Adoption of Third-party Libraries in Mobile Apps: A Case Study on Open-source Android Applications

Aidan Polese*, Safwat Hassan†, Yuan Tian*
*School of Computing, Queen's University, Canada
{aidan.polese,y.tian}@queensu.ca
†Thompson Rivers University, Canada
shassan@tru.ca

ABSTRACT

Third-party libraries are frequently adopted in open-source Android applications (apps). These libraries are essential to the Android app development ecosystem as they often provide vital functionality that would take significant development time to implement otherwise. Researchers have mainly studied the prevalence and updates of third-party libraries in Android apps. However, no prior work investigates the adoption percentages of third-party libraries in apps and how they evolve. It remains unknown whether there are any patterns in third-party libraries' adoption percentages in Android apps.

In this study, we empirically investigate the adoption of third-party libraries in 2,997 open-source Android apps over a six-year study period (2015-2020). We collected 39,882 commits from repositories hosting the target apps, and identified all changes to the adoption percentages of third-party libraries in each app. We then calculated the adoption percentage of each library in each app over specific time periods. Using the collected data, we report adoption statistics of popular libraries, propose a new taxonomy to characterize their evolutionary patterns, investigate the adoption percentages of third-party libraries across different app categories, and explore the groups of libraries that have similar release patterns and version-level adoption patterns. Our findings provide insight on third-party library adoption in open-source Android apps and thus might help researchers create tools to improve the library adoption in mobile apps.

CCS CONCEPTS

• **Software and its engineering → Software libraries and repositories**.

KEYWORDS

Android application, Third-party libraries, Mobile applications, Open-source, Adoption percentage, Empirical study

ACM Reference Format:
Aidan Polese*, Safwat Hassan†, Yuan Tian*. 2022. Adoption of Third-party Libraries in Mobile Apps: A Case Study on Open-source Android Applications. In *IEEE/ACM 9th International Conference on Mobile Software Engineering and Systems (MOBILESoft '22), May 17–24, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3524613.3527810

1 INTRODUCTION

Dependencies often make up large parts of the foundation for modern apps. Similarly, in the Android app ecosystem, third-party libraries are used in almost every popular open-source Android app project. They are also used in less popular apps. Libraries are essential to the ecosystem as they often provide key functionality or implement features that would take a lot of extra development time to implement otherwise. For instance, libraries can be used to internally organize GUI elements with view binding, load, and display ads to generate revenue for developers and allow an app to connect to social media platforms and integrate social functionality. When developing an Android app, utilizing external libraries can help alleviate development effort by using shared code and creating more fully featured and robust apps.

This study focuses on libraries and library version usage from 2015 through 2020 within open-source Android apps. We have defined third-party libraries as libraries that do not begin with *androidx*, *com.android*, *org.jetbrains.kotlin*, and *android.arch*, because direct SDK libraries are not indicative of the current library landscape - they represent developer choice from different native Android modules. We have also excluded Kotlin libraries because they represent which language a developer prefers to write in and are not necessarily directly linked to their library choices. Specifically, we collect and analyze the libraries and their associated versions used in open-source apps by examining the usage percentages of libraries, usage percentages of associated versions, commenting on release statistics, and trends present in the Android library space.

Few researchers or organizations have approached this topic before. There are sites like AppBrain[24] which provide library market share information. However, AppBrain and other similar sites only consider statistics from the current year or what they subjectively deem "useful" which is usually not fully qualified. To fill the gap, this paper hopes to create a fully formed compilation of data to examine the state of open-source Android development as of current. Analyzing the adoption of libraries and library versions in apps would help developers understand which library versions are the most stable to prevent build breaking, avoid bugs, runtime errors, ease development effort. It may also help developers explore the potential useful popular libraries widely adopted by other apps. For SE researchers, studying the market-wise adoption of libraries in Android apps may exhibit the state of the Android library landscape and the direction it is heading in, be it a dip, transition period, or something similar.

The main goal of this work is to analyze the adoption percentages of third-party libraries in Android apps and explore how popular libraries' adoption percentages evolve over time. As we would like to mine the entire history of library adoption in apps, we decided to focus on open-source android apps. Specifically, we collected a set of 2,278 open-source Android apps from three resources, i.e, F-droid [4], Android Time Machine [10], and AndroidTest [17]. We cloned the repositories of these projects and mined 39,882 changes on the adoption of third-party libraries. Based on the collected data, we identify different patterns of third-party library adoption and evolution by addressing the following three research questions:

RQ1: What are the highest adopted libraries? Given the fact that thousands of third-party libraries are adopted in open-source apps, it is necessary to identify which libraries have the highest adoption percentages, as they represent the most important functionality types that Android developers are interested in. We report the the maximum adoption percentage for each highly adopted library over all periods that are considered, showing how highly adopted each library is within in the study period. We also categorize highly adopted libraries into distinct sets. The adoption percentages of libraries may be determined by the respective functionality of each mobile app. We found that different classifications do indeed have varying levels of library category usage. For instance, *Games* apps in our target set utilize ad libraries heavily to generate revenue as they are largely free.

RQ2: How do the adoption percentages of popular libraries evolve? Intuitively, like normal software products, the adoption of third-party libraries in Android apps might not remain stable. Thus, this question aims to summarize evolutionary patterns in popular libraries' adoption percentages identified in RQ1. We grouped libraries that have similar adoption percentage change patterns and found that there are five main adoption patterns: libraries that experience an increase in overall adoption percentage, those which have overall decaying adoption percentage, libraries which have oscillating levels of adoption percentages, libraries that maintain consistent usage which do not increase or decrease, as well as libraries that experience growth in their adoption percentage, and then decay.

RQ3: What are the relationships between library version releases and the adoption percentages of libraries and their associated version patterns? In this RQ, we examine the specific library versions being adopted in selected apps. We explore the relationships between version-level adoption and the release patterns of libraries. The rationale behind this question is that rapid-releasing libraries might have clients (apps that use the library) scatted across different versions and thus affects the version-level adoption percentage and evolution patterns. Knowing such information can help both app developers and library providers to better select third-party library versions. We found that using K-Means clustering, having three clusters best represented the library version usage relationships, giving three distinct patterns.

As far as we know, this is the first study providing adoption percentages of third-party libraries using a large amount of Android apps over a long study period. Our findings provide insight for stakeholders in the Android ecosystem, and may help researchers create tools that can improve the adoption of third-party libraries in mobile apps. Our dataset contains updated fully processed 2,278 open-source apps with all changes on library adoption. We make available the data we used in this research at [3]. This could be used to verify our results as well as for follow-up research.

2 BACKGROUND

How are Android apps built using external libraries? The most common building mechanism for Android apps is Gradle [28]. When using Gradle to build an Android project, a file called the *build.gradle* is used to define Gradle tasks, build configurations, project plugins, and declare library dependencies [1]. The section labelled *dependencies* is where library dependencies and their associated versions are declared. These keywords are then followed by the dependency name and version number. The keywords that will be focused on in this study are *implementation*, *api* and *compile* as these keywords focus on app functionality rather than testing and build dependencies [1]. When the app build process starts, the dependencies are pulled from their maven repositories and then the app is built [Fig. 1].

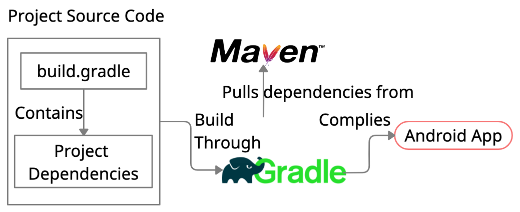

Figure 1: Gradle Building Process

How can the adoption percentages of libraries be mined? The Gradle wrapper file (*gradlew*) contains the exact Gradle configuration that the developers define and use when they develop and build their app. The *gradlew* file can be used in place of having a matching local Gradle environment to the developers of the app. The reason for this is because when first executing Gradle commands using *gradlew* [2], the necessary version of Gradle is downloaded and then used to execute commands in place of a personal installation. The reason that using a *gradlew* file is important is because extracting app library dependencies is not possible unless one is using the correct Gradle version defined by the developers.

Using *gradlew* to execute Gradle commands is the same as using a standard Gradle installation to execute Gradle commands. To view the dependencies that are used in an app, one can use *gradlew* to create a dependency tree for the app. Figure 2 shows an example of a dependency tree [5]. This tree shows all declared dependencies, their sub-dependencies as well as all associated versions. To extract this tree, one can use and run the Gradle dependencies command by using the *gradlew* file that is included in most Android app

[1]https://docs.gradle.org/current/userguide/tutorial_using_tasks.html

projects. Here is a sample dependencies command *./gradlew -q dependencies –configuration implementation*. The dependencies are pulled from their respective maven repositories and then their respective dependency versions are resolved. These dependencies are then used in generating the dependency tree. This is the basis which this study is built upon.

```
\--- org.eclipse.jgit:org.eclipse.jgit:4.9.2.201712150930-r
     +--- com.jcraft:jsch:0.1.54
     +--- com.googlecode.javaewah:JavaEWAH:1.1.6
     +--- org.apache.httpcomponents:httpclient:4.3.6
     |    +--- org.apache.httpcomponents:httpcore:4.3.3
     |    +--- commons-logging:commons-logging:1.1.3
     |    \--- commons-codec:commons-codec:1.6
     \--- org.slf4j:slf4j-api:1.7.2
```

Figure 2: Partial Gradle Dependency Tree Output

3 DATA COLLECTION

Figure 3 shows the overview of our data collection approach. The following sections describe the details of our data collection process.

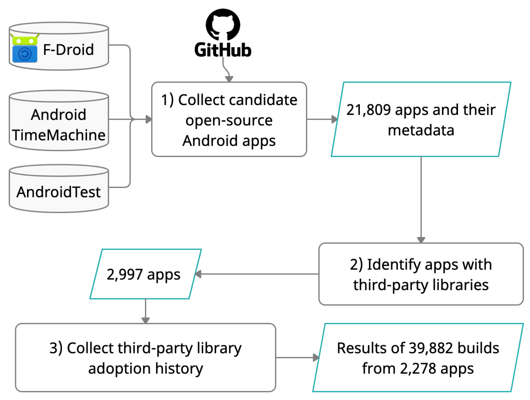
F-Droid
GitHub
Android TimeMachine
AndroidTest
1) Collect candidate open-source Android apps
21,809 apps and their metadata
2) Identify apps with third-party libraries
2,997 apps
3) Collect third-party library adoption history
Results of 39,882 builds from 2,278 apps

Figure 3: An Overview of Our Data Collection Process

3.1 Collecting open-source Android apps

As a baseline, we consider three data resources where we collected a list of open-source Android apps that are widely studied in the analysis of mobile apps. In particular, the three resources are F-Droid [4], a well-known site for hosting free and open-source Android apps, AndroidTimeMachine [10] a collection of open-source Android apps collected by Geiger et al., and AndroidTest [17] a collection of Android apps from GitHub, provided by Lin et al.

Apps on F-Droid are indexed using the app's associated package name. F-Droid pages also contain links to the apps' repositories. AndroidTimeMachine is a highly-cited Android app dataset (used in [9]). It was created by Geiger et al. to analyze the relationship between app source code and metadata. For each of the app in AndroidTimeMachine, information, such as Android app package names, repository links and metadata such as Google Play category, are provided. The third dataset, AndroidTest, was created in 2020 by Lin et al. [17] to understand the test automation culture among mobile app developers. For each app in this dataset, information, such as the app package name, number of commits made to the repo and the number of contributors are provided.

At the time of data collection in October 2021, there were 2,954 open-source Android apps (with repository URLs included) collected from F-Droid and 8,431 from AndroidTimeMachine, 12,562 from AndroidTest, forming a total of 21,809 unique candidate Android apps (package names and associated repository URLs) for us to explore.

3.2 Identifying apps with third-party libraries

Among the 21,809 apps collected in step 1, we use git to successfully clone 20,096 (92.14%) repositories. We filter out the remaining non-git supported repositories to make the following commit parsing process consistent. Next, we perform the following steps to identify eligible apps that can be analyzed in our study:

- **Removing repositories that do not contain information about the adopted libraries.** As described in Section 2, the *build.gradle* and the *gradlew* files can be used to extract the used libraries (along with their library versions) in a repository. In this work, we focus on analyzing how third-party libraries are adopted on a large scale. Hence, we remove repositories that do not contain the *build.gradle* and the *gradlew* files.
- **Removing apps that do not define any depending libraries.** As introduced in Section 2, adopted libraries are defined using three keywords *implementation*, *compile*, and *api*. Thus, we only keep app repositories that declare at least one of the three keywords used to declare third-party libraries.
- **Resolving repositories with multiple app packages.** An Android app is identified by a unique package identifier, usually in the form of *com.example.app*. Sometimes one repository can host multiple packages, e.g., one main package and another as a lite version. For those repositories, we manually select the main package name by checking the information related to the app's respective naming convention.

Table 1 summarizes the number of apps achieved after following each of the above steps. At the end of step 2, we collected a set of 2,997 projects that have specified at least one library using a specified keyword in their latest app version. We noticed that many candidate apps are filtered in step 2. However, we believe that this would provide us with a clean data set for us and future researchers to conduct research on library adoption and usage. More discussion on threats to validity introduced in the above process are presented in Section 5.

3.3 Collecting third-party library adoption history

For each of the selected 2,997 apps from step 2, we collect all commits which modify any of an app's *build.gradle* files. For each identified commit, we check out the repository at the specific commit and identify the corresponding Gradle wrapper file. We then run the Gradle wrapper file to generate the corresponding dependency tree. Next, we utilize a simple custom parser to parse the dependency

Table 1: Statistics of Apps Collected In Step 1 and Step 2

	Step 1: Collecting open-source Android apps		Step 2: Identifying apps with third-party libraries	
Data source	Number of Package Names	Number of URLs	Number of Package Names	Number of URLs
AndroidTimeMachine	8,424	8,216	418	314
AndroidTest	12,562	12,562	2,414	2,414
F-Droid	3,046	2,946	813	797
Total	21,809	23,278	3,024	2,997

tree and collect the set of third-party libraries adopted in each committed version of the app. It should be noted that the number of apps was further reduced to 2,278 as 716 apps contain commit(s) that could not be built. At the end of this step, we successfully compiled a list of 39,882 builds of 2,278 apps and we identified 3,211 unique libraries. Table 2 shows the distribution of apps by the year they were created in.

Table 2: Distribution of Apps by Year Created

Year Created	# Apps
2008-2014	175
2015	570
2016	382
2017	191
2018	534
2019	359
2020	54
2021	13

3.4 Data Statistics and Time Periods

Android apps, like other software, may become inactive after they mature. Considering the libraries adopted in those inactive app might bias our study towards older apps and thus do not reflect the common adoption practices given a specific period. Therefore, to answer RQ1-3, we only consider *active apps* as a target when analyzing the adoption of third-party libraries. In particular, at point in time t, *active apps* (t) are apps with at least a single commit to their repositories within the past 12 months of the specified time t.

Fig. 4 shows the distribution of active apps, cumulative active apps, and the average active apps over six years (from 2015-01-01 to 2021-01-01). On average, we have around 1,000 active apps per year. We have collected the commits made to the selected apps until October 2021 (when we start data collection). However, to answer three RQs, we decided not to consider the commits made in 2021 as we noticed that the year 2021 (2021-01-01 to 2021-10-31) has fewer active apps than previous years.

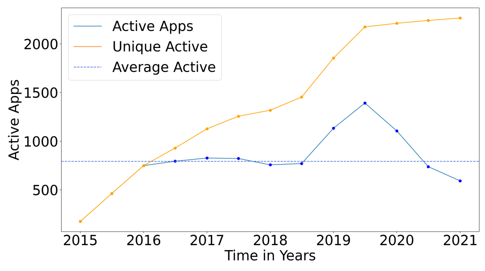

Figure 4: Active Apps from 2015 to 2021

4 EMPIRICAL STUDY AND RESULTS

In this section, we present our study in terms of three research questions. For each RQ, we discuss the motivation, approach and the obtained results.

4.1 RQ1: What are the highest adopted libraries?

Motivation: The main goal of this study is to take an empirical look at the historical Android app landscape and identify libraries that are popular among developers within the specified time frame of 2015 through 2020. Answering RQ1 would help us find important libraries in different periods and explore the possible reasons for being popular. For instance, a library might have a high adoption percentage for its novelty and broad functionalities.

Apps have different categories (e.g., games, education, and productivity). The adoption of third-party libraries can vary from one app category to another. Thus we also investigate if **there are any differences in the adoption percentages of libraries across different app categories**. Understanding the most dominantly adopted libraries in each app category can help app developers better prioritize their effort to adopt and maintain third-party libraries.

Approach: To study how widely a library L is used at a certain time t, we measured the *Adoption Percentage* ($AP_{(L,t)}$) as the ratio of the number of active apps(t) that adopt the library L to the total number of active apps(t). For instance, at specified time t, if there are 100 active apps and 20 of them are using a library L, the $AP_{(L,t)}$ is 20%.

We introduce *MAP (Maximum Adoption Percentage)* as the highest value of the adoption percentage of a third-party library over the study period. Specifically, $MAP_{(L)}$ is defined as:

$$MAP_{(L)} = max(AP_{(L,t_1)}, AP_{(L,t_2)}, \ldots AP_{(L,t_n)})$$

Where $AP(L, t_k)$ refers to the adoption percentage of library L in time period k. n refers to the total number of time periods, i.e., 12 (6 years * 2) in this study.

We use MAP to exclude the novice third-party libraries so that we can analyze libraries that have been widely adopted in our collected apps. For example, a third-party library with a MAP of 10% means that, at a specified time, 10% of the studied active apps have adopted this library.

Calculate the adoption percentage of each library: From January 1st of 2015 up to December 31st of 2020, we measured the adoption percentage of every library used in all active apps every six months (i.e., we consider time t as a span of 6-months snapshots). Finally, we identify a list of the top 10 most popular libraries (i.e., libraries with the highest adoption percentage) every six months. At the end, we identified the unique third-party libraries that are widely

Table 3: Basic Statistics of the Most Popular Third-party Libraries. MAP stands for maximum adoption percentage.

Library Category (#libs)	Description	Library Name	MAP	Provider
Ads (1)	Libraries that are used to load and display ads	com.google.android.gms:play-services-ads	9.54	Google
Analytics (1)	Allows for monitoring and collecting user engagement and interaction statistics.	com.google.android.gms:play-services-analytics	10.39	Google
Connectivity (6)	Libraries that allow for apps to connect to the internet, other web services, interact with http or provide other wireless capabilities.	com.squareup.okhttp3:okhttp	15.99	Square
		com.squareup.okhttp:okhttp	9.57	Square
		com.squareup.retrofit2:converter-gson	16.07	Square
		com.squareup.retrofit2:retrofit	18.05	Square
		com.squareup.retrofit:retrofit	6.85	Square
		com.google.firebase:firebase-messaging	10.56	Google
Logging (1)	Libraries that allow for the collection and writing of logging information	com.squareup.okhttp3:logging-interceptor	10.56	Square
Utility (7)	Libraries that allow for connect or use external api services such as Google Play or Firebase. This also includes libraries that offer backend functionality utilities which are more focused on internal workings of app.	com.google.android.gms:play-services	21.23	Google
		com.google.android.gms:play-services-base	16.91	Google
		com.google.android.gms:play-services-basement	18.98	Google
		com.google.android.gms:play-services-tasks	15.45	Google
		com.google.firebase:firebase-common	10.09	Google
		com.google.firebase:firebase-core	22.01	Google
		com.nineoldandroids:library	8.90	Individual
UI (6)	Libraries that are either used in the creation of front end UI or have front end UI elements interact with backend.	com.actionbarsherlock:actionbarsherlock	7.14	Individual
		com.google.android.material:material	46.42	Google
		com.jakewharton:butterknife	15.48	Individual
		com.github.bumptech.glide:glide	16.98	Bump Technologies
		com.github.chrisbanes.photoview:library	7.14	Baseflow
		com.squareup.picasso:picasso	15.61	Square
Data Persistence (6)	Libraries that deal with either backend I/O work or work, file operations or interactions regarding storage	com.squareup.okio:okio	15.81	Square
		commons-codec:commons-codec	9.52	Apache
		commons-io:commons-io	9.52	Apache
		org.jsoup:jsoup	7.14	Individual
		com.google.code.gson:gson	20.16	Google
		javax.inject:javax.inject	17.14	Java
Location (2)	Libraries that allow for checking user location.	com.google.android.gms:play-services-location	8.88	Google
		com.google.android.gms:play-services-maps	9.72	Google

adopted (i.e., appear in the top 10 most popular libraries at any snapshot *t*) by the studied apps over the study period.

Identify the type of top libraries: For the identified most popular libraries, we also identify their function types, e.g., ads library, logging library, etc.

Analyze category-wise library adoption: To identify the app category, we crawled the Google Play Store in Oct 2021. We considered the category provided in the sources for apps that do not exist in Google Play Store (either removed or do not publish their app on the Google Play Store). Then we identified four app categories with more than 100 apps in our collected data, i.e., *Games* (100 apps), *Tools* (284 apps), *Productivity* (153 apps), *Education* (172 apps). A total of 709 apps are considered for category-wise third-party library adoption analysis.

Next, for each app in the selected four app categories, we examine their adoption of libraries on a monthly basis. We then consider libraries with a MAP larger than 5%. There are a total of 60 libraries selected. We then manually identify their functional groups. For each app category, we calculate the percentage of apps in that category adopted at one selected library in each library category. For instance, we figured that at 2018-08-01, 7.41% of the active education apps had adopted at least one ads library. In the end, we compare the adoption preferences of four app categories.

Results: We identify 32 libraries that are widely adopted. The identified libraries belong to eight library groups, as summarized in Table 3. We find that the Google libraries are the most popular - 40.6% of most popular libraries are provided by Google. Since Google acquired Android in 2005, naturally, their proprietary libraries are the most popular. The *play-services* brand of libraries are mainly for interacting with the Google Play Store, and they also provide back-end utility for interfacing with Google APIs. Google also provides popular Firebase libraries, i.e., *firebase-common,firebase-core*, and *firebase-messaging*. Firebase is a powerful platform by Google that provides functionalities and helps with the back-end development of mobile apps (e.g., user authentication).

Followed by Google, Square libraries make up 25% of the top libraries. It is interesting to find that, Square, a company primarily known for its digital payment systems, developed five out of six most popular connectivity libraries. Among the connectivity libraries, *com.squareup.retrofit2:retrofit* has the highest maximum adoption percentage (MAP) over all the 12 periods we considered. This library is very simple and powerful when dealing with HTTP requests.

Besides the big companies mentioned above, we also notice that individual developers contribute to 4 out of the 32 most popular libraries. Three of them were created by a developer named Jake Wharton. Jake Wharton worked at CashApp and Square [25]. He has made a significant amount of contributions to both his own android projects as well as all of Square's and CashApp's Android-related repositories. He has over 120 open-source repositories, and almost all of them are related to Android.

As shown in Table 3, the most popular library is com.google.android.-material:material with a 46.42% MAP. Material library is extremely popular because it is a powerful and all-in-one component-based front-end UI library developed by Google. When developers begin to create an Android app, they have a high probability of using this library to design and create GUI elements.

It is also noteworthy that the four most predominant types of libraries are ones that focus on utilities (21.9% of the 32 libraries), UI, connectivity, and data persistence. This does make sense as Android app development is largely centered around creating functional interfaces and communicating between devices and the internet, and thus needs UI and connectivity support via third-party libraries. Utility libraries often interact with Firebase and Google Play, which are common platforms to utilize when creating apps. There are also popular utility libraries and data persistence libraries that provide back-end support, e.g., provide more efficient threading and file I/O processes which are necessary processes among all apps.

To compare the preference of third-party library adoption across the four selected app categories, we performed a Scott-Knott ESD test (V3.0) [23] on the monthly adoption percentage of each app category on each identified library category. We only consider the adoption data from 2016 to 2020 as we observed that some app categories did not have a high adoption percentage of third-party libraries in 2015. Note that, as we increase the scope of target libraries to 60 libraries, we identified two additional library groups upon the eight mentioned in Table 3, i.e., *Security* libraries and *Compatibility* libraries. Security libraries provide security-related services to secure Android app and provide cryptographic APIs. Compatibility libraries are used to support apps in their transit from one SDK version to another.

Table 4 shows the results of tests on all library categories, where different colors are used to differentiate cluster groups returned by a Scott-Knott ESD test. **We observed that among the four app categories, Education apps are more likely to adopt third-party libraries, and Tool apps are less likely to adopt libraries.** For eight out of the ten considered library categories, Education apps have the highest mean adoption percentage. Tools are interesting in terms of not having a preference in adopting third-party libraries other than UI libraries. One potential reason is that Tool apps are mainly focused on creating new functionalities, likely performing a niche function, so they do not generally use many common third-party libraries.

All app categories focus on good UI, and thus we observe that a large portion of them have adopted UI libraries. Besides UI libraries, different app categories show various interests over library categories. Education apps focus on Utility, Data Persistence, and Connectivity. Often, Education apps need to store a lot of data locally and receive data from online resources. Thus they prefer to adopt libraries that can support general functions, data persistence, and connection. Game apps emphasize ads libraries, as they need to earn money mainly from advertisements. Connectivity libraries also play an important role because modern games require a connection to an online server to function. Productivity apps largely favor UI, Utility, and Data Persistence libraries. The reason behind this might be that Productivity apps are often the apps like exercise trackers and study reminders that have highly personalized data, so keeping track of the user locally and appealing to look at and use would make an effective app in this category.

Answer to RQ1: 32 libraries spread across 8 functional categories are identified as the most popular ones by checking the top-10 libraries each in every 6 month time frame across the six year time period. Google libraries make a 40.6% of the 32 libraries, followed by Square, and individual contributors. These libraries can achieve a maximum adoption percentage as high as 46.42% and as low as 6.85%. Utility and UI libraries are mostly frequently adopted. Education apps are most likely to adopt all kinds of libraries, and in contrast, Tool apps are less likely to adopt libraries. Games have the highest emphasis on ads libraries because ad viewing is the primary way modern mobile games make money. Productivity apps also focus on data persistence libraries as they are often focused on personalized single user tasks.

4.2 RQ2: How do the adoption percentages of popular libraries evolve?

Motivation: RQ1 analyzes the max adoption percentage for popular libraries and their adoption in different app categories. In RQ2, we further investigate the dynamics of adoption percentages and study how the adoption percentages of popular libraries evolve. Identifying groups of libraries with similar evolution of adoption percentages would help us explore potential reasons behind the evolution patterns.

Approach: We take as input the adoption percentages of the 32 most popular libraries in 12 considered periods (from 2015 to the end of 2020, every six months) from RQ1. The first two authors manually examine each of the 32 curves reflecting the evolution of the adoption percentage of 32 libraries and categorize them into different evolution trends.

Results: From the collected data, we identify five evolution patterns: Libraries where their adoption percentage is growing, decaying, oscillating while decaying, steadily used libraries and libraries that exhibit a growing then decaying arc. Next, we describe each pattern as follows.

Pattern 1 - Growing: Growing libraries are libraries which go from a low adoption percentage to a higher adoption percentage. Figure 5 shows the sample curves belong to this category. Growing libraries among the 32 most popular libraries are as follows:

- io.reactivex.rxjava2:rxjava
- io.reactivex.rxjava2:rxandroid
- com.squareup.retrofit2:retrofit
- com.squareup.retrofit2:converter-gson
- com.squareup.okhttp3:okhttp
- com.squareup.okhttp3:logging-interceptor
- com.google.android.material:material
- com.github.bumptech.glide:glide

Table 4: Results of SK test across different lib categories. GMAP stands for app group-wise mean adoption percentage. The darkness of each cell represents the identified group(s). For instance, Education apps adopt significantly more "Analytics" libraries than other three app categories, followed by Games apps and Productivity apps, and then Tools apps.

Lib Category	App Category (GMAP)	App Category (GMAP)	App Category (GMAP)	App Category (GMAP)
Ads	Game (9.98)	Education (5.26)	Productivity (3.17)	Tools (2.29)
Analytics	Education (14.86)	Games (11.74)	Productivity (11.14)	Tools (3.66)
Connectivity	Education (32.41)	Games (21.76)	Productivity (18.81)	Tools (18.01)
Logging	Education (16.32)	Productivity (12.33)	Tools (12.03)	Games (9.98)
Utility	Education (42.49)	Productivity (39.19)	Games (22.78)	Tools (17.51)
UI	Education (37.46)	Tools (33.34)	Productivity (29.80)	Games (17.40)
Data Persistence	Education (42.24)	Productivity (29.65)	Tools (26.85)	Games (13.52)
Location	Education (17.21)	Productivity (5.57)	Games (4.63)	Tools (3.82)
Compatibility	Productivity (5.56)	Games (4.33)	Tools (2.69)	Education (2.24)
Security	Education (9.60)	Games (6.43)	Productivity (5.61)	Tools (2.23)

We manually examine the functionalities and release notes of the above libraries and identify two potential reasons to explain their growth over the years. First, these libraries are easy to use and could reduce the code complexity introduced in implementing relevant services in the mobile app. Secondly, these libraries often introduce essential features that are not always easy to achieve using native Android libraries. An example of growing library being *io.reactivex.rxjava2:rxjava.* This library enables powerful threading paradigms to be implemented easily. Without the help of this library, threading within Android can be quite a hassle.

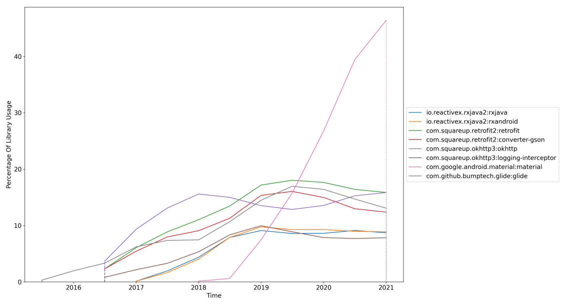

Figure 5: Growing Libraries

Pattern 2 - Decaying: Decaying libraries are libraries which go from a high adoption percentage to a lower adoption percentage. Figure 6 shows the sample curves belong to this category. The decaying libraries are as follows:

- commons-io:commons-io
- commons-codec:commons-codec
- com.squareup.retrofit:retrofit
- com.squareup.okhttp:okhttp
- com.nineoldandroids:library
- com.google.android.gms:play-services-analytics
- com.google.android.gms:play-services
- com.github.chrisbanes.photoview:library
- com.actionbarsherlock:actionbarsherlock

We manually examine the homepage of each decaying library and Android SDK release notes. We believe that the main reason behind the curves is that Android SDK implemented third-party libraries' main features, which ends up in the libraries no longer

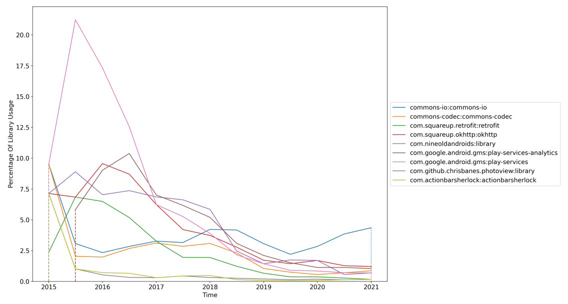

Figure 6: Decaying Libraries

fulfilling a niche. Hence, those libraries begin to die off because many developers have started using a new native way to implement code. A secondary reason is that Android sometimes makes drastic changes or overhauls to native systems, making libraries no longer needed or incompatible with the new system. Hence, these libraries either adapt or die out.

Pattern 3 - Oscillating: Oscillating libraries are libraries that have periods of a higher adoption percentage and periods of lower adoption percentages with a fluctuating pattern but eventually die out. Figure 7 shows the sample curve belong to this category. We find one oscillating library, i.e.,

- javax.inject:javax.inject

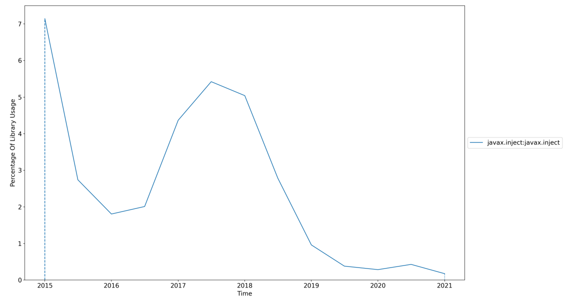

Figure 7: Oscillating Libraries

The above library is only used when there are shifts in technology, or a wave of apps are created that use a library. This evolution pattern indicates app development as a whole, e.g., how newly created apps use a slowly dying technique.

Pattern 4 - Steadily Used: Steadily used libraries are libraries that have a fairly stable adoption percentage over the whole study period. Figure 8 shows the sample curves belong to this category. The steady libraries are as follows:

- com.google.code.gson:gson
- com.google.android.gms:play-services-ads
- com.google.android.gms:play-services-maps
- com.google.android.gms:play-services-location
- org.jsoup:jsoup

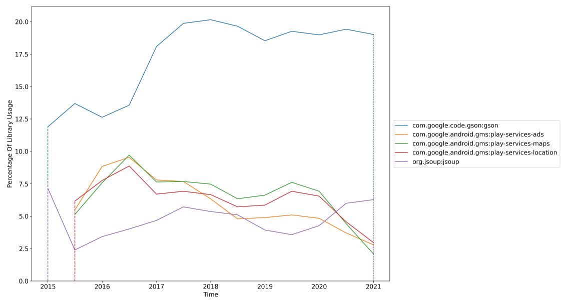

Figure 8: Steady Libraries

We observe that the above libraries that are consistently used are the ones that are not subject to major changes and fulfill a role that other services cannot fill. For instance, play-service libraries are steadily used as they serve the main function integral to Android app availability as a whole. The stable libraries could also be the ones that are novel and cannot be dethroned or made obsolete by a change in first party libraries.

Pattern 5 - Growing then Decaying: Libraries that grow and then decay are classified as libraries that start at a low popularity, rise to a peak of popularity and then decay down to a lower popularity. Figure 9 shows the sample curves belong to this category. The growing then decaying libraries are as follows:

- com.squareup.picasso:picasso
- com.squareup.okio:okio
- com.jakewharton:butterknife
- com.google.firebase:firebase-messaging
- com.google.firebase:firebase-core
- com.google.firebase:firebase-common
- com.google.android.gms:play-services-tasks
- com.google.android.gms:play-services-basement
- com.google.android.gms:play-services-base

We observe that the libraries in this category are often integrated into Android SDK or are replaced by newer versions. This frequently happens with play-services libraries as they shift towards Firebase libraries or are packaged into different play-service libraries. This happens less frequently with *core* or *common* libraries as they are generally incorporated into Android SDK. Those libraries might be falling as new technology is introduced, namely, Firebase libraries soaking up play-services libraries.

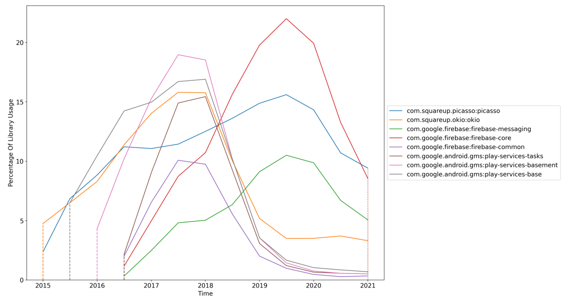

Figure 9: Growing then Decaying Libraries

Answer to RQ2: We observed five main evolution patterns of library max adoption percentages: growing, decay, oscillate, stable, grow and decay. Growing libraries often implement a feature that is easy to use and significantly reduces code complexity otherwise. Libraries tend to decay and die when a new Android SDK implements a similar feature or dramatically changes the Android architecture leveraged by the libraries.

4.3 RQ3: What are the relationships between library version releases and the adoption percentages of libraries and their associated version patterns?

Motivation: The purpose of this RQ is to determine if a library's release patterns correlate with how developers choose which library version to use. Release patterns for libraries can change very heavily from developer to developer and from company to company. The disparity in release patterns might cause adoption trends to shift as a whole.

Approach: When libraries release, they generally follow a release naming scheme of *x.y.z* where *x* is the major release version, *y* is the minor release version, and *z* is the patch release version. Therefore, library releases can be classified as one of three forms, a major release of form *x.0.0*, a minor release of form *x.y.0*, or a patch release in the form *x.y.z*. Occasionally, a developer might have a version similar to *x.y.z-alpha* where *-alpha* is an alpha, beta, rc, etc. version, which will be considered a patch release. As a visual example, *com.google.code.gson:gson* has multiple versions that get adopted at different times with different popularities (i.e., percentage of apps that use each library version) [Fig. 10]

Fig. 10 shows that some of the popular *gson* versions are minor releases or major releases. There are also a fair few patch releases adopted but at very low percentages. To quantify such information, we introduce a metric, namely *MVAP (Maximum Version Adoption Percentage)* as the highest value of the adoption percentage of a third-party library version within apps adopting the library over the study period. MVAP is defined as:

$$MVAP(L_i) = max(VAP_{(L_i,t_1)}, VAP_{(L_i,t_2)}, \ldots VAP_{(L_i,t_k)})$$

Where $VAP(L_i, t_k)$ refers to the adoption percentage of version i of library L in time period k within active apps adopting library L.

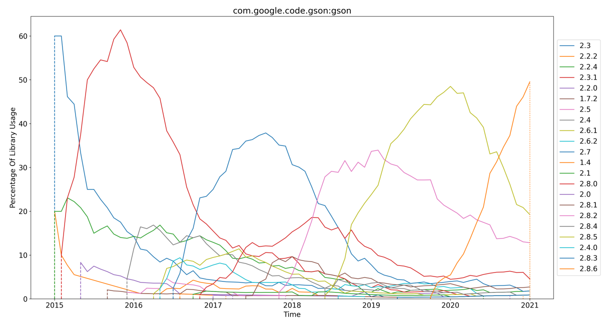

Figure 10: Version Adoption Percentages of Library com.google.code.gson:gson.

$VAP(L_i, t_k)$ is calculated using:

$$\frac{\#active\ apps\ adopting\ version\ i\ of\ library\ L\ in\ period\ k}{\#active\ apps\ adopting\ library\ L\ in\ period\ k}$$

MVAP represents the highest relative popularity of a specific version among all available versions of a popular library.

Next, we define the following seven metrics to capture the characteristics of a library based on its release patterns and the MVAPs of their releases:

- We measure **medMajorMVAP** as the median value from a set containing the MVAPs of all *major* version releases of a library.
- We measure **medMinorMVAP** as the median value from a set containing the MVAPs of all *minor* version releases of a library.
- We measure **medPatchMVAP** as the median value from a set containing the MVAPs of all *patch* releases of a library.
- We measure **medReleaseUpTime** as the *median value in months*, from a set of all the counts of how long a release existed before a subsequent release.
- We measure **ptMajorRelease** as the percentage of all releases for a library that are a *major* release.
- We measure **ptMinorRelease** as the percentage of all releases for a library that are a *minor* release.
- We measure **ptPatchRelease** as the percentage of all releases for a library that are a *patch* release.

Leveraging the above seven metrics, we perform a K-means clustering [11] on the most popular 32 libraries in our studied apps identified in RQ1. The purpose of clustering is to identify libraries with similar releases and release adoption patterns. We use the silhouette validation [20] technique to determine the best cluster number.

Results: By applying silhouette validation, we find that the best cluster number is 3, with an average silhouette score of 0.5279. We show the representative statistics (centers) of the three identified clusters in Table 5.

For cluster 1, there is a pretty even spread of adoption between major, minor and patch releases, although the edge is given to minor releases. This behavior might be expected because of how many minor version releases there are compared to major releases. This suggests that likely, most feature updates are released in minor releases. These styles of libraries also tend to have a large amount of patch releases which seem to release almost every month. This style of library is well supported and constantly being worked on.

For cluster 2, there is a heavy emphasis on major release adoption. A fair number of releases do tend to be major releases, so this adoption percentage does make sense. There are also quite a few patch releases. This style of library looks to be well supported but developers tend to only set-and-forget the major releases.

Cluster 3 contains a set of libraries with only minor releases being adopted. The low representative *ptMajorRelease* value and the high *ptMinorRelease* value indicate that those libraries have rare major releases as major releases could occur only internally, and only the minor releases with bug fixes are published.

Answer to RQ3: Using a set of 7 metrics, we quantitatively cluster three sets of similar libraries based on their associated version patterns: 1) libraries tend to have pretty fleshed out adoption percentages which might indicate a well supported library, 2) libraries with clients that grab major releases and follow a set-and-forget adoption pattern, 3) libraries where major releases are probably kept internal as minor releases are by far the most adopted versions of these libraries.

5 THREATS TO VALIDITY

Threats to internal validity: Concerning factors that can affect our results. Our method for collecting open-source apps and their associated third-party library adoption history, may fail to collect the actual adoption of libraries in a commit of an Android app repository. However, we manually checked our code and results on a sample popular app in our dataset and achieve an accuracy of 100% while comparing to human annotation.

We include Google libraries when analyzing the most popular libraries adopted in active apps during different time periods, though Google owns Android, because they are not provided by the Android SDK directly. Like all libraries, they do make changes based on the changes offered in Android SDK but they have evolved mostly separate from the official Android SDK and provide optional functionality, which is why they are worth including. Moreover, literature studies have analysed Google libraries in Android apps. For instance, prior study on ad libraries [6] reported that google ads is the most popular third-party ad library in Google Play Store free-to-download apps. Excluding Google libraries may lead to slightly different results for RQ1-RQ3.

Threats to external validity: Concerning the generalization of our findings. Our analysis is limited to open-source apps as we aim to parse the commit history of each target android app. Thus, the results might not be applicable to commercial apps. To make sure we can get an accurate set of adopted libraries for each app in a specific time period, we filtered out apps that do not contain *build.gradle* and the Gradle wrapper file. This design removes a large portion of our initial set of 21,809 candidate apps, and thus the reported adoption percentage might not represent the whole open-source Android app ecosystem. However, we believe that such design is necessary, as modern and high-quality open-source apps often

Table 5: Representative Statistics of Three Identified Clusters of Libraries.

cluster	medMajorMVAP	medMinorMVAP	medPatchMVAP	medReleaseUpTime	ptMajorRelease	ptMinorRelease	ptPatchRelease
1	10.69	15.08	8.86	1.21	10.48	40.71	48.81
2	93.92	18.48	18	2.25	30.12	6.57	63.31
3	0	100	35	2.5	1.19	65.18	33.63

follow the standard Gradle build mechanism to define and maintain library dependencies. Moreover, the resulting dataset contains 2,278 apps, which is comparable with state-of-the-art dataset used for library analysis that rely on the mining of repository of apps [21].

6 IMPLICATIONS

Developer Adoption Patterns: We speculated that a possible reason a developer might adopt a library is due to a few factors, ease of implementation, powerful and ease of use. The libraries that make it into the top 10 most libraries in a time frame are those which can be easily implemented into code and are able to function with as few lines as possible. So, if a developer would want their app to perform well, they might consider having their library function with as few lines as possible with one clear intended use outside of the Android SDK if they wish to be widely adopted. Jake Wharton is a prime example of this [25]. It might be worth looking into how versions are adopted as it would help developers to understand when to implement and release big feature versions that will stick around.

Insights for Platform Providers: If a company like Square or google would want to increase their market share, they might consider looking into the results of the study. By examining the functions of the libraries with the highest adoption percentages, they might be able to dedicate some development efforts into creating libraries that serve similar purposes to further increase their influence. A company like Square could take notice of other libraries that contribute to the connectivity library space and consider implementing similar libraries as their brand has a good reputation in the space that could be further expanded upon.

7 RELATED WORK

Existing related work has a large focus on the effects of 3rd party libraries in a privacy and security based setting. Many of these works focus on closed sourced apps. A detailed summary of research on third-party libraries in Android Apps can be found in a recent intensive literature review conducted by Zhan et al. [27].

When analyzing third-party libraries used in mobile apps, many researchers mainly identify apps using third-party libraries by reverse engineering APKs of free Android apps and analyze the impact of libraries on the reporting of downstream applications, such as clone detection [18], malware detection [14], and repackaged apps detection [14]. Li et al. [14] also reported that the top used library is *com.google.ads*, which is used by 17% of their decompiled 1.5 million Android apps. Besides, many studies have proposed tools [15, 16, 19, 22, 28] to identify libraries in Android apps (i.e., APKs), or to identify the appearance of malicious libraries [29], and vulnerable library version [26]. Different from their work, we focus on analyzing the adoption of third-party libraries and how the popularity of libraries evolve. Our analysis requires a set of snapshots of open-source apps, while they take as input a set of APKs collected at one timestamp.

Another line of research focuses on analyzing the updates of third-party libraries in mobile apps. For example, Ahasanuzzaman et al. [6, 7] studied the adoption of third-party advertisement libraries in popular mobile apps. Derr et al. [8] performed an empirical study on third-party library updatability over 1,264,118 Android apps. They found that most of the libraries can be upgraded without modifying the source code. A following study by Huang et al. [12] found that the prior reported updatability rate by Derr et al. is under real conditions overestimated by a factor of 1.57–2.06. Salza et al. [21] then enhances the state of the art in this direction. They empirically investigated when, how, and why mobile developers update third-party libraries in their code by mining the evolution history of 2,752 open-source apps and with a survey of 73 developers. They found that developers rarely update the version of third-party libraries they used in their mobile apps. This aligns with a prior study by Kula et al. [13]. They also found that most of library updates are done with the aim of avoiding bug propagation or making an app compatible with Android SDK releases. Differing from them, we analyze library adoption percentages. However, we also argue that Android releases have a huge impact on the adoption of third-party libraries and their over-time popularity (ref. Section 4.2).

8 CONCLUSION AND FUTURE WORK

In this paper, we created a snapshot of the current open-source Android library landscape through a large-scale app and library collection. Through examining library adoption percentages over six years, we were able to measure some of the effective practices of library development and release cycles. We identified 32 most popular libraries adopted in our studied apps over six years and used five patterns to characterize their evolution pattern. Our findings provide information for both developers and researchers for their future adoption of libraries and analysis of library adoption in mobile apps. In the future, we would like to expand the scope of our study by considering closed apps and relaxing the thresholds used to pick target popular libraries.

ACKNOWLEDGEMENT

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), [funding reference number: RGPIN-2019-05071].

REFERENCES

[1] Add build dependencies nbsp;: nbsp; android developers. https://developer.android.com/studio/build/dependencies?buildsystem=ndk-build.

[2] The gradle wrapper. https://docs.gradle.org/current/userguide/gradle_wrapper.html.

[3] Replicate package. https://figshare.com/s/5623c7b2b3d15f4800b0.

[4] F-droid - free and open source android app repository. https://f-droid.org/, 2010.

[5] Viewing and debugging dependencies. https://docs.gradle.org/current/userguide/viewing_debugging_dependencies, 2022.

[6] M. Ahasanuzzaman, S. Hassan, C.-P. Bezemer, and A. E. Hassan. A longitudinal study of popular ad libraries in the Google Play Store. *Empirical Software Engineering*, 25(1):824–858, 2020.

[7] M. Ahasanuzzaman, S. Hassan, and A. E. Hassan. Studying ad library integration strategies of top free-to-download apps. *IEEE Transactions on Software Engineering*, 48(2):209–224, 2022.

[8] E. Derr, S. Bugiel, S. Fahl, Y. Acar, and M. Backes. Keep me updated: An empirical study of third-party library updatability on android. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2187–2200, 2017.

[9] F.-X. Geiger, I. Malavolta, L. Pascarella, D. Di Nucci, and A. Bacchelli. Android-timemachine. https://androidtimemachine.github.io/publications/.

[10] F.-X. Geiger, I. Malavolta, L. Pascarella, F. Palomba, D. D. Nucci, and A. Bacchelli. A graph-based dataset of commit history of real-world android apps. In *Proceedings of the 15th International Conference on Mining Software Repositories, MSR*, New York, NY, May 2018. ACM. doi: https://doi.org/10.1145/3196398.3196460.

[11] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1):100–108, 1979.

[12] J. Huang, N. Borges, S. Bugiel, and M. Backes. Up-to-crash: Evaluating third-party library updatability on android. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 15–30. IEEE, 2019.

[13] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue. Do developers update their library dependencies? *Empirical Software Engineering*, 23(1):384–417, 2018.

[14] L. Li, T. Riom, T. F. Bissyandé, H. Wang, J. Klein, et al. Revisiting the impact of common libraries for android-related investigations. *Journal of Systems and Software*, 154:157–175, 2019.

[15] M. Li, W. Wang, P. Wang, S. Wang, D. Wu, J. Liu, R. Xue, and W. Huo. Libd: Scalable and precise third-party library detection in android markets. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 335–346. IEEE, 2017.

[16] M. Li, P. Wang, W. Wang, S. Wang, D. Wu, J. Liu, R. Xue, W. Huo, and W. Zou. Large-scale third-party library detection in android markets. *IEEE Transactions on Software Engineering*, 46(9):981–1003, 2018.

[17] J.-W. Lin, N. Salehnamadi, and S. Malek. Test automation in open-source android apps. *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020. doi: 10.1145/3324884.3416623.

[18] M. Linares-Vásquez, A. Holtzhauer, C. Bernal-Cárdenas, and D. Poshyvanyk. Revisiting android reuse studies in the context of code obfuscation and library usages. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 242–251, 2014.

[19] Z. Ma, H. Wang, Y. Guo, and X. Chen. Libradar: fast and accurate detection of third-party libraries in android apps. In *Proceedings of the 38th international conference on software engineering companion*, pages 653–656, 2016.

[20] P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. ISSN 0377-0427. doi: https://doi.org/10.1016/0377-0427(87)90125-7.

[21] P. Salza, F. Palomba, D. Di Nucci, A. De Lucia, and F. Ferrucci. Third-party libraries in mobile apps. *Empirical Software Engineering*, 25(3):2341–2377, 2020.

[22] W. Tang, P. Luo, J. Fu, and D. Zhang. Libdx: A cross-platform and accurate system to detect third-party libraries in binary code. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 104–115. IEEE, 2020.

[23] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto. The impact of automated parameter optimization on defect prediction models. *IEEE Transactions on Software Engineering*, 45(7):683–711, 2018.

[24] M. Vogelzang. Monetize, advertise and analyze android apps. https://www.appbrain.com/, 2010.

[25] J. Wharton. Jakewharton - overview. https://github.com/JakeWharton.

[26] X. Zhan, L. Fan, S. Chen, F. Wu, T. Liu, X. Luo, and Y. Liu. Atvhunter: Reliable version detection of third-party libraries for vulnerability identification in android applications. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1695–1707. IEEE, 2021.

[27] X. Zhan, T. Liu, L. Fan, L. Li, S. Chen, X. Luo, and Y. Liu. Research on third-party libraries in android apps: A taxonomy and systematic literature review. *IEEE Transactions on Software Engineering*, (01):1–1, 2021.

[28] Y. Zhang, J. Dai, X. Zhang, S. Huang, Z. Yang, M. Yang, and H. Chen. Detecting third-party libraries in android applications with high precision and recall. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 141–152. IEEE, 2018.

[29] Z. Zhang, W. Diao, C. Hu, S. Guo, C. Zuo, and L. Li. An empirical study of potentially malicious third-party libraries in android apps. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 144–154, 2020.