




A longitudinal study of popular ad libraries in the Google Play Store

Md Ahasanuzzaman¹  · Safwat Hassan¹ · Cor-Paul Bezemer² · Ahmed E. Hassan¹

Published online: 12 December 2019

© Springer Science+Business Media, LLC, part of Springer Nature 2019

Abstract

In-app advertisements have become an integral part of the revenue model of mobile apps. To gain ad revenue, app developers integrate ad libraries into their apps. Such libraries are integrated to serve advertisements (ads) to users; developers then gain revenue based on the displayed ads and the users' interactions with such ads. As a result, ad libraries have become an essential part of the mobile app ecosystem. However, little is known about how such ad libraries have evolved over time. In this paper, 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), 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

Communicated by: Romain Robbes

✉ Md Ahasanuzzaman
md.ahasanuzzaman@queensu.ca

Safwat Hassan
shassan@cs.queensu.ca

Cor-Paul Bezemer
bezemer@ualberta.ca

Ahmed E. Hassan
ahmed@cs.queensu.ca

¹ Software Analysis and Intelligence Lab (SAIL), Queen's University, Kingston, Ontario, Canada

² Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Canada

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 platforms (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.

Keywords Android mobile apps · Ad library · Google Play Store · Longitudinal study · Software engineering

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 Intelligence 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.

Figure 1 shows an overview of the in-app advertising model. As shown in Fig. 1, the in-app advertising model contains three main components: (1) advertiser companies (i.e., companies that pay to show advertisements to promote their products), (2) integrating apps (i.e., apps that promote products and earn revenue by displaying advertisements of products (Hao et al. 2017)), (3) mobile ad networks that act as a bridge between the integrating apps and advertiser companies.

To display advertisements, every ad network provides an ad library that needs to be integrated into the integrating apps. 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. 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 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 important for ad library 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 2.2 describes our process for identifying the integrated ad libraries). Second, for each update of an app, we calculated

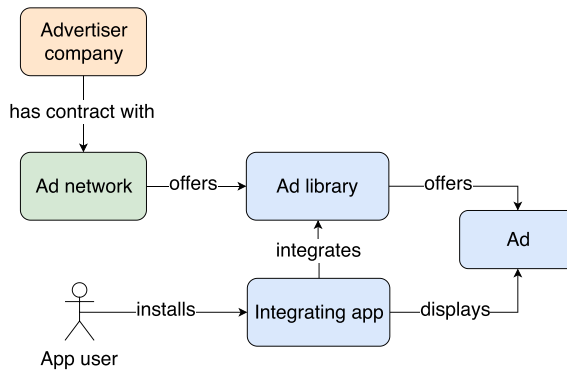


Fig. 1 An overview of the in-app advertising model

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 2 shows the size of the studied app binaries and the combined size of their integrated ad libraries during 18 months (from April 20th 2016 to September 20th 2017). As shown in Fig. 2, 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:

$$\text{Ad library growth ratio } (A) = \frac{PAL_{\text{end}}(A)}{PAL_{\text{start}}(A)} \quad (1)$$

Where $PAL_{\text{start}}(A)$ and $PAL_{\text{end}}(A)$ 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

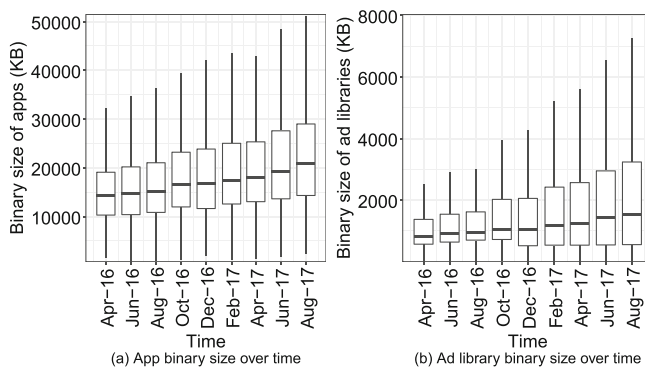


Fig. 2 The size of apps and ad libraries during a 1.5 year period

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 paper we 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 (Segment 2019; Google 2019c). 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 important for ad library 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; Medvidovic and Jakobac 2006; Hassan and Holt 2000, 2002), 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 paper is organized as follows. Section 2 describes our data collection process. Section 3 presents the results of our longitudinal study of ad libraries. Section 4

describes the quality attributes of our derived reference architecture for ad libraries. Section 5 discusses the implications of our work. Section 6 describes the threats to validity of our findings. Section 7 presents related work, and Section 8 concludes the paper.

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 3 gives an overview of the main steps of our data collection process. We detail each step below.

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

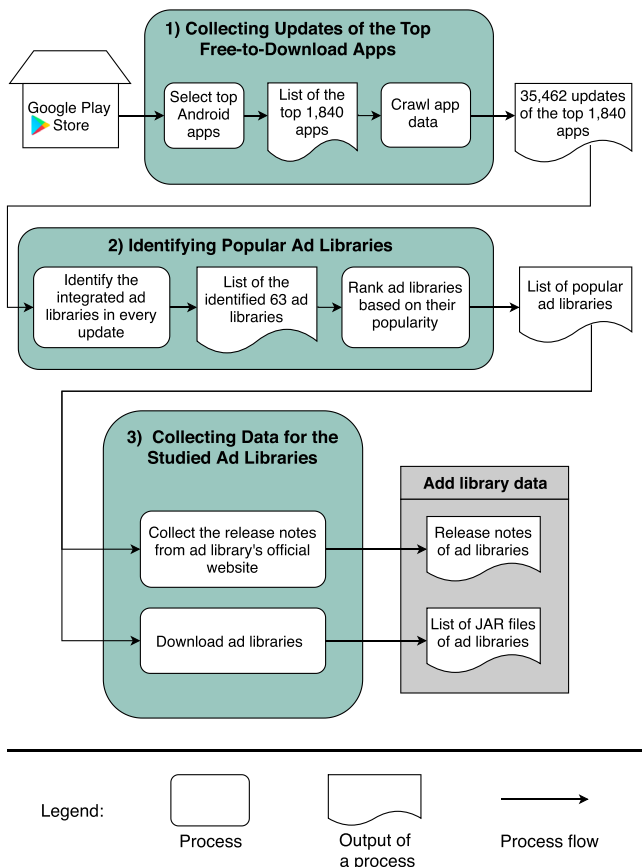


Fig. 3 An overview of our data collection process

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 2019) for 18 months from April 20th 2016 to September 20th 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.

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 2019). Then, we used the BCEL tool (Apache Software Foundation 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 regular expression also matches class names that are not related to ad libraries (e.g., [com.facebook.load.ImageLoad](#)). Hence, to identify ad libraries, we followed Ruiz et al.'s (2016) approach 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 (with ad libraries) integrate one or more of these libraries. Table 1 shows the top ten popular ad libraries.

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.

Table 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

3 A Longitudinal Study of the Evolution of Ad Libraries

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.

3.1 RQ1: How Often are Ad Libraries Released, and How Large are These Releases?

Motivation: In Section 3.1, 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 (Segment 2019; Google 2019c). 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.

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 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 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.

Table 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

The size of all the studied ad libraries (except Google AdMob and Vungle) increases over time Figure 4 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 Fig. 4, 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 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).

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 2019) whose size of 924KB, is almost half of the Vungle library size. The RxJava library is useful for applications that are designed for reactive programming (ReactiveX 2019) (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.

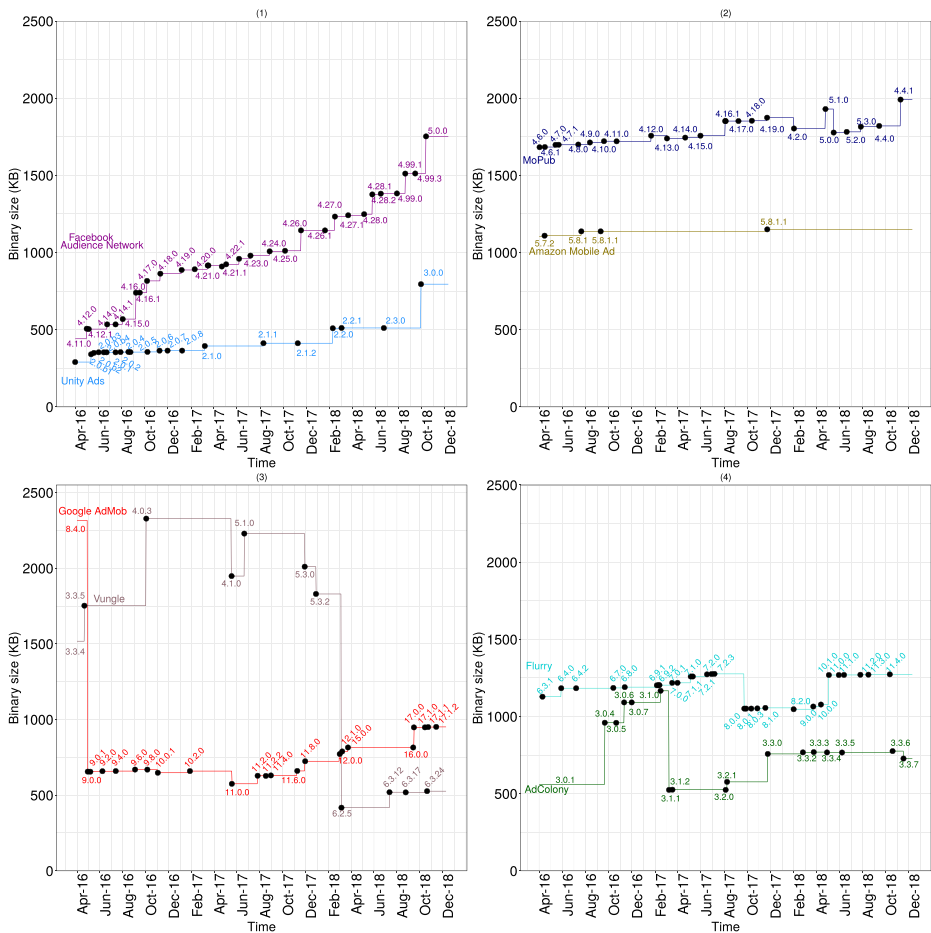


Fig. 4 The identified trends in the size evolution of the studied ad libraries: (1) *Explosive growth*, (2) *Stable growth*, (3) *Shrinkage*, and (4) *Fluctuating size*

Although the size of most ad libraries increased, ad library developers took measures to limit the growth in size. Figure 4 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

Table 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

Ads Lite (Google Mobile Ads Lite SDK 2019). 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 Mobile Ads Lite SDK 2019). 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 2019).

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.

3.2 RQ2: What Drives Ad Library Developers to Release a New Version?

Motivation: In Section 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: The first and the second author of this paper (as two *coders*) independently followed an iterative approach that is similar to the open coding method (Khandkar 2009). Each author 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 authors 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” (psych 2019) 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 shows the list of the identified drivers along with the description and an example of each driver.

Findings: **Adding and improving the streaming video ad functionality is the most occurring driver for releasing an ad library version.** Table 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

Table 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	<i>D:</i> The version fixes a crash or exception in an ad library. <i>E:</i> “Fixed crash when interacting with the screen after rewarded video finishes and before showing the end card.”
	API refactoring	<i>D:</i> The version refactors (e.g., adds, removes, or modifies) methods or classes that are related to the ad library APIs. <i>E:</i> “Removed the deprecated <code>EventListener.onVideoView()</code> API.”
	Improve code obfuscation	<i>D:</i> The version obfuscates the ad library code (e.g., using ProGuard (ProGuard 2013)). <i>E:</i> “Resolves a ProGuard issue introduced in 9.0.0.”
	Improve logging	<i>D:</i> The version improves the logged information. <i>E:</i> “Improved logging when attempting to show an ad that is not ready.”
	Manage orientation and layout	<i>D:</i> The version improves the layout of the displayed ads. <i>E:</i> “Support for vertical ads and improved ad orientation controls.”
Managing the displayed content	Display video streaming ad	<i>D:</i> The version adds new features for video streaming or fixes issues that are related to video ads. <i>E:</i> “Added support for rewarded video”
	Fix privacy issues	<i>D:</i> The version fixes users’ privacy issues. <i>E:</i> “Removed collection of IMEI as per Google Play Content Developer Policy”, “Removed MAC address tracking.”
Accessing user data	Improve analytics	<i>D:</i> The version improves ads analytics features <i>E:</i> “Improvements on analytics.”
	Support user device models	<i>D:</i> The version fixes issues that are related to the interaction with user’ devices. <i>E:</i> “Bug preventing MediaPlayer from resuming playback on certain devices.”
	Support Android platforms	<i>D:</i> The version provides support for new Android platforms or fixes issues in the supported Android platforms <i>E:</i> “Added support for Android Nougat (Android v7.0)”
Resource optimization	Optimize network resources	<i>D:</i> The version improves the communication with ad networks. <i>E:</i> “Replaced usage of <code>NSURLConnection</code> with <code>NSURLSession</code> for optimizing ad server communication protocols.”
	Optimize memory	<i>D:</i> The version improves the memory management (e.g., caching mechanism) of ad libraries.

Table 4 (continued)

Driver category	Driver name	Description (D) - Example (E)
General features	resources	<i>E</i> : “Fix Memory leak caused by LocalBroadcastReceiver holding onto MediaView reference.”
	Optimize energy resources	<i>D</i> : The version improves the energy consumption (e.g., battery drain) of ad libraries. <i>E</i> : “Improvements to reduce battery drain.”
	Optimize device storage resources	<i>D</i> : The version fixes issues that are related to using device storage. <i>E</i> : “Fixed storage overuse issue reported by a small number of publishers upgrading from 2.x -> 3.x.”
	Integrate with other ad networks	<i>D</i> : The version supports the communication with other ad networks. <i>E</i> : “Added and updated mediated network versions.”
	Unspecified	<i>D</i> : The release note does not contain detailed information about the fixed issues or the added features. <i>E</i> : “bug fixes”

Table 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

video controls, (3) handling native video ads, and (4) prefetching video ads. Table 6 shows the description and an example of the identified video streaming features.

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*”)

Table 6 The main identified features for video streaming ads

Category	Description (D) - Example (E)
Offering reward videos	<p>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.</p> <p>E: “<i>Rewarded video support from the MoPub Marketplace (Beta).</i>”</p>
Adding video controls	<p>D: Ad library developers allow users to control (e.g., pause, resume, mute, and replay) the displayed video ads.</p> <p>E: “<i>Added new design for play/pause button in Rewarded Video.</i>”</p>
Handling native video ads	<p>D: Ad library developers improve or add features that display video ads that have the same look and feel as the integrating app.</p> <p>E: “<i>Added the <code>setAdChoicesPlacement()</code> method to the <code>NativeAdOptions.Builder</code> class, which app publishers can now use to specify the location of their <code>AdChoices</code> in native ads.</i>”</p>
Prefetching video ads	<p>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.</p> <p>E: “<i>Video cache limit updated to 64mb for prefetching.</i>”</p>

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*”).

Ad libraries leverage users’ information to provide analytics features for integrating apps As shown in Table 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 7 shows the analytics features that were added or improved by the studied ad libraries during the studied 33 months, along with the collected user data.

As shown in Table 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 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 2019a). We observed during the

Table 7 The main identified features that are offered in ad analytics

Analytics main features	Collected data	Description (<i>D</i>) - Example (<i>E</i>)
Provide metrics about the displayed ads	Ad session data	<i>D</i> : Collect the duration of user engagement with an ad (i.e., how long users watch a video ad before closing it). <i>E</i> : MoPub has a <code>ExternalViewabilitySessionManager</code> class that provides methods (e.g., <code>createVideoSession</code> , <code>recordVideoEvent</code>) to capture the session information.
	Ad revenue	<i>D</i> : Collect metrics (e.g., ad viewability ratio (Measuring Ad Viewability 2019), click through ratio (The simple guide to understand facebook ads metrics 2019)) that are useful for estimating ad revenue.
	metrics	<i>E</i> : “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	<i>D</i> : 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. <i>E</i> : <i>Unity Ads collect advertising id with the method named <code>fetchAdvertisingId</code></i>
	Device information	<i>D</i> : Collect data related to a user’s device information (e.g., device model) to improve the displayed ads. <i>E</i> : “Reporting more device stats to serve better and better ads.”
	Demographic data	<i>D</i> : Collect a user’s demographic data (e.g., language, country, and location information) to display ads that are suitable for that particular demographic. <i>E</i> : “Added auto-population of location information for apps that explicitly grant the location permission.”

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.

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.

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.

Step 2: Generating a concrete architecture of each ad library. In this step, we used the Understand tool (Understand Tool 2019) to generate and visualize the dependency call graph of each version of each studied ad library.¹ 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.

¹The Understand projects of each of the studied ad libraries are available from our supplementary data: https://github.com/SAILResearch/suppmaterial-18-ahsan-ads-provider_libs.

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 (2000).

Figure 5 shows our proposed reference architecture, and Table 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.

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.

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

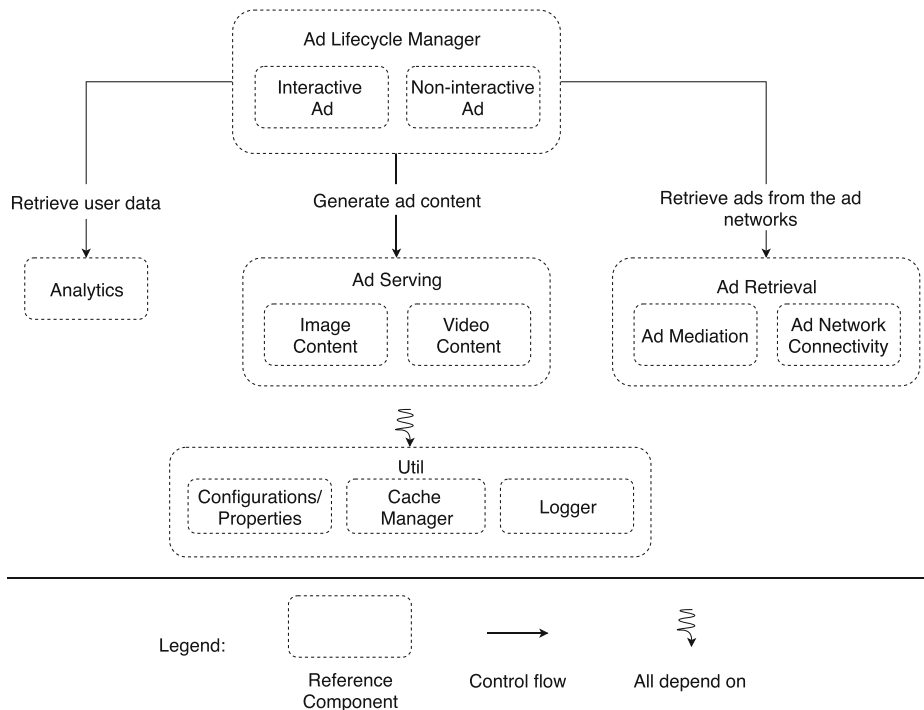


Fig. 5 Ad library reference architecture. A line between two components indicates that there is a relationship between the components

Table 8 The identified components of the reference architecture of ad libraries

Component name	Component definition
Ad lifecycle manager	The <i>Ad Lifecycle Manager</i> 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 <i>Ad Lifecycle Manager</i> performs its functionality using two subcomponents (<i>Interactive Ad</i> and <i>Non-interactive Ad</i>). The <i>Interactive Ad</i> 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 (Why Playable Ads are the Key to Engaged Users 2019; How Virtual and Augmented Reality Ads Improve Consumer Engagement 2019). 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 <i>Non-interactive Ad</i> , users are allowed only to control (e.g., play or close) the displayed ads. This component consists of the following four subcomponents: (1) <i>Banner</i> , (2) <i>Native</i> , (3) <i>Interstitial</i> (full-screen), and (4) <i>Rewarded Video</i> .
Ad retrieval	The <i>Ad Retrieval</i> component provides functionality to communicate with ad networks and download ad data. In particular, the <i>Ad Lifecycle Manager</i> sends messages to the <i>Ad Retrieval</i> component for fetching ads from an ad network. Then, the <i>Ad Retrieval</i> component fetches ad data (lightweight format such as JSON) from the ad networks by using HTTP requests.
Ad serving	The main functionality of the <i>Ad Serving</i> 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 <i>Analytics</i> 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 <i>Ad Lifecycle Manager</i> for fetching and displaying appropriate ads for every user.
Util	The <i>Util</i> 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 sub-components: (1) <i>Cache Manager</i> , (2) <i>Logger</i> , and (3) <i>Configuration/Properties</i> .

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 (Why Playable Ads are the Key to Engaged Users 2019; How Virtual and Augmented Reality Ads Improve Consumer Engagement 2019).

We can use these components to identify the differences between ad libraries in terms of their supported ad formats. For example, Fig. 6 shows the architecture of the Vungle and the

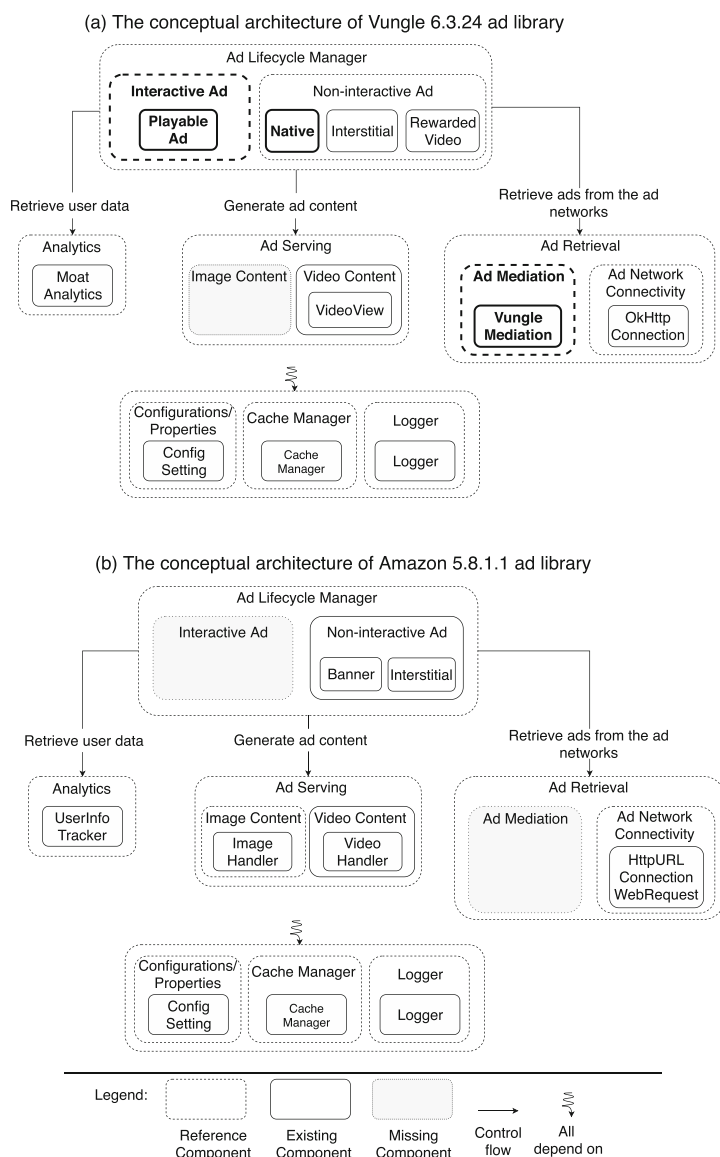


Fig. 6 The differences between the ad library architectures of the Vungle and Amazon ad libraries. A bold box with a bold font shows a new component that appears in the version of the ad library

Amazon Mobile Ad library. In Fig. 6, we highlight all components that exist in the reference architecture but not in the particular ad library. As shown in Fig. 6, 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 6 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, Fig. 7 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 (Why Playable Ads are the Key to Engaged Users 2019).

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.

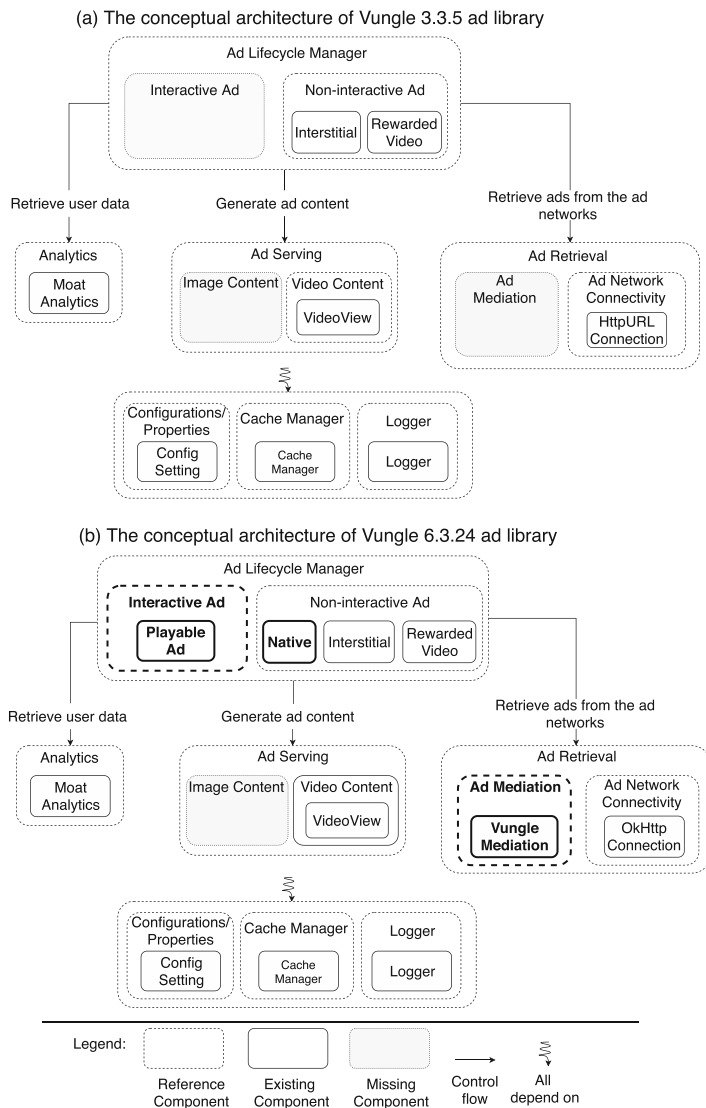


Fig. 7 Evolution of the architecture of the Vungle ad library during our study period. A bold box with a bold font shows a new component that appears in the version of the ad library

4 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.

4.1 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.

4.2 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.

4.3 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 Minimizing the Size of the Ad Library

The size of an app is negatively associated with the number of installations of an app (Segment 2019). 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 2019), their modular architecture allows app developers to save up to 60% of the library size by including only the needed modules.

5 Implications

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

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 (Segment 2019). Hence, it is necessary to keep app size (and therefore the size of all its libraries) as small as possible. As described in Section 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 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 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 2019d) to identify memory leaks.

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 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.

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, the first and the second author 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.

7 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 study is the first to investigate the evolution of the ad libraries themselves. We discuss the related work below.

7.1 The Updating 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. The 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 work 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.

7.2 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.

7.3 The Security of Ad Libraries

Prior work shows that privacy and security are emerging issues in mobile apps (Calciati and Gorla 2017; Calciati et al. 2018; Felt et al. 2011; Au et al. 2012; Backes et al. 2016; Wang et al. 2019). 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.

8 Conclusions

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 paper, we conduct a longitudinal analysis of eight 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.

Appendix: Identified Ad Libraries

Table 9 shows the list of identified 63 ad libraries. In addition, Table 10 shows the list of 303 packages that we manually analyzed over the web.

Table 9 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
CMAAdSDK	com.cmcm
DoApp	com.doapps
DU Ads platform	com.duapps
Facebook Audience Network	com.facebook.ads
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

Table 9 (continued)

Ad library	Package name
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
Sponsorpay	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 10 List of 303 packages that we manually search on the web for ad library identification

Package name			
adyen.com	com.cyberlink	com.medialets	com.ucweb
android.databinding	com.daimajia	com.microsoft	com.unity3d
antistatic.spinnerwheel	com.devbrackets	com.mikepenz	com.univision
app.teamv	com.dexati	com.milkmangames	com.upalytics
com.acb	com.dianxinos	com.millennialmedia	com.upsight
com.ad_stir	com.digitalchemy	com.miniclip	com.urbanairship
com.ad4screen	com.disney	com.mobfox	com.usage
com.adapter	com.doapps	com.mobile	com.uservoice
com.adapters	com.dotc	com.mobimagic	com.vdopia
com.adclient	com.dreamsocket	com.mobimento	com.vervewireless
com.adcolony	com.drew	com.mobisystems	com.video
com.addlive	com.droid27	com.mobsandgeeks	com.virgo
com.adincube	com.duapps	com.mobvista	com.visa

Table 10 (continued)

Package name			
com.adjust	com.ducaller	com.moodstocks	com.vungle
com.admarvel	com.ea	com.my	com.wantu
com.admob	com.ensighten	com.nbc	com.wsi
com.admob.mediation	com.espn	com.nbcuni	com.xinmei
com.adobe	com.etiennelawlor	com.newrelic	com.xlabz
com.ads	com.etsy	com.nextplus	com.xtify
com.adsgbase	com.everyplay	com.nq	com.xvideostudio
com.adsdk	com.exacttarget	com.ntracecloud	com.yahoo
com.adsmob	com.example	com.onelouder	com.yandex
com.adtech	com.experia	com.onemobile	com.yinzcam
com.adtoapp	com.facebook	com.ooyala	com.yume
com.aduwant	com.flurry	com.openx	com.zenjoy
com.adwhirl	com.flymob	com.outfit7	com.zeus
com.adxcorp	com.fotoable	com.ovuline	com.zooz
com.adyen	com.fusepowered	com.parbat	CoronaProvider.ads
com.aerserv	com.fw	com.passportparking	de.guj
com.airwatch	com.fyber	com.pinger	emoji.keyboard
com.altamob	com.gismart	com.pingstart	gov.nih
com.amazon	com.github	com.pinsightmediaplus	imoblife.luckad
com.amazonaws	com.glow	com.pocketprep	in.ubee
com.amobee	com.gokeyboard	com.pop	io.presage
com.androidnative	com.google	com.prime31	io.smooch
com.antivirus	com.googlecode	com.publisheriq	javazoom.jl
com.anvato	com.greedygame	com.purplebrain	jp.co
com.aol	com.h6ah4i	com.qbiki	jp.wasabeef
com.apalon	com.hannedorfmann	com.qihoo	kankan.wheel
com.appboy	com.helpshift	com.qisi	kotlin.reflect
com.appintop	com.heyzap	com.qq	ks.cm
com.applicaster	com.hp	com.radaee	ly.kite
com.applovin	com.hudomju	com.rcplatform	me.dingtone
com.appnext	com.iconology	com.revmob	me.everything
com.appnexus	com.ihandysoft	com.rfm	me.iwf
com.appodeal	com.ihs	com.riffsy	me.tango
com.apprupt	com.iinmobi	com.rjfun	mobi.charmer
com.appsflyer	com.ijinshan	com.rovio	mobi.infolife
com.apptentive	com.joysoft	com.scompa	mobi.wifi
com.apptracker	com.imo	com.seatgeek	mono.com
com.apus	com.inlocomedia	com.seattleclouds	nativesdk.ad
com.arlib	com.inmobi	com.sec	net.adways
com.asherjunk	com.inneractive	com.segment	net.afpro
com.att	com.inqbarna	com.sgiggle	net.hockeyapp
com.auditudo	com.intentsoftware	com.sileria	net.pubnative
com.avast	com.intercom	com.smartcross	org.adw

Table 10 (continued)

Package name			
com.avg	com.ironsource	com.sololearn	org.andengine
com.avg	com.ironsource	com.sololearn	org.andengine
com.avocarrot	com.jakewharton	com.sponsorpay	org.apache
com.babycenter	com.jb	com.sports	org.droidparts
com.badoo	com.jirbo	com.startapp	org.holoeverywhere
com.bamnetworks	com.jiubang	com.supersonic	org.jdom
com.bee7	com.jumio	com.sygic	org.jivesoftware
com.behance	com.jumtap	com.taobao	org.lds
com.box	com.kika	com.tapdaq	org.mozilla
com.braintreepayments	com.kikatech	com.tapit	org.restlet
com.burstly	com.krux	com.tapjoy	org.robobinding
com.callorado	com.layer	com.tapsense	org.saturn
com.chad	com.lemon	com.tesolutions	psm.advertising
com.cleanmaster	com.library	com.textmeinc	retrofit2.adapter
com.cloudtech	com.life360	com.tme	roboguice.adapter
com.cmc	com.lifestreet	com.tools	ru.mail
com.commerce	com.liverail	com.tremorvideo	tv.freewheel
com.commonware	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	

References

- Addo ID, Ahamed SI, Yau SS, Buduru A (2014) A reference architecture for improving security and privacy in internet of things applications. In: 2014 IEEE International conference on mobile services, pp 108–115
- Akdeniz (2019) Google Play Crawler. <https://github.com/Akdeniz/google-play-crawler>. (Last accessed: July 2019)
- Android version history (2019) https://en.wikipedia.org/wiki/Android_version_history. (Last accessed: July 2019)
- Apache Software Foundation (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)
- AppBrain (2016) Video ads. <https://www.appbrain.com/stats/libraries/tag/video-ads/video-ads>. (Last accessed: July 2019)
- AppBrain Intelligence (2019) <https://www.appbrain.com/stats/>. (Last accessed: July 2019)
- AppAnnie (2018) App Annie. <https://www.appannie.com/>. (Last accessed: July 2019)
- Au K W Y, Zhou Y F, Huang Z, Lie D. (2012) PScout: analyzing the Android permission specification. In: Proceedings of the 2012 ACM conference on computer and communications security, CCS '12, pp 217–228
- Backes M, Bugiel S, Derr E, McDaniel P, Octeau D, 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, pp 1101–1118
- Belanche D, Flavián C., Pérez-Rueda A (2017) Understanding interactive online advertising: congruence and product involvement in highly and lowly arousing, skippable video ads. *J Interact Mark* 37:75–88

- Book T, Pridgen A, Wallach DS (2013) Longitudinal analysis of Android ad library permissions. Computing Research Repository, arXiv:[abs/1303.0857](https://arxiv.org/abs/1303.0857)
- Calciati P, 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, pp 37–41
- Calciati P, Kuznetsov K, Bai X, 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, pp 142–152
- Cohen J (1960) A coefficient of agreement for nominal scales. *Educ Psychol Meas* 20(1):37–46
- Davidson D, Fredrikson M, Livshits B (2014) MoRePriv: mobile OS support for application personalization and privacy. In: Proceedings of the 30th annual computer security applications conference, ACSAC '14, pp 236–245
- de la Iglesia JLM, Gayo JEL (2009) Doing business by selling free services. In: *Web 2.0*. Springer, pp 1–14
- Derr E, Bugiel S, Fahl S, Acar Y, 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, pp 2187–2200
- dex2jar (2019) <http://sourceforge.net/projects/dex2jar/>. (Last accessed July 2019)
- Dong F, Wang H, Li L, Guo Y, Bissyandé TF, Liu T, Xu G, 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, pp 257–268
- Dueñas JC, Oliveira WLd, Puente JAdl (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, pp 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 AP, Chin E, Hanna S, Song D, Wagner D (2011) Android permissions demystified. In: Proceedings of the 18th ACM conference on computer and communications security, CCS '11, pp 627–638
- Gao C, Zeng J, Sarro F, Lyu MR, 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, pp 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 (2019a) Best practices for unique identifiers. <https://developer.android.com/training/articles/user-data-ids>. (Last accessed: July 2019)
- Google (2019b) Manage your app's memory. <https://developer.android.com/topic/performance/memory>. (Last accessed: July 2019)
- Google (2019c) 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 (2019d) View the Java heap and memory allocations with memory profiler. <https://developer.android.com/topic/performance/memory>. (Last accessed: July 2019)
- Google Mobile Ads Lite SDK (2019) <https://developers.google.com/admob/android/lite-sdk>. (Last accessed: July 2019)
- Grace MC, Zhou W, Jiang X, 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, pp 101–112
- Grosskurth A, Godfrey MW (2005) A reference architecture for web browsers. In: Proceedings of the 21st international conference on software maintenance, ICSM '05, pp 661–664
- Gui J, McIlroy S, Nagappan M, Halfond WGJ (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, pp 100–110
- Gui J, Nagappan M, Halfond WG (2017) What aspects of mobile ads do users care about? An empirical study of mobile in-app ad reviews. arXiv:[1702.07681](https://arxiv.org/abs/1702.07681)
- Guo C, Zhang J, Yan J, Zhang Z, Zhang Y (2013) Characterizing and detecting resource leaks in android applications. In: Proceedings of the 28th international conference on automated software engineering, ASE '13, pp 389–398
- Hao L, Guo H, Easley RF (2017) A mobile platform's in-app advertising contract under agency pricing for app sales. *Prod Oper Manag* 26(2):189–202
- Hassan AE, Holt RC (2000) A reference architecture for web servers. In: Proceedings of the 7th working conference on reverse engineering, WCRE '10, pp 150–160

- Hassan AE, Holt RC (2002) Architecture recovery of web applications. In: Proceedings of the 24th international conference on software engineering, ICSE '02, pp 349–359
- Hassan S, Shang W, Hassan AE (2017) An empirical study of emergency updates for top android mobile apps. *Empir Softw Eng* 22(1):505–546
- Hassan S, Bezemer C, Hassan AE (2018) Studying bad updates of top free-to-download apps in the Google Play Store. *IEEE Trans Softw Eng*, 1–21
- How Virtual and Augmented Reality Ads Improve Consumer Engagement (2019) <http://www.econtentmag.com/Articles/News/News-Feature/How-Virtual-and-Augmented-Reality-Ads-Improve-Consumer-Engagement-117710.htm>. (Last accessed: July 2019)
- Khandkar SH (2009) Open coding. <http://pages.cpsc.ucalgary.ca/saul/wiki/uploads/CPSC681/open-coding.pdf>. (Last accessed: July 2019)
- Kim D, Son S, Shmatikov V (2016) What mobile ads know about mobile users. In: Proceedings of the 23rd annual network and distributed system security symposium, NDSS '16, pp 1–14
- Landis JR, Koch GG (1977) The measurement of observer agreement for categorical data. *Biometrics*, 33
- Li L, Bissyandé TF, Klein J, Traon YL (2016) An investigation into the use of common libraries in android apps. In: Proceedings of the 23rd software analysis, evolution, and reengineering, SANER '16, pp 403–414
- Manic M (2015) The rise of native advertising. *Bulletin of the Transilvania University of Brasov. Econ Sci Series V* 8(1):53
- McDonnell T, Ray B, 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, pp 70–79
- Measuring Ad Viewability (2019) <https://www.thinkwithgoogle.com/feature/viewability/>. (Last accessed: July 2019)
- Medvidovic N, Jakobac V (2006) Using software evolution to focus architectural recovery. *Autom Softw Eng* 13(2):225–256
- Medvidovic N, Taylor RN (2000) A classification and comparison framework for software architecture description languages. *IEEE Trans Softw Eng* 26(1):70–93
- MoPub (2019) 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)
- Preston-Werner T (2013) Semantic versioning 2.0.0. <https://semver.org/>. (Last accessed: July 2019)
- ProGuard (2013) <https://www.guardsquare.com/en/products/proguard>. (Last accessed: July 2019)
- psych (2019) Procedures for psychological, psychometric, and personality research. <https://cran.r-project.org/web/packages/psych/index.html>. (Last accessed: July 2019)
- ReactiveX (2019) <http://reactivex.io/intro.html>. (Last accessed: July 2019)
- Roy B, Mondal AK, Roy CK, Schneider KA, Wazed K (2017) Towards a reference architecture for cloud-based plant genotyping and phenotyping analysis frameworks. In: IEEE International conference on software architecture, pp 41–50
- Ruiz IJM, Nagappan M, Adams B, Berger T, Dienst S, Hassan AE (2014) Impact of ad libraries on ratings of android mobile apps. *IEEE Softw* 31(6):86–92
- Ruiz IJM, Nagappan M, Adams B, Berger T, Dienst S, Hassan AE (2016) Analyzing ad library updates in android apps. *IEEE Softw* 33(2):74–80
- RxJava (2019) <https://github.com/ReactiveX/RxJava>. (Last accessed: July 2019)
- Segment (2019) Effect of mobile app size on downloads. <https://segment.com/blog/mobile-app-size-effect-on-downloads/>. (Last accessed: July 2019)
- The simple guide to understand facebook ads metrics (2019) <https://adespresso.com/blog/understand-facebook-ads-metrics-guide/>. (Last accessed: July 2019)
- Understand Tool (2019) <https://scitools.com/>. (Last accessed: July 2019)
- Vungle (2019) Generate more revenue. <https://vungle.com/monetize/>. (Last accessed: July 2019)
- Wang H, Li H, 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, pp 1988–1999
- Why Playable Ads are the Key to Engaged Users (2019) <https://applift.com/blog/playable-ads-2>. (Last accessed: July 2019)



Md Ahasanuzzaman is a graduate student in the Software Analysis and Intelligence Lab (SAIL) at Queen's University, Canada. His research interests include mining software repositories, analyzing mobile apps and app stores, analyzing community question answering sites, natural language processing, and machine learning. His works got published in top venues of Software Engineering (e.g., MSR, SANER, and EMSE). He obtained his BSc from the University of Dhaka (Department of Computer Science and Engineering), Bangladesh. He has been awarded prestigious awards, such as Dean's scholarship award and Prime Minister Gold Medal for his outstanding achievements in the B.Sc program. More about Md Ahasanuzzaman can be read on his website:<https://ahasanuzzaman.com/research/>.



Safwat Hassan currently works as a Postdoctoral Fellow in the Software Analysis and Intelligence Lab (SAIL) at Queen's University, Canada. Hassan worked as a software engineer for ten years in different corporations including Egyptian Space Agency (ESA), HP, EDS, VF Germany (outsourced by HP), and Etisalat. During his ten years in the software industry, he worked on different large-scale systems (varying from Web- Based systems to embedded systems) and in diverse project types (design service, customer support, and R&D) across various domains (Telecommunication, Supply-chain, and Aerospace). His research interests include data mining, big data analytics, software engineering, mobile app store analytics. Contact him at shassan@cs.queensu.ca.



Cor-Paul Bezemer is an assistant professor in the Electrical and Computer Engineering department at the University of Alberta. He heads the Analytics of Software, Games And Repository Data (ASGAARD) lab. Before that, he was a postdoctoral research fellow in the Software Analysis and Intelligence Lab (SAIL) at Queen's University in Kingston, Canada. His research interests cover a wide variety of software engineering and performance engineering-related topics. His work has been published at premier software engineering venues such as the TSE and EMSE journals and the ESEC-FSE, ICSME and ICPE conferences. He is one of the vice-chairs of the SPEC research group on DevOps Performance. Before moving to Canada, he studied at Delft University of Technology in the Netherlands, where he received his BSc (2007), MSc (2009) and PhD (2014) degree in Computer Science. For more information about Cor-Paul and the ASGAARD lab see: <http://asgaard.ece.ualberta.ca/>.



Ahmed E. Hassan is an IEEE Fellow, an ACM SIGSOFT Influential Educator, an NSERC Steacie Fellow, the Canada Research Chair (CRC) in Software Analytics, and the NSERC/BlackBerry Software Engineering Chair at the School of Computing at Queen's University, Canada. His research interests include mining software repositories, empirical software engineering, load testing, and log mining. He received a PhD in Computer Science from the University of Waterloo. He spearheaded the creation of the Mining Software Repositories (MSR) conference and its research community. He also serves/d on the editorial boards of IEEE Transactions on Software Engineering, Springer Journal of Empirical Software Engineering, and PeerJ Computer Science. More information at: <http://sail.cs.queensu.ca/>.