple ad libraries.** We manually analyze a statistically representative random sample of 62 apps that display ads and derive a set of rules to automatically identify the strategies for integrating multiple ad libraries. Our study shows that app developers customize the integration process indicating that the current off the shelf features of ad libraries for integrating multiple ad libraries are not satisfying their needs. Ad library developers can leverage our study to systemically ensure that their ad libraries can support the varying needs of the apps.

2. **Conducting an in-depth analysis of the main drivers for ad library developers to release a new version.** We manually analyze the release notes of the eight most popular ad libraries and identify the main drivers for releasing a new ad library version. Our study shows that ad library developers are constantly updating their ad libraries to support the latest Android version and even to support older version of Android so that their library can work on as many device as possible. The main occurring driver for releasing a new version is to add or improve video streaming ad features. Software developers who wish to build their own ad libraries can leverage our study to understand the challenges of developing ad libraries.

3. **Deriving a reference architecture for ad libraries.** We are the first to derive a reference architecture for ad libraries. We also study the change of the architecture of the studied ad libraries over time. Our study shows that as ad libraries evolve, their architectures tend to converge towards our derived reference architecture. Our derived reference architecture and the analysis of the architecture of the studied ad libraries can guide developers to build their own ad libraries.

5.2 Future Work

In this section, we explore potential opportunities for improving our work.

- We perform a static source code analysis and derive a set of rules to identify the strategies for integrating multiple ad libraries. Such static analysis may not cover all possible behavior of the apps interacting with ad libraries. Future research can perform a dynamic analysis of apps to get more insights about the ad library integration practices.

- To analyze the evolution of the architecture of the studied ad libraries, we manually derive the architecture for each version of an ad library and manually identify the changes in the architecture of that ad library. Such manual study is difficult and time consuming. Therefore, future research can develop an automatic approach that can identify the changes in the architecture of ad libraries and visualize these changes.

- In this thesis, we study ad libraries focusing on the ad library developers point of view and document a few implications for them. Future research can also focus

on the point of app developers who integrate ad libraries for displaying ads (e.g., how our identified integration strategies can help app developers).

Bibliography

Addo, I. D., Ahamed, S. I., Yau, S. S., and Buduru, A. (2014). A reference architecture for improving security and privacy in internet of things applications. In *2014 IEEE International Conference on Mobile Services*, pages 108–115.

AdEspresso (2018). The Simple Guide to Understand Facebook Ads Metrics. https://adespresso.com/blog/understand-facebook-ads-metrics-guide/. (Last accessed: July 2019).

Akdeniz (2013). Google Play Crawler. https://github.com/Akdeniz/google-play-crawler. (Last accessed: July 2019).

Analytics, U. A. (2019). Urban Airship Analytics. https://www.airship.com/platform/analytics-data/performance-analytics/. (Last accessed: July 2019).

Android version history (2019). Android version history. https://en.wikipedia.org/wiki/Android_version_history. (Last accessed: July 2019).

Apache BCEL (2018). Download Apache Commons BCEL. https://archive.apache.org/dist/commons/bcel/. (Last accessed July 2019).

AppAnnie (2017). In-App Advertising Spend to Triple, Reach $201 Billion by 2021. https://www.appannie.com/en/insights/market-data/app-advertising-spend-2021/. (Last accessed: July 2019).

AppAnnie (2018). App Annie. https://www.appannie.com/. (Last accessed July 2019).

AppBrain (2016). Video ads. https://www.appbrain.com/stats/libraries/tag/video-ads/video-ads. (Last accessed: July 2019).

AppBrain (2019). AppBrain Intelligence. https://www.appbrain.com/stats/. (Last accessed: July 2019).

Applift (2017). Why Playable Ads are the Key to Engaged Users. https://applift.com/blog/playable-ads-2. (Last accessed: July 2019).

AppsFlyer (2019). AppsFlyer. https://www.flurry.com/. (Last accessed: July 2019).

Au, K. W. Y., Zhou, Y. F., Huang, Z., and Lie, D. (2012). PScout: Analyzing the Android Permission Specification. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 217–228.

Backes, M., Bugiel, S., Derr, E., McDaniel, P., Octeau, D., and Weisgerber, S. (2016). On Demystifying the Android Application Framework: Re-visiting Android Permission

Specification Analysis. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, pages 1101–1118.

Belanche, D., Flavián, C., and Pérez-Rueda, A. (2017). Understanding interactive online advertising: Congruence and product involvement in highly and lowly arousing, skippable video ads. *Journal of Interactive Marketing*, 37:75–88.

Book, T., Pridgen, A., and Wallach, D. S. (2013). Longitudinal Analysis of Android Ad Library Permissions. *Computing Research Repository*, abs/1303.0857.

BusinessOfApps (2017). Video ads bring in 31% of app revenue, while rewarded video ads top user experience approval charts. http://www.businessofapps.com/news/video-ads-bring-in-31-of-app-revenue-while-rewarded-video-ads-\top-user-experience-approval-charts/. (Last accessed July 2019).

Calciati, P. and Gorla, A. (2017). How Do Apps Evolve in Their Permission Requests?: A Preliminary Study. In *Proceedings of the 14th International Conference on Mining Software Repositories*, MSR '17, pages 37–41.

Calciati, P., Kuznetsov, K., Bai, X., and Gorla, A. (2018). What Did Really Change with the New Release of the App? In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, pages 142–152.

CFR (2013). CFR. http://www.benf.org/other/cfr/. (Last accessed July 2019).

Cohen, J. (1960). A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20(1):37–46.

CRAN (2017). psych: Procedures for Psychological, Psychometric, and Personality Research. https://cran.r-project.org/web/packages/psych/index.html. (Last accessed: July 2019).

Davidson, D., Fredrikson, M., and Livshits, B. (2014). MoRePriv: Mobile OS Support for Application Personalization and Privacy. In *Proceedings of the 30th Annual Computer Security Applications Conference*, ACSAC '14, pages 236–245.

De La Iglesia, J. L. M. and Gayo, J. E. L. (2009). Doing business by selling free services. In *Web 2.0*, pages 1–14.

Derr, E., Bugiel, S., Fahl, S., Acar, Y., and Backes, M. (2017). Keep Me Updated: An Empirical Study of Third-Party Library Updatability on Android. In *Proceedings of the 24th ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 2187–2200.

dex2jar (2016). dex2jar. http://sourceforge.net/projects/dex2jar/. (Last accessed July 2019).

Dong, F., Wang, H., Li, L., Guo, Y., Bissyandé, T. F., Liu, T., Xu, G., and Klein, J. (2018). FraudDroid: Automated Ad Fraud Detection for Android Apps. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE'18, pages 257–268.

Dueñas, J. C., Oliveira, W. L. d., and Puente, J. A. d. l. (1998). A Software Architecture Evaluation Model. In *Proceedings of the Second International ESPRIT ARES Workshop on Development and Evolution of Software Architectures for Product Families*, pages 148–157.

Facebook (2017). Introducing Rewarded Video for Game Developers. https://www.facebook.com/audiencenetwork/news-and-insights/introducing-rewarded-video. (Last accessed: July 2019).

Felt, A. P., Chin, E., Hanna, S., Song, D., and Wagner, D. (2011). Android permissions demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 627–638.

Flurry (2019). Flurry Analytics. https://developer.yahoo.com/flurry/docs/analytics/. (Last accessed: July 2019).

Gao, C., Zeng, J., Sarro, F., Lyu, M. R., and King, I. (2018). Exploring the effects of ad schemes on the performance cost of mobile phones. In *Proceedings of the 1st International Workshop on Advances in Mobile App Analysis*, A-mobile '18, pages 13–18.

Google (2016). SDK-less Mediation: An easier way to mediate. https://www.blog.google/products/admob/sdk-less-mediation/. (Last accessed: July 2019).

Google (2018). Playtime 2018: Helping you build better apps in a smaller bundle. https://android-developers.googleblog.com/2018/10/playtime-2018.html. (Last accessed: July 2019).

Google (2019a). App Manifest Overview. https://developer.android.com/guide/topics/manifest/manifest-intro. (Last accessed: July 2019).

Google (2019b). Best practices for unique identifiers. https://developer.android.com/training/articles/user-data-ids. (Last accessed: July 2019).

Google (2019). com.google.android.gms.ads.identifier. https://developers.

google . com / android / reference / com / google / android / gms / ads / identifier/package-summary. (Last accessed July 2019).

Google (2019a). Google Mobile Ads Lite SDK. https://developers.google.com/admob/android/lite-sdk. (Last accessed: July 2019).

Google (2019b). Manage your app's memory. https://developer.android.com/topic/performance/memory. (Last accessed: July 2019).

Google (2019c). Measuring Ad Viewability. https://www.thinkwithgoogle.com/feature/viewability/. (Last accessed: July 2019).

Google (2019d). Mobile App Reporting in Google Analytics - Android. https://developers.google.com/analytics/devguides/collection/android/v4/. (Last accessed: July 2019).

Google (2019). Usage of Android Advertising ID. https://play.google.com/about/monetization-ads/ads/ad-id/. (Last accessed July 2019).

Google (2019). View the Java heap and memory allocations with Memory Profiler. https://developer.android.com/topic/performance/memory. (Last accessed: July 2019).

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*, ACSAC '14, pages 101–112.

Grosskurth, A. and Godfrey, M. W. (2005). A Reference Architecture for Web Browsers. In *Proceedings of the 21st International Conference on Software Maintenance*, ICSM '05, pages 661–664.

Gui, J., Mcilroy, S., Nagappan, M., and Halfond, W. G. J. (2015). Truth in Advertising: The Hidden Cost of Mobile Ads for Software Developers. In *Proceedings of the 37th International Conference on Software Engineering*, ICSE '15, pages 100–110.

Gui, J., Nagappan, M., and Halfond, W. G. (2017). What Aspects of Mobile Ads Do Users Care About? An Empirical Study of Mobile In-app Ad Reviews. *arXiv preprint arXiv:1702.07681*.

Guo, C., Zhang, J., Yan, J., Zhang, Z., and Zhang, Y. (2013). Characterizing and Detecting Resource Leaks in Android Applications. In *Proceedings of the 28th International Conference on Automated Software Engineering*, ASE '13, pages 389–398.

Hassan, A. E. and Holt, R. C. (2000). A Reference Architecture for Web Servers. In *Proceedings of the 7th Working Conference on Reverse Engineering*, WCRE '10, pages 150–160.

Hassan, A. E. and Holt, R. C. (2002). Architecture Recovery of Web Applications. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 349–359.

Hassan, S., Bezemer, C., and Hassan, A. E. (2018). Studying Bad Updates of Top Free-to-Download Apps in the Google Play Store. *IEEE Transactions on Software Engineering*, pages 1–21.

Hassan, S., Shang, W., and Hassan, A. E. (2017). An Empirical Study of Emergency Updates for Top Android Mobile Apps. *Empirical Software Engineering*, 22(1):505–546.

IronSource (2019). What is eCPM and How Can You Increase It? https://www.ironsrc.com/blog/what-is-ecpm/. (Last accessed: July 2019).

Khandkar, S. H. (2009). Open coding. http://pages.cpsc.ucalgary.ca/~saul/wiki/uploads/CPSC681/open-coding.pdf. (Last accessed July 2019).

Kim, D., Son, S., and Shmatikov, V. (2016). What Mobile Ads Know About Mobile Users. In *Proceedings of the 23rd Annual Network and Distributed System Security Symposium*, NDSS '16, pages 1–14.

Kochava (2019). Kochava. https://support.kochava.com/. (Last accessed: July 2019).

Landis, J. R. and Koch, G. G. (1977). The Measurement of Observer Agreement for Categorical Data. *Biometrics*, 33.

Li, L., Bissyandé, T. F., Klein, J., and Traon, Y. L. (2016). An Investigation into the Use of Common Libraries in Android Apps. In *Proceedings of the 23rd Software Analysis, Evolution, and Reengineering*, SANER '16, pages 403–414.

Life360 (2019). Life360. https://www.life360.com/. (Last accessed: July 2019).

Localitics (2019). Localitics. https://docs.localytics.com/dev/android.html#android. (Last accessed: July 2019).

Manic, M. (2015). The rise of native advertising. *Bulletin of the Transilvania University of Brasov. Economic Sciences. Series V*, 8(1):53.

Martin, E. J. (2017). How Virtual and Augmented Reality Ads Improve Consumer Engagement. http://www.econtentmag.com/Articles/News/News-Feature/

How-Virtual-and-Augmented-Reality-Ads-Improve-Consumer-Engagement\-117710.htm. (Last accessed: July 2019).

McDonnell, T., Ray, B., and Kim, M. (2013). An Empirical Study of API Stability and Adoption in the Android Ecosystem. In *Proceedings of the 29th IEEE International Conference on Software Maintenance*, ICSME '13, pages 70–79.

Medvidovic, N. and Jakobac, V. (2006). Using Software Evolution to Focus Architectural Recovery. *Automated Software Engineering*, 13(2):225–256.

Medvidovic, N. and Taylor, R. N. (2000). A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, 26(1):70–93.

Moat (2017). Moat Analytics. https://moat.com/. (Last accessed: July 2019).

MobileMarketing (2018). Rewarded video ads will continue to dominate through 2018. https://mobilemarketingmagazine.com/rewarded-video-ads-will-continue-to-dominate-through-2018. (Last accessed July 2019).

MoPub (2016). Customize the MoPub SDK for only the formats you use. https://www.mopub.com/2016/09/15/customize-the-mopub-sdk-for-only-the-formats-you-u. (Last accessed: July 2019).

NinthDecimal (2019). NinthDecimal. https://www.ninthdecimal.com/. (Last accessed: July 2019).

Preston-Werner, T. (2013). Semantic versioning 2.0.0. https://semver.org/. (Last accessed: July 2019).

ProGuard (2013). ProGuard. https://www.guardsquare.com/en/products/proguard. (Last accessed: July 2019).

Quantcast (2019). Quantcast. https://www.quantcast.com/. (Last accessed: July 2019).

Quora (2016). How do you decide which Ad Network is best for your Mobile App (eCPM, retention)? https://www.quora.com/How-do-you-decide-which-Ad-Network-is-best-for-your-Mobile-App\-eCPM-retention. (Last accessed: July 2019).

Quora (2017). Should I create a mobile ad mediation for internal use or use an existing ad mediation solution (Fyber, Appodeal, Heyzap)? https://www.quora.com/Should-I-create-a-mobile-ad-mediation-for-internal-use-or-\use-an-existing-ad-mediation-solution-Fyber-Appodeal-Heyzap /answer/Eric-Benjamin-Seufert. (Last accessed January 2019).

ReactiveX (2013). ReactiveX. http://reactivex.io/intro.html. (Last accessed: July 2019).

Reinhardt, P. (2016). Effect of Mobile App Size on Downloads. https://segment.com/blog/mobile-app-size-effect-on-downloads/. (Last accessed: July 2019).

Roy, B., Mondal, A. K., Roy, C. K., Schneider, K. A., and Wazed, K. (2017). Towards a Reference Architecture for Cloud-Based Plant Genotyping and Phenotyping Analysis

Frameworks. In *IEEE International Conference on Software Architecture*, ICSA'17, pages 41–50.

Ruiz, I. J. M., Nagappan, M., Adams, B., Berger, T., Dienst, S., and Hassan, A. E. (2014). Impact of Ad Libraries on Ratings of Android Mobile Apps. *IEEE Software*, 31(6):86–92.

Ruiz, I. J. M., Nagappan, M., Adams, B., Berger, T., Dienst, S., and Hassan, A. E. (2016). Analyzing Ad Library Updates in Android Apps. *IEEE Software*, 33(2):74–80.

RxJava (2013). RxJava. https://github.com/ReactiveX/RxJava. (Last accessed: July 2019).

Statista (2019). Number of mobile app downloads worldwide in 2017, 2018 and 2022 (in billions). https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/. (Last accessed: July 2019).

Terkki, E., Rao, A., and Tarkoma, S. (2017). Spying on Android users through targeted ads. In *Proceedings of the 9th International Conference on Communication Systems and Networks*, COMSNETS'17, pages 87–94.

Understand (2015). Understand Tool. https://scitools.com/. (Last accessed: July 2019).

Vungle (2019). Generate more revenue. https://vungle.com/monetize/. (Last accessed: July 2019).

Wang, H., Li, H., and Guo, Y. (2019). Understanding the Evolution of Mobile App

Ecosystems: A Longitudinal Measurement Study of Google Play. In *The World Wide Web Conference*, WWW '19, pages 1988–1999.

APPENDIX A

Identified Ad Libraries

Table A.1 shows the list of identified 63 ad libraries. In addition, Table A.2 shows the list of 303 packages that we manually analyzed over the web.

Table A.1: List of identified 63 ad libraries.

Ad Library	**Package Name**
AdColonoy	com.jirbo.adcolony / com.adcolony
AdinCube	com.adincube
AdMarvel	com.admarvel
Admob	com.admob
AdServ	com.adserv.sdk
AdTech	com.adtech
AdUWant	com.aduwant.ads
AdWhirl	com.adwhirl
AerServ	com.aerserv.sdk
Altamob	com.altamob.sdk
Amazon Mobile Ad	com.amazon.device.ads
Amobee	com.amobee.adsdk
AOL	com.aol
Appbrain	com.appbrain
AppInTop	com.appintop
Applovin	com.applovin
AppNext	com.appnext
Appodeal	com.appodeal.ads
Avocarrot	com.avocarrot.sdk
Bee7	com.bee7
Calldorado Mobile SDK	com.calldorado
Chartboost	com.chartboost.sdk
CMAdSDK	com.cmcm
DoApp	com.doapps
DU Ads platform	com.duapps
Facebook Audience Network	com.facebook.ads

Ad Library	Package Name
Flurry	com.flurry.android.ads
FreeWheel	tv.freewheel
Fyber	com.fyber
Google AdMob	com.google.android.gms.ads
HeyZap	com.heyzap
InMobi	com.inmobi
Inneractive	com.inneractive.api.ads
Integral Ad	com.integralads
ironSource	com.ironsource
JumpTap	com.jumptap
JustAd	tv.justad
Kuala Ad	com.xinmei
Loopme	com.loopme
Medialets	com.medialets
Millennialmedia	com.millennialmedia
MobFox	com.mobfox.sdk
MobiMagic	com.mobimagic
MobVista	com.mobvista
MoPub	com.mopub
myTarget	com.my.target
Openex	com.openx
Qihoo 360	com.qihoo
RevMob	com.revmob
Rovio	com.rovio
Smaato	com.smaato
Smart AdServer	com.smartadserver.android
SmartCross	com.smartcross
Smato	com.smaato
Sponorpay	com.sponsorpay
StartApp	com.startapp
Supersonic	com.supersonic
Tapdaq	com.tapdaq
Tapit	com.tapit
Tapjoy	com.tapjoy
Unity3d Ads	com.unity3d.ads
Verve Wireless	com.vervewireless
Vungle	com.vungle

Table A.2: List of 303 packages that we manually search on the web for ad library identification.

Package name		
adyen.com	com.cyberlink	com.medialets
android.databinding	com.daimajia	com.microsoft
antistatic.spinnerwheel	com.devbrackets	com.mikepenz
app.teamv	com.dexati	com.milkmangames
com.acb	com.dianxinos	com.millennialmedia
com.ad_stir	com.digitalchemy	com.miniclip
com.ad4screen	com.disney	com.mobfox
com.adapter	com.doapps	com.mobile
com.adapters	com.dotc	com.mobimagic
com.adclient	com.dreamsocket	com.mobimento
com.adcolony	com.drew	com.mobisystems
com.addlive	com.droid27	com.mobsandgeeks
com.adincube	com.duapps	com.mobvista
com.adjust	com.ducaller	com.moodstocks
com.admarvel	com.ea	com.my
com.admob	com.ensighten	com.nbc
com.admob_mediation	com.espn	com.nbcuni
com.adobe	com.etiennelawlor	com.newrelic
com.ads	com.etsy	com.nextplus
com.adsbase	com.everyplay	com.nq
com.adsdk	com.exacttarget	com.ntracecloud
com.adsmob	com.example	com.onelouder
com.adtech	com.expedia	com.onemobile
com.adtoapp	com.facebook	com.ooyala
com.aduwant	com.flurry	com.openx
com.adwhirl	com.flymob	com.outfit7
com.adxcorp	com.fotoable	com.ovuline
com.adyen	com.fusepowered	com.parbat
com.aerserv	com.fw	com.passportparking
com.airwatch	com.fyber	com.pinger
com.altamob	com.gismart	com.pingstart
com.amazon	com.github	com.pinsightmediaplus
com.amazonaws	com.glow	com.pocketprep
com.amobee	com.gokeyboard	com.pop

Package name		
com.androidnative	com.google	com.prime31
com.antivirus	com.googlecode	com.publisheriq
com.anvato	com.greedygame	com.purplebrain
com.aol	com.h6ah4i	com.qbiki
com.apalon	com.hannesdorfmann	com.qihoo
com.appboy	com.helpshift	com.qisi
com.appintop	com.heyzap	com.qq
com.applicaster	com.hp	com.radaee
com.applovin	com.hudomju	com.rcplatform
com.appnext	com.iconology	com.revmob
com.appnexus	com.ihandysoft	com.rfm
com.appodeal	com.ihs	com.riffsy
com.apprupt	com.iinmobi	com.rjfun
com.appsflyer	com.ijinshan	com.rovio
com.apptentive	com.ijoysoft	com.scoompa
com.apptracker	com.imo	com.seatgeek
com.apus	com.inlocomedia	com.seattleclouds
com.arlib	com.inmobi	com.sec
com.asherjunk	com.inneractive	com.segment
com.att	com.inqbarna	com.sgiggle
com.auditude	com.intentsoftware	com.sileria
com.avast	com.intercom	com.smartcross
com.avg	com.ironsource	com.sololearn
com.avg	com.ironsource	com.sololearn
com.avocarrot	com.jakewharton	com.sponsorpay
com.babycenter	com.jb	com.sports
com.badoo	com.jirbo	com.startapp
com.bamnetworks	com.jiubang	com.supersonic
com.bee7	com.jumio	com.sygic
com.behance	com.jumptap	com.taobao
com.box	com.kika	com.tapdaq
com.braintreepayments	com.kikatech	com.tapit
com.burstly	com.krux	com.tapjoy
com.calldorado	com.layer	com.tapsense
com.chad	com.lemon	com.tesolutions
com.cleanmaster	com.library	com.textmeinc
com.cloudtech	com.life360	com.tme
com.cmcm	com.lifestreet	com.tools

Package name		
com.commerce	com.liverail	com.tremorvideo
com.commonsware	com.longtailvideo	com.trulia
com.contextlogic	com.loopme	com.turner
com.conviva	com.lyrebirdstudio	com.uber
com.cootek	com.magic	com.ubercab
com.cube	com.mapmyfitness	com.uc
com.ucweb	com.yinzcam	mobi.wifi
com.unity3d	com.yume	mono.com
com.univision	com.zenjoy	nativesdk.ad
com.upalytics	com.zeus	net.adways
com.upsight	com.zooz	net.afpro
com.urbanairship	CoronaProvider.ads	net.hockeyapp
com.usage	de.guj	net.pubnative
com.uservoice	emoji.keyboard	org.adw
com.vdopia	gov.nih	org.andengine
com.vervewireless	imoblife.luckad	org.andengine
com.video	in.ubee	org.apache
com.virgo	io.presage	org.droidparts
com.visa	io.smooch	org.holoeverywhere
com.vungle	javazoom.jl	org.jdom
com.wantu	jp.co	org.jivesoftware
com.wsi	jp.wasabeef	org.lds
com.xinmei	kankan.wheel	org.mozilla
com.xlabz	kotlin.reflect	org.restlet
com.xtify	ks.cm	org.robobinding
com.xvideostudio	ly.kite	org.saturn
com.yahoo	me.dingtone	psm.advertising
com.yandex	me.everything	retrofit2.adapter
me.iwf	roboguice.adapter	ru.mail
me.tango	tv.freewheel	mobi.charmer
mobi.infolife		