# Analyzing Ad Library Updates in Android Apps

**Israel J. Mojica Ruiz**, McAfee

**Meiyappan Nagappan**, Rochester Institute of Technology

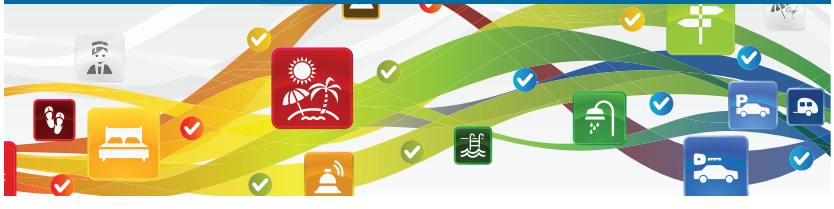
**Bram Adams**, Ecole Polytechnique de Montreal

**Thorsten Berger**, University of Waterloo

**Steffen Dienst**, University of Leipzig

**Ahmed E. Hassan**, Queen's University

// An analysis of Android apps showed that over 12 months, almost half underwent ad library updates. Frequently, no changes to the app's own API occurred. This suggests that maintaining the ad libraries entailed substantial additional effort for the app developers. //



**BECAUSE 91 PERCENT** of the apps in app stores are free,[1] many app developers turn to alternative revenue models. One of the most common revenue models for apps is advertising, in which apps display ads and developers make money every time users click on the ads. To serve the ads, developers must embed specialized code, or *ad libraries*, in their apps. Researchers have estimated that 51 to 60 percent of free Android apps have at least one ad or analytics library.[2–4] (For more on research related to ad libraries, see the sidebar.)

Advertising on mobile devices is a growing business. In 2013, app advertising revenue increased by 56 percent.[5] The Interactive Advertising Bureau and PricewaterhouseCoopers reported that the revenues from mobile ads surpassed $11.5 billion in the first quarter of 2014.[6]

Because the ad industry is highly competitive, ad libraries are regularly updated in apps to embed new ad-serving models or address legal and financial issues (for example, advertising company mergers). App developers must decide whether to integrate the updated ad libraries in their apps. Although an ad library usually doesn't affect an app's core functionality (and so could be safely ignored), ad library updates could significantly impact the app's revenue. However, releasing a new version of an app primarily because of an updated ad library could be a hard sell to existing users. Downloading the new version costs money (especially on a 4G network), and users get concerned about whether the new version is safe (whether it breaks existing behavior or corrupts existing data).

So, ad library updates can incur direct costs due to maintenance and indirect costs due to lost revenue when updates are skipped. If these updates are too frequent and therefore too costly, the software engineering research community and ad library companies need to develop innovative solutions to reduce costs. But if these updates aren't too frequent or solutions could be developed to reduce their costs, their costs might be bearable.

Unfortunately, no research has discussed the frequency of ad library updates in mobile apps.[7] Previous studies and surveys considered only

 0740-7459/16/$33.00 © 2016 IEEE

the cost of developing the first version of an app, with only rough approximations for maintenance costs.[8,9] So, we performed an empirical case study to quantify the prevalence of ad library updates in mobile apps. In almost half of the app versions we studied, an ad library was updated. On the basis of our study results, we explored why the app developers needed to update the libraries.

## Ad-Serving Models

Figure 1 shows the process for serving ads in mobile apps. A company or an advertiser that wants to advertise on mobile apps signs a contract with an advertising company. The advertising company provides ad libraries to app developers, who integrate them into their apps. These libraries are invoked at runtime to serve ads on the apps on the basis of information about the users (for example, their locations or interests). Developers receive financial compensation when users interact with the ad (for example, by clicking on it or buying a product or service through it).

When an app requests an ad from an ad library, the library requests ads from the advertising company's ad pool. However, the average fill rate for ads from ad libraries is only 18 percent.[10] So, app developers embed multiple ad libraries in their apps to maximize their chances of obtaining an ad to display. In such cases, if the first ad library can't fill the ad, the app requests an ad from another one.

## Update Costs

Here we discuss costs related to updating ad libraries in mobile apps. This is by no means a complete list of expenses but rather an initial list (that is, the lower bound of expenses).

## RELATED RESEARCH ON AD LIBRARIES

Prior research on ad libraries has primarily investigated privacy and security issues. Michael Grace and his colleagues analyzed 100 libraries and found that most ad and analytics libraries collected personal information.[1] Some of them even allowed the possibility to execute code unauthorized. William Enck and his colleagues examined the top 1,100 Google Play apps.[2] They discovered that ad and analytics libraries weren't dangerous per se but often exploited tracking identifiers and insecurely transmitted them to their servers. Drew Davidson and Benjamin Livshits eliminated 2,684 unnecessary permissions from 3,120 Android apps.[3] The most common permissions were `INTERNET` (699 apps) and `ACCESS_COARSE_LOCATION` (502 apps). Other researchers have looked into the energy that ads in apps consume. For example, Abhinav Pathak and his colleagues found that free apps used 75 percent more energy owing to poorly developed ad libraries.[4]

### References


1. M.C. Grace et al., "Unsafe Exposure Analysis of Mobile In-App Advertisements," *Proc. 5th ACM Conf. Security and Privacy in Wireless and Mobile Networks* (WISEC 12), 2012, pp. 101–112.
2. W. Enck et al., "A Study of Android Application Security," *Proc. 20th USENIX Conf. Security* (SEC 11), 2011, p. 21.
3. D. Davidson and B. Livshits, "MoRePriv: Mobile OS Support for Application Personalization and Privacy," May 2012; http://research.microsoft.com/pubs/163596/MSR-TR.pdf.
4. A. Pathak, Y.C. Hu, and M. Zhang, "Where Is the Energy Spent inside My App? Fine Grained Energy Accounting on Smartphones with Eprof," *Proc. 7th ACM European Conf. Computer Systems* (EuroSys 12), 2012, pp. 29–42.

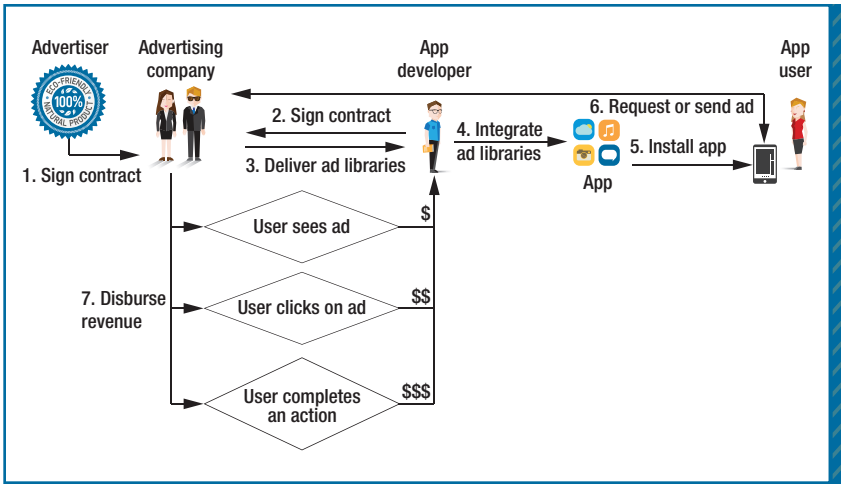




**FIGURE 1.** The process for serving ads in mobile apps. Ad libraries have a low average fill rate, so developers often embed multiple ad libraries in their apps to maximize their chances of obtaining an ad to display.

### Software Maintenance

One major expense is software maintenance. App developers keep track of ad library updates and make sure to include new ad libraries that might provide additional revenue, while removing ad libraries that violate any new app store policies. Once a new version of an ad library is available, app developers embed it in the app and change the app code to reflect the new version (that is, replace calls to the old ad API with calls to the new one). This might even entail redesigning part of the UI to fit in the new ad library. Once app developers change the code, they rebuild and retest the app. All these activities take considerable time and effort,[11] increasing the software maintenance cost.

### App Delivery Delays

Once an app has undergone ad library updates, it gets deployed in app stores. In app stores such as Google Play, in which deployment is fully automated, new apps can become available in just a few hours. In stores such as the Apple App Store, deployment can take days or weeks because each app version is manually verified. But even when the new version of an app is available in the app stores, it's uncertain whether users will update it in their mobile devices. Users don't necessarily welcome new app releases because an update (even if it involves only ad library updates) might disrupt the app's performance.[12] All these delays result in lost revenue.

### Lack of Ads

App developers could lose revenue when they don't update ad libraries. Such revenue loss is due to a lack of ads being served from dead ad libraries,[13] which are libraries that remain in the app but can't serve ads. Developers could also lose revenue when the ads delivered through an old library aren't as interesting to users as ads delivered through a new library. If users aren't interested in an ad, they might not click on it, and it won't generate revenue for the developers.

### Poor User Experience

Advertising companies update ad libraries to provide better ads, improve performance, and fix bugs. However, when app developers ignore these updates, end users can be left with ads that aren't relevant to them, consume too much CPU or battery life, or, in the worst case, access irrelevant private information. Across all app stores, user complaints about ads in apps are often associated with poor ratings and thus affect future downloads of an app.[14] Furthermore, because of the ads, end users often state they'll either not use the app or, worse, uninstall it. In the former case, app developers won't be able to get ad revenue from users. In the latter case, even if the app is updated to address the ad-related issues, users won't get the update notification.

## Identifying Ad Libraries

To perform our study, we needed to identify the ad libraries. Because no list of ad libraries exists, we identified them from data obtained by crawling Google Play.[15] The crawl downloaded all free-to-download apps in 2011, which produced 120,981 Android apps with 236,245 versions. Using this dataset, we identified the ad libraries and their distribution throughout a large number of apps.

The crawled data contained each app's Android Packages (APKs), not its source code. We extracted the Java bytecode from the APKs using dex2jar (http://code.google.com/p/dex2jar). Then, we used the Apache bcel library (http://commons.apache.org/bcel) to extract for each class in each app the fully qualified class name (package or namespace in which a class is contained and the class name) and its set of method names. We manually filtered the fully qualified class names of all apps with the regular expression [aA][dD] (for example, com.package AdlibraryName.AdclassName).

That regular expression led to a large number of matched class names, but these matched classes may not correspond to ad libraries. So, we needed to manually validate the matched classes. For this, we grouped and sorted the fully qualified class names according to their frequency. (The most popular class was com.google.ads.AdActivity, with 149,321 occurrences.) For each fully qualified class name that had a frequency of more than 200 (that is, was included in at least 200 apps, or 0.1 percent of the apps in the crawled data), we did a Web search of the package name to find the website for the ad library's advertising company. We expected each advertising company to have a website for app developers to sign contracts and receive ads and payments. In the end, we identified 72 known ad libraries.

## Update Prevalence

We defined ad library updates as libraries that were added to, deleted from, or modified in an app. We used this definition to determine ad library updates that occurred in an app between versions.

Because we couldn't access the apps' source code, we analyzed extracted signatures from their bytecode. We focused on updates in the actual ad libraries but not in the glue code used to interface with the librar-

ies. The latter would have required us to design static analyses specific to every ad library and deal with obfuscated bytecode (because we would have needed access to the code in each class). Similarly, in the actual ad libraries, we could analyze only the number of libraries that had an update and not the amount of churn (in terms of LOC) in each library.

To determine ad updates, we considered only apps that had at least two versions. Because apps with very few raters might be spam apps[16] and introduce a bias in the results, we also considered only apps that had at least 10 raters in Google Play. This filtering left 5,937 apps with 13,983 versions.

We compared how each app had been updated over time. In particular, we sought to determine whether a new version of an app had ad updates, non-ad (that is, the core of the app) updates, or both. To perform such an analysis for a large number of classes, we used software bertillonage,[17] with which we created a unique signature for each class of each app. Such signatures can be compared efficiently. We computed the signatures as follows:

1. We grouped fully qualified class names with the list of method names and parameters for each class into a string *S*. This string was the class's signature.
2. We classified each *S* as a specific ad library (on the basis of the previously identified ad libraries) or as non-ad library code (core code).
3. For each *S*, we generated a hash value for quick matching. To generate these values, we applied SHA1.

For each app, we obtained a set of signatures for the ad library code and core code. We compared the signatures of each ad library in a particular version of an app ($version_i$) with those in a subsequent version ($version_{i+1}$). Using this approach, for every consecutive app version, we distinguished the signatures according to the following:

- The signatures were identical in both versions (the ad libraries' APIs didn't change).
- The signatures belonged to a new ad library in $version_{i+1}$ (none of the class signatures in that library existed in $version_i$).
- The signatures in $version_i$ were deleted or removed from $version_{i+1}$.
- The signatures were updated in $version_{i+1}$. By "update," we mean the signatures in $version_i$ and $version_{i+1}$ differed (even if only one class signature differed) and were neither completely new nor completely removed.

By our definition, an ad library had been updated if any of the last three of the four previous signature changes occurred between $version_i$ and $version_{i+1}$. Although the comparison of class signatures based on software bertillonage was coarse grained (we couldn't track changes made in the implementation of a particular method), the results on the ad library updates were a lower bound. So, at a minimum, the ad libraries underwent the number of updates we identified. If we could have examined the code in a method in a class, we would have been able to identify even more ad library updates.

TABLE 1

**Frequency of the types of ad library updates in apps.**

| Type of update | No. of app versions updated |
|---|---|
| Modified | 4,470 (65.25%) |
| Added | 2,993 (43.69%) |
| Removed | 1,894 (27.64%) |

The app developers actively maintained the ad libraries: in 48.98 percent of the apps (6,850 of the 13,983 versions), the ad libraries were updated. Table 1 lists the types of ad library updates. Nearly two-thirds of the versions had an ad library modified, one-half gained a new library, and one-quarter had an ad library removed. From Table 1, we conclude that the app developers were more actively modifying their current ad libraries than adding or removing ad libraries. This highlights the importance and complexity of keeping ad libraries up to date.

In 13.75 percent (942) of the app versions with ad library updates, the APIs in the core code weren't modified (the class signatures unrelated to the ad library classes were identical). For example, View.CalCollection.SangGeon.Cauly (a calculator app) was updated seven times in 2011. However, none of these updates modified the API of its core code—all changes were related to the AdLantis ad library. This app is a good example of how a very stable app, such as a calculator, might require constant maintenance due to ad library updates.

## Update Frequency and Rationales

App developers needed to frequently update the ad libraries because the

TABLE 2

Updates of the most popular ad libraries in 2011.

| Library name | No. of updates | Rationale |
|---|---|---|
| Google Ads and AdMob | 6 | The transition to merge the two libraries. Bug fixes to improve user interaction with ads and to handle clicks. |
| Millennial Media | 9 | Improvements to video ads and click events for those ads. |
| Flurry | 8 | First released for App Circle, a new ad product. Updates to pass Google-imposed conditions for the user experience. Improving checks for ad availability. |
| AdWhirl | 7 | Updates to work properly with different ad networks. |
| Mobclix | 12 | Adding the capability of interstitial ads (ads that display before the app content loads). Preventing autoplay ads. Adding mediation (coordinating ad networks) capability for the Google/AdMob software developers kit (SDK) and the Millennial Media Android SDK. Fixing bugs related to memory management and clicks. Encrypting personal information before transmitting it. |
| YouMi | 15 | Fixing bugs that prevent ads from displaying on Android OS 2.3. Adding a lightweight initialization process to display ads. |

libraries' APIs were being updated frequently. Table 2 presents how many times the APIs of the six most popular ad libraries in the studied apps were updated and the rationales for those updates (based on information from either the changelog files in the ad library or the ad library's website). Some of the main reasons for the updates were to

- enhance interaction capabilities between ads and app users,
- integrate new types of ads (for example, video ads),
- fix memory management bugs,
- add better and more secure management of personal information, and
- fix bugs that prevented ads from displaying properly.

The advertising companies responsible for the six most popular ad libraries updated those libraries from six to 15 times that year. Such frequent updates forced the app developers to update their apps often to include an ad library's latest version.

App developers shouldn't ignore ad library maintenance because their revenues could be affected both positively and negatively. New ad libraries provide opportunities for additional income but require effort to integrate in a mobile app. Developers should choose an advertising company that has a stable ad library with minimal updates, which will reduce update costs. When app developers decide whether to add an ad library, they should perform a cost–revenue analysis.

Eventually, the software engineering community will face the problem of how to reduce the expenses and impact of ad library updates in mobile apps. More specifically, because these libraries don't contribute to app functionality, maintaining them presents different challenges compared to traditional libraries, such as

- changing an ad library API without inducing revenue loss for the app developer,
- reducing the effort of making these updates without any functional or nonfunctional field failure, and
- pushing updates to users without disturbing them.

These challenges call for research to improve the maintenance and quality assurance of ad-supported mobile apps. In addition, no studies (only informal discussions in Q&A forums or non-peer-reviewed articles) have estimated the actual cost of the expenses we discussed in the section "Update Costs." A large-scale developer survey is needed to provide a better understanding of the exact dollar values for these expenses.

## References


1. "Gartner Says Mobile App Stores Will See Annual Downloads Reach 102 Billion in 2013," Gartner, 19 Sept. 2013; www.gartner.com/newsroom/id/2592315.
2. D. Davidson and B. Livshits, "MoRePriv: Mobile OS Support for Application Personalization and Privacy," 2012; http://research.microsoft.com/pubs/163596/MSR-TR.pdf.

## ABOUT THE AUTHORS

**ISRAEL J. MOJICA RUIZ** is a software engineer at McAfee. His research interests include mobile software and empirical software analysis. Mojica received an MS in computer science from Queen's University. Contact him at israel_mojica@mcafee.com.

**THORSTEN BERGER** is a postdoctoral fellow in the University of Waterloo's Generative Software Development Lab. His research interests include model-driven development, variability modeling for software product lines and software ecosystems, variability-aware static analyses of source code, and mining software repositories. Berger received a PhD in computer science from the University of Leipzig. Contact him at tberger@gsd.uwaterloo.ca.

**MEIYAPPAN NAGAPPAN** is an assistant professor in the Rochester Institute of Technology's Department of Software Engineering. He previously was a postdoctoral fellow in the Software Analysis and Intelligence Lab at Queen's University. His research centers on using large-scale software engineering data to address stakeholders' concerns. Nagappan received a PhD in computer science from North Carolina State University. He received a best-paper award at the 2012 and 2015 International Working Conference on Mining Software Repositories. Contact him at mei@se.rit.edu; mei-nagappan.com.

**STEFFEN DIENST** is a PhD student in the University of Leipzig's Business Information Systems department. His research interests include machine-learning techniques to help monitor the operation of renewable power plants, reverse engineering, and functional programming. Dienst received an MSc in computer science from the University of Leipzig. Contact him at sdienst@informatik.uni-leipzig.de.

**BRAM ADAMS** is an assistant professor at Ecole Polytechnique de Montreal, where he heads the Maintenance, Construction, and Intelligence of Software lab. His research interests include software release engineering, software integration, software build systems, software modularity, and software maintenance. Adams received a PhD in computer science engineering from Ghent University. He was an organizer of the 1st International Workshop on Release Engineering. He's a member of IEEE. Contact him at bram.adams@polymtl.ca.

**AHMED E. HASSAN** is the Natural Sciences and Engineering Research Council of Canada / BlackBerry Software Engineering Chair at the School of Computing at Queen's University. His research interests include mining software repositories, empirical software engineering, load testing, and log mining. Hassan received a PhD in computer science from the University of Waterloo. He spearheaded the creation of the International Working Conference on Mining Software Repositories and its research community. Hassan also serves on the editorial boards of *IEEE Transactions on Software Engineering*, *Empirical Software Engineering*, and *Computing*. Contact him at ahmed@cs.queensu.ca.

3. W. Enck et al., "A Study of Android Application Security," *Proc. 20th USENIX Conf. Security* (SEC 11), 2011, p. 21.
4. M.C. Grace et al., "Unsafe Exposure Analysis of Mobile In-App Advertisements," *Proc. 5th ACM Conf. Security and Privacy in Wireless and Mobile Networks* (WISEC 12), 2012, pp. 101–112.
5. "Mobile App Advertising and Monetization Trends 2012–2017," IDC and App Annie, 2014; http://go.appannie.com/app-annie-idc-mobile-app-advertising-and-monetization-trends-2012-2017.
6. "At $11.6 Billion in Q1 2014, Internet Advertising Revenues Hit All-Time First Quarter High," Internet Advertising Bureau, 12 June 2014; http://www.iab.net/about_the_iab/recent_press_releases/press_release_archive/press_release/pr-061214.
7. "How Much Does It Cost to Develop an iPhone Application?," Stack Overflow, 2010; http://stackoverflow.com/questions/209170/how-much-does-it-cost-to-develop-an-iphone-application.
8. "The State of Enterprise Mobile Readiness 2013," AnyPresence, 2013; www.anypresence.com/blog/2013/07/11/state-enterprise-mobile-readiness-2013.
9. R. Chomko, "The Real Cost of Developing an App," *Manufacturing.net*, 30 July 2012; www.manufacturing.net/articles/2012/07/the-real-cost-of-developing-an-app.
10. "Smaato Releases Q3 2011 Mobile Metrics Report," Smaato, 29 Nov. 2011; www.smaato.com/metricsq32011.
11. K.-M. Cutler, "How Do Top Android Developers QA Test Their Apps?," TechCrunch, 2 June 2012; http://techcrunch.com/2012/06/02/android-qa-testing-quality-assurance.
12. H. Khalid et al., "What Do Mobile App Users Complain About?," *IEEE Software*, vol. 32, no. 3, 2015, pp. 70–77.
13. "Google Ads Developer Blog," Google, Apr. 2014; http://googleadsdeveloper.blogspot.com/search/label/admob_sdk.
14. I.J. Mojica, "Large-Scale Empirical Studies of Mobile Apps," MSc thesis, School of Computing, Queen's Univ., 2013.
15. S. Dienst and T. Berger, "Static Analysis of App Dependencies in Android Bytecode," tech. note, 2014; www.informatik.uni-leipzig.de/~berger/tr/2012-dienst.pdf.
16. I. Mojica et al., "A Large-Scale Empirical Study on Software Reuse in Mobile Apps," *IEEE Software*, vol. 31, no. 2, 2014, pp. 78–86.
17. J. Davies et al., "Software Bertillonage: Finding the Provenance of an Entity," *Proc. 8th Working Conf. Mining Software Repositories* (MSR 11), 2011, pp. 183–192.