Towards the Adoption and Adaptation of the AndroidX Library: An Empirical Study

Jiacheng Li*, Kerui Huang*, Sinan Wang, Yepang Liu†
Southern University of Science and Technology, Shenzhen, China
{12012705,12012427,wangsn}@mail.sustech.edu.cn, liuyp1@sustech.edu.cn

Abstract—AndroidX is an official Android library that enables backward compatibility for Android APIs used by various apps. It is the successor of the Android Support library since Android 9.0. Since then, many apps that originally relied on Android Support needed to be adapted to use AndroidX. However, for app developers, such a migration task can be challenging and error-prone. Yet, there is no systematic study on the migration status of real-world Android apps or the issues that may arise during the migration to AndroidX. To fill this knowledge gap, we conducted the first comprehensive study concerning the adoption and adaptation of the AndroidX library. In this study, we inspected 171 Stack Overflow posts about AndroidX and identified common categories of issues that can occur when adapting apps to use AndroidX, as well as the causes. We also examined the trend of these issues in recent five years to assess their impact over time. Then, we investigated the utilization status of both Android Support and AndroidX libraries in 15,334 top commercial apps and 2,470 open-source apps. Finally, we developed an algorithm that utilizes cosine similarity to identify Java class mappings between Android Support and AndroidX. The algorithm allows us to recover an additional mapping of 579 Java class pairs, which can supplement the official class mapping. Our study reveals the following key findings: (1) Improper API usage, misuse of dependencies, and incorrect build configurations are common challenges faced by developers during the adoption of AndroidX; (2) Around 13.3% commercial apps and 17% open-source apps still rely on the Android Support library; (3) Inconsistencies are observed between the API migration table in Android Studio and the API mapping table provided in AndroidX documentation. (4) On average, the migrated AndroidX classes achieve a 0.978 code similarity score with the original Android Support classes.

Keywords–Android; AndroidX; Android Support Library; API Migration

1. Introduction

Android is the most popular open-source mobile operating system over the world. Due to its continuous evolution, Android undergoes frequent updates and releases. Consequently, Android app developers need to deal with compatibility issues across various OS versions [1], [2] and customized devices [3]. This presents a crucial technical challenge when building and maintaining Android apps.

The Android Support library is a cross-version Android compatibility library [4]. It was introduced by Google [5] to address the API compatibility issue on Android devices. The Android Support library focuses on back porting features for new SDK releases. It provides wrappers for a subset of interfaces (or types) on different SDK versions. Instead of directly invoking Android SDK APIs, apps can call their Android Support wrappers. As such, apps developed for a new Android version can be run on previous versions, without extensive modification on the app program [2].

The Android Support library was also evolving. Initially, it only contained fundamental components [6] such as `Fragment`, `ViewPager`, `RecyclerView`, etc. These components made it easy to develop high-quality GUIs. Over the time, the Android Support library continues to iterate [4], adding new components and APIs, and keeping in sync with the Android platform. During the evolution, developer's feedback indicated that the growth of the Android Support library has become confusing in some naming rules [7]. As an example, there are components and packages named "`v7`" while the minimal SDK level support is 14. To fix these confusions, in 2018, Google released AndroidX [7] to replace the original Android Support library [4]. Now, AndroidX is constantly being iterated, updated, and improved in terms of functionality, performance, and compatibility [7], [8], with Google regarding it as the future direction of Android development.

AndroidX was introduced to address issues with the old Android Support library. Therefore, developers are encouraged to transition from the Android Support library to AndroidX. It's been over four years since AndroidX was launched, however, no prior study has systematically explored the adoption and adaptation of AndroidX, which motivates our work.

In this study, we analyzed 171 high-quality Stack Overflow posts to identify six common AndroidX issues, categorizing them by occurrence during app building and runtime. Building issues, notably related to dependencies and build tools, prevailed over runtime errors, which stemmed primarily from improper API usage.

Issue trends from 2018 to 2023 revealed a decrease in build tool-related bugs and an increase in API usage errors, suggesting developers' growing familiarity with AndroidX, yet occasional mistakes when handling specific APIs.

Besides, our study extensively examined 15,334 commercial apps from Google Play and 2,470 open-source apps from F-Droid, uncovering that approximately 13.3% of commercial apps and 17% of open-source apps still rely on the Android Support library.

Additionally, we compared mapping tables for Support-to-

*The two authors equally contributed to this work.
†Corresponding author.

2693-9177/23/$31.00 ©2023 IEEE
DOI 10.1109/QRS60937.2023.00048

AndroidX migration, highlighting inconsistencies between Android Studio's automatic migration tool and the official AndroidX documentation, which could lead to migration failures or confusion for developers.

Furthermore, we applied cosine similarity and Soot Jimple IR to examine code similarity between Support and AndroidX libraries. On average, migrated AndroidX classes exhibited a 0.978 code similarity with their original Android Support counterparts, and this algorithm expanded the official mapping table with 579 additional entries.

In summary, our study makes the following contributions:

- We performed the first empirical investigation of AndroidX issues. Through an in-depth analysis of real problems from Stack Overflow, we have compiled a comprehensive taxonomy of AndroidX-related issues. Our findings can provide valuable guidance to developers and shed light on potential future research in common issue detection techniques.
- We analyzed a substantial number of Android apps to investigate their utilization of the AndroidX and Support libraries. We mined the migration status of these apps from Support to AndroidX. Our results can provide useful statistical insights into the AndroidX ecosystem.
- We devised a cosine similarity-based algorithm for measuring class similarity and have employed this algorithm to identify an additional 579 mapping entries for the official mapping table of AndroidX and Support APIs.
- We released the source code, dataset, and results in a GitHub repository[1] to facilitate follow-up studies.

2. Background

To ease presentation, we will use *Support* and *AndroidX* to refer the Android Support library and the AndroidX library throughout this paper. This section provides the necessary background knowledge, including the mechanisms and approaches used by the two libraries to ensure compatibility across different Android OS versions, and the migration process from Support to AndroidX.

The code snippet in Listing 1 is an example, which comes from the source code of an AndroidX class `ActivityCompat`[9]. It is a representative API in Support/AndroidX to address API backward-compatibility issue.

```
public boolean isLaunchedFromBubble(Activity activity) {
  if (Build.VERSION.SDK_INT >= 31) {
    //activity.isLaunchedFromBubble()
    return Api31Impl.isLaunchedFromBubble(activity);
  } else if (Build.VERSION.SDK_INT == 30) {
    return Api30Impl.getDisplay(activity) != null &&
        Api30Impl.getDisplay(activity).getDisplayId() !=
        Display.DEFAULT_DISPLAY;
  } else if (Build.VERSION.SDK_INT == 29) {
    return activity.getWindowManager().getDefaultDisplay()
        != null && activity.getWindowManager().
        getDefaultDisplay().getDisplayId() != Display.
        DEFAULT_DISPLAY;
  }
  return false;
}
```

Listing 1: An example of AndroidX API

[1]https://github.com/huangkr03/AndroidX-Empirical-Research

The Android API `Activity.isLaunchedFromBubble()` is to determine whether the activity was launched from a bubble (a feature that allows certain activities to be presented in a floating bubble on top of other apps). It is only accessible in SDK version 31 and above. For earlier versions like 29 or 30, maintaining extra code is required to avoid crashes when invoking this API. The Support and AndroidX classes do this job to unify their API signatures, as shown in the code snippet, thus developers can use the unified API and avoid struggle with multiple API representations in different Android versions.

To migrate from Support to AndroidX, the app developers should modify all dependencies from Support to their corresponding AndroidX artifacts and updating all references in the host app's code. As an example, the Support artifact `com.android.support:support-compat` is mapped to the AndroidX artifact `androidx.core.app`, and the Support class `android.support.v4.app.ActivityCompat` is mapped to `androidx.core.app.ActivityCompat`. Android provides a official migration tutorial [10], in which Android Studio can automatically migrate the app to AndroidX with appropriate settings. Nevertheless, app developers may still encounter migration issues at times, which require manual adjustment of dependency and reference classes. To aid migration, developers can refer to the official mapping table [11], [12]. However, as we will show, there are some omissions in this mapping table.

3. Empirical Study and Results

In this section, we present our empirical study in terms of three research questions. For each RQ, we discuss its motivation, our studying method and the obtained results. Figure 1 shows the overview of our empirical study, which concerns the following three research questions:

- *RQ1 (Issue types and causes) What are the common types of issues when adopting and adapting the AndroidX library, and what are their causes?*
- *RQ2 (API usage) What is the current utilization and migration status of AndroidX APIs in top Android apps?*
- *RQ3 (Class mapping) What can we learn from the official Support-to-AndroidX mapping table? Are there any omissions?*

3.1 RQ1. What are the common types of issues when adopting and adapting the AndroidX library, and what are their causes?

Motivation. Examining developer concerns with AndroidX provides insights into common obstacles. Understanding prevalent issue types during adoption guides enhancements for smoother migration. Analyzing recurring problems helps pinpoint key challenges and develop effective solutions.

We collected Stack Overflow posts discussing AndroidX issues, capitalizing on its popularity among developers as a platform for problem-solving. Stack Overflow data has been widely employed in various software engineering research studies. [13], [14], [15].

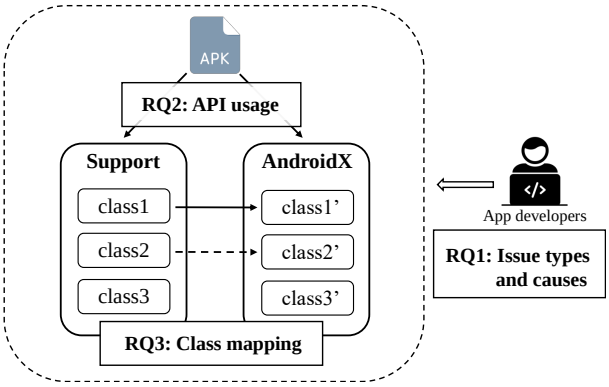

Figure 1: Overview of our empirical study

TABLE I: Numbers of Stack Overflow posts about AndroidX

Year	2018	2019	2020	2021	2022	2023
#posts (all)	324	1,101	593	227	165	51
#posts (filtered)	30	67	21	32	15	6

Data collection. To collect posts related to AndroidX, we utilized the Stack Overflow REST API [16]. We performed a search using the "AndroidX" tag as a filter, which resulted in 2,461 posts. The distribution of these posts of date of creation across different years is shown in Table I (the "all" row).

The numbers of posts reveal a clear pattern in the distribution over the years. There was a notable peak in 2019, followed by a gradual decrease in subsequent years. This pattern can be attributed to the release of AndroidX in the middle of 2018. After its release, there was a surge in the adoption and usage of AndroidX by developers, resulting in a significant number of posts in the following year. As time went on, developers became more familiar with AndroidX and its features and they can refer to existing posts for guidance, leading to a decrease in the number of new posts created over time.

To gain more understanding, we performed manual analysis. However, analyzing posts is a time-consuming task, and it is important to prioritize newer issues while avoid being overwhelmed by older ones. Therefore, for posts from 2018, 2019, and 2020, we selected only the top 20% of posts based on their view counts, ensuring a reasonable number of posts for manual analysis. We further filtered the posts based on the following criteria: first, the issues discussed in the posts should be bugs related to AndroidX, as we wanted to specifically focus on topics related to this library; second, the posted questions should have accepted answers, indicating their solutions or explanations were confirmed. With the above process, we obtained 171 high-quality Stack Overflow posts related to AndroidX. Table I also shows the yearly distribution of posts after filtering.

Analysis Method. We followed an open coding procedure [17] to categorize the issues discussed in each post. Two authors of this paper independently analyzed the posts, including their title, body, accepted answers, and comments. After each round of analysis and categorization, the two authors compared and discussed their results, and made adjustment when necessary. If conflicting opinions occurred during this process, all co-authors of this paper participated in thorough deliberation to reach a conclusive resolution. Finally, consensus on posts categorization was reached after five rounds of iteration.

Results. The collected posts can be broadly classified into two main categories: *host app bugs* and *AndroidX library bugs*. Host app bugs are primarily caused by incorrect implementation of the Android app by the developers. On the other hand, AndroidX library bugs stem from issues within the AndroidX library itself. These library bugs commonly occur during the evolution of AndroidX versions, and developers often face challenges in finding effective solutions. As a result, developers tend to raise issues on the Google Issue Tracker [18] and wait for a fixing version of AndroidX libraries, an example post is #66578828.

> ***Finding 1:*** AndroidX issues not only can be caused by developers' problematic implementation of apps, but also by bugs in the AndroidX library itself.

We found host app bugs have more complicate reasons. In our categorization, host app bugs can be further classified into six subcategories based on their causes.

3.1.1 Host app bugs categorization: Figure 2 provides a summary of all the identified bugs. The six bug causes for host app bugs are described as below:

- **Reference to wrong class.** This type of bug occurs when the client references the wrong AndroidX class in their code. A typical symptom of this bug is that the client receives an error message that says "*classes could not be found*". This type of bug mainly occurs when the client is trying to migrate from Support to AndroidX. For example, in post #51617186, the auto-migration process changed the app dependency from Support to AndroidX, but it missed some client code references. As a result, the user's reference class could not be found. Another typical cause of this bug is that the client is confused by some AndroidX namespaces. For example, in post #57050888, the client should have used the class `androidx.viewpager.widget.ViewPager`, while they wrongly referenced `androidx.core.view.ViewPager` instead.
- **Improper usage of API.** This type of bug occurs when the client use some AndroidX API improperly. Typical symptom is that client has wrong implementation logic or wrong XML configurations. Typical post is #57837514, in which client has wrong implementation logic. And in post #53458821, client use the wrong XML configuration which cause UI not working.
- **Dependency misuse.** To utilize AndroidX library, developers are required to configure their dependencies and specify the versions. This type of bug is related to app dependency configuration. Client referenced to some classes but missed its dependency or use the wrong dependency, the typical error message is *"failed to resolve"*. For example, in post #54809410, developers missed necessary dependency.

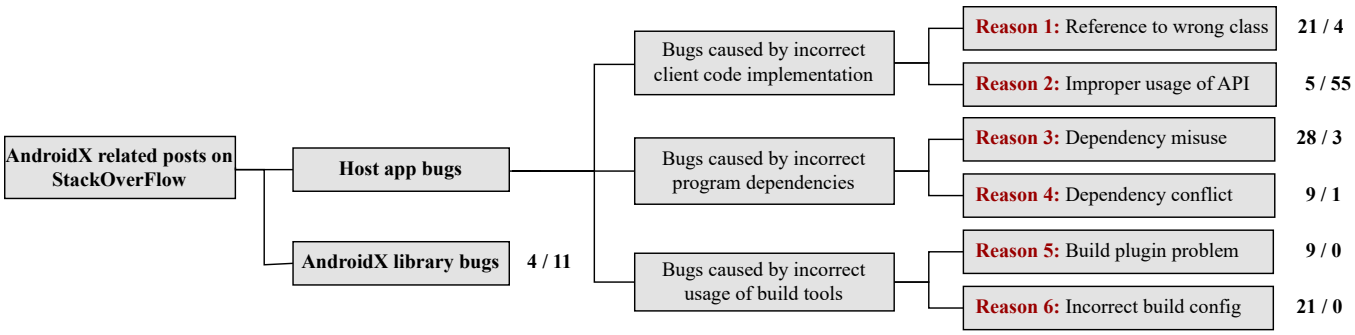

Figure 2: Categorization results of Stack Overflow posts about AndroidX, right side of each cause category represent the count of posts, displayed as: *# build issue / # runtime issue*

- **Dependency conflict.** This type of bugs indicate apps' dependencies have conflicts. For example, in post #70550502, some dependencies are pulling different versions of the `androidx.lifecycle:lifecycle-common-java8` artifact, which cause conflicts and give the error message *"duplicate class"*. Also, we noticed that some conflicts occur when developers trying to use Support and AndroidX at the same time, as discussed in post #51918301.
- **Build plugin problem.** This type of bug is occurred because some build plugin problems. Typically, migrating to AndroidX needs a build plugin Jettifier [19], which can migrate Support-dependent libraries to instead rely on the equivalent AndroidX packages. However, Jettifier may fail to transform in some cases, as discussed in post #53484988, the client shall manually set Jettifier blacklist to ignore specific libraries.
- **Incorrect build config.** Mistakes can happen during migration or utilization configuration. As in the post #53268865, developer should set *android.useAndroidX=true* and *android.enableJetifier=true*. Another typical problem relates to incorrect `compileSdkVersion`. SDK version is required to be above 28. However, in post #51291407, this requirement is not met.

> ***Finding 2:*** Build issues occur more frequently (56.7%) than runtime issues (43.3%). The primary causes of build issues are *Reference to wrong class*, *dependency misuse*, and *Incorrect build config*, while runtime issues are mainly caused by *Improper usage of APIs*.

3.1.2 The trends of AndroidX issues: Figure 3 gives the yearly percentage distribution for our collected posts. The Y-axis value represents the percentage of this category bugs of the yearly total count. We have the following observations:

- Issues of Reasons 1, 5 and 6 decreased over the years. These issues commonly happen during the migration from Support to AndroidX, developers doesn't fully change all class references, and wrongly use build plugins or set wrong build configuration. The decreasing trend in these issues suggests that, as the adoption of AndroidX has become widespread, developers have gradually completed the migration, resulting in fewer such issues being reported.
- The percentage of issues related to Reason 2 has exhibited an upward trend over time. This category of issues attribute to improper usage of AndroidX APIs. It suggests that developers shift their focus towards utilizing AndroidX itself. Consequently, an increasing percentage of posts are centered around the usage of AndroidX APIs.
- The percentage of issues associated with Reasons 3 & 4 does not exhibit a discernible trend. However, it consistently maintains a proportion of approximately 30% over the course of these years. This consistent proportion suggests that the identified problem continues to perplex developers.

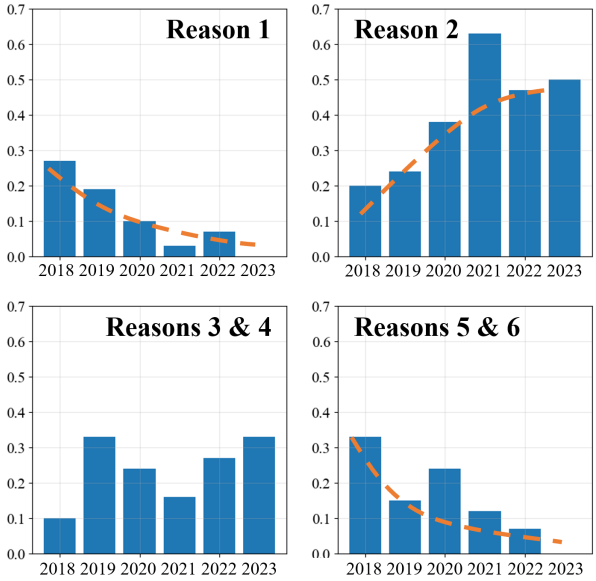

Figure 3: AndroidX issue trends from 2018 to 2023

> ***Finding 3:*** The percentages of AndroidX issues caused by *Reference to wrong class*, *Build plugin problem* and *Incorrect build config* have decreased over time. On the other hand, the percentage of issues stemming from *Improper usage of API* has increased.

3.1.3 Discussion: *AndroidX library bugs* present a distinct category on their own and accounts 8.8%. These kinds of bugs often lack straightforward solutions, leading to prolonged troubleshooting periods. We remain the further exploration of bugs within the AndroidX library itself as our future work.

From the issue counts by category, the *Reference to wrong class* category accounts for 14.6% of the total bugs. Our observations revealed that in some cases, automated migration tools failed to correctly change all class references made by developers. Additionally, we also noticed instances where developers struggled with the mapping relationship from Support classes to AndroidX classes. These findings suggest the possibility of inaccuracies or incompleteness in the auto-migration tool and the mapping table from AndroidX to Support, which will be subject to further investigation in our RQ3.

The dependency related (Reasons 3 & 4) and build related (Reasons 5 & 6) bugs constituted a significant portion of the developers' bugs, accounting for 30.0% and 17.5%, respectively. And they consistently maintains a high proportion of during during the observed years. These findings have implications for the development of auto-build configuration and dependency detection tools, which could assist developers in resolving such issues.

We found that *Improper usage of API* represents 35.1% of all the issues, and its prevalence has been consistently increasing in recent years. While the causes of these bugs differ, we noticed recurring issues with popular APIs like `navigation`, `fragment`, and `viewpager`. Investigating these common wrong implementation patterns could offer developers guidance in API usage and help enhance official API design. This aspect remains for future work.

3.2 RQ2. What is the current utilization and migration status of AndroidX APIs in top Android apps?

Motivation. The analysis of AndroidX and Support API utilization patterns in the latest top apps holds immense value in providing key insights into prevailing best practices and trends within the Android app development industry. Gaining an understanding of the migration status among the top apps can offer valuable knowledge. This includes identifying the proportion of apps that have already completed the migration process, those currently in the process, and those that have yet to initiate migration. It can guide developers in deciding whether to migrate their own apps to AndroidX or continue utilizing Support. Therefore, this RQ aims at mining and uncovering these invaluable data points, shedding light on the adoption and usage of AndroidX among developers and providing practical guidance for the Android app development community.

Data Collection. AppBrain [20] provides ranking information for the latest Google Play apps. Our data collection process involved crawling the top 500 most prominent APKs from each category available on AppBrain. To obtain the corresponding APK files for these apps, we utilized the AndroidZoo dataset [21], [22], an ever-expanding collection of Android apps, in which 15,334 apps were successfully crawled. Furthermore,

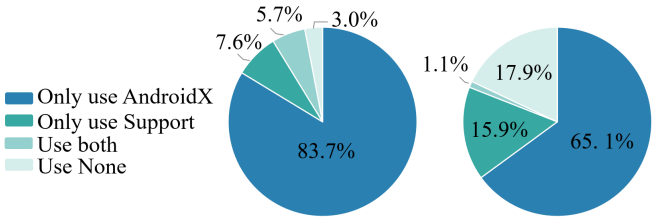

Figure 4: Migration status of apps on Google Play (left) and F-Droid (right)

we explored F-Droid [23], a curated catalogue of FOSS (Free and Open Source Software) apps specifically designed for the Android platform. Our crawling efforts encompassed all the apps released on F-Droid after 2019, the year after AndroidX was released, resulting in 2,470 apps.

Analysis Method. To extract the API usages from APK files, we utilize the static analysis framework Soot [24]. Soot enables us to efficiently process APK files by converting their code into an intermediate representation (IR), Jimple. This structured representation facilitates our program analysis.

We leveraged Soot to extract all application classes, fields, and methods. We considered an API "utilized" if it is invoked from the client code (excluding official libraries). Then, we conducted a thorough scan of the client code, identifying utilized APIs whose namespaces start with `androidx.*` or `android.support.*`. By following these outlined steps, we successfully identify utilized AndroidX and Support APIs within the APK files.

Result. We use the utilizing situation to reflect the migrating status of an app, to see weather it utilizes only AndroidX, only Support, both AndroidX and Support, or use none. We draw the pie graphs for the apps on Google Play and F-Droid, respectively, in Figure 4. It shows that a large portion of apps have completely adopted AndroidX. However, there are a proportion of apps still utilizing Support.

> ***Finding 4:*** Around 13.3% commercial apps and 17% open-source apps are still using the Android Support library.

Notice that the proportion of open-source apps not using Support and AndroidX (17.9%) is greater than that of commercial apps (3.0%), and the adoption of AndroidX in open-source apps (65.1%) is less than that in commercial apps (83.7%). A possible reason is, commercial apps often benefit from dedicated development teams or access to professional developers with ample resources and expertise. Consequently, they are more likely to prioritize compatibility with a wide range of devices while also keeping up-to-date with the latest libraries and tools. In contrast, open-source projects often involve a diverse group of contributors with varying levels of experience. This can result in compatibility considerations being overlooked and can slow down the adoption of newer libraries in these projects.

TABLE II: Collected data source and count

Data Source	JAR	AAR	Support	AndroidX
Google Maven	28	515	121	478
Other repository	50	6		

TABLE III: Numbers of APIs in the dataset

	#Class	#Outmost Class	#Method	#Field
Support	6,938	2,325	54,217	24,502
AndroidX	66,105	30,766	332,486	136,873

3.3 RQ3. What can we learn from the official Support-to-AndroidX mapping table? Are there any omissions?

Motivation. There are two kinds of mapping provided by Google from Support to AndroidX. The first one is Google's class mappings table [12] (referred to as WEB mapping) and the second one is Android Studio's automated migration mapping [25] (referred to as IDE mapping). The former assists with manual migration, while the latter enables automatic migration. In RQ1, based on the analysis of collected posts, we revealed that developers encounter challenges during the migration from Support to AndroidX, instances of automatic migration failures were observed, and developers expressed confusion regarding the mapping relation between the two libraries. Furthermore, Figure 4 indicates that a proportion of apps continue to utilize the Support library. These motivate us to investigate both mappings in detail. By undertaking this investigation, we aim to discern any discrepancies or disparities between the two mappings and evaluate their overall completeness.

Data Collection. We implement Selenium based crawler Python script to crawl the artifact information and obtain the download URLs for each artifacts of Support and AndroidX from the Maven repository of Google [26]. For each artifact, we would find its latest release version. If there was no such version, we chose its latest alpha version. We favored AAR file for each artifact. If there was no AAR file, we turned to find JAR file. If both AAR and JAR files were missing, we would store the artifact names and try manually search the files from other public Maven repository [27]. Table II shows the our final dataset of JAR/AAR files from Google Maven and other repository, and the number of files related to Support library or AndroidX library.

Result. We analyzed the crawled files with Soot to extract the containing APIs. Table III shows the numbers of APIs among our collected artifacts. We divide our analysis result into 4 sections:

3.3.1 Dependency between AndroidX and Support: It showed that in a few AndroidX related artifacts (8 artifacts), both Support classes and AndroidX classes are packaged, in which 327 AndroidX classes have dependency on the Support classes. Upon further examination, we observed that certain AndroidX classes reuse code from the Support classes to implement specific functions. For instance, in the artifact `androidx.browser:browser:1.5.0`, the class `CustomTabsCallback` has method `newSession()` to create a new session through an instance of `ICustomTabsService`[28], which is a Support class.

> ***Finding 5:*** A modest subset of AndroidX classes, comprising a total of 327 instances, maintain dependencies on the Android Support library.

3.3.2 Difference between WEB and IDE mappings: The WEB mapping is on the Google AndroidX website [12], developers can directly access it. IDE mapping is obtained from the source code of Android Studio, Android Studio has the function *Migrate to AndroidX*, we analyzed the source code and locate the implementation logic for this in class `MigrateToAndroidxAction`, and the migration process is based on a mapping file `migration.xml`, which we regard as IDE mapping. Upon analyzing this XML file and comparing it to the WEB mapping, we have found the former contains 259 class mappings, 93 package mappings, and 155 dependency mappings. In comparison, these 259 class mappings and 93 package mappings essentially cover the original 1,569 mapping entries of WEB mapping. However, there are a total of 64 distinct mappings. Among them, 3 mappings have different target classes for migration, 17 mappings are present in IDE mapping but not in WEB mapping, and 44 mappings are present in WEB mapping but not in IDE mapping. A small portion of them is displayed in Table IV.

> ***Finding 6:*** There are variations in the mapping relationships between the official web mapping table and the automatic mapping table in the Android Studio IDE. Some mappings have different target classes, while others are exclusively present in either the IDE mapping or the WEB mapping.

3.3.3 Analysis of WEB class mappings: WEB mapping has 1,549 class mappings from Support to AndroidX (excluding inner classes and anonymous classes). We gained some insights from the mapping table. For example, there are 786 mapping entries that merely convert the prefix `android.support` in the package names into `androidx`. There are 610 mappings that modify, add or delete certain words in the package names; and there are 153 mappings that do not target `androidx`. We have observed that within these 153 mapping entries, certain Support classes remained unchanged before and after migration, such as `android.support.v4.media.*`. There were some Support classes being mapped into package names starting with `com.google.android`, rather than `androidx`. For instance, `android.support.design.*` became `com.google.android.material.*`.

> ***Finding 7:*** Not all Support libraries are migrated to the `androidx` packages. Some of them maintain their original package names, and some have changed their package names starting with `com.google.android`.

TABLE IV: Differences between the IDE mapping and the WEB mapping (partial)

Support Class	AndroidX Class (IDE Mapping)	AndroidX Class (WEB Mapping)
android.support.v4.os.ResultReceiver	android.support.v4.os.ResultReceiver	androidx.core.os.ResultReceiver
android.support.test.InstrumentationRegistry	androidx.test.platform.app.InstrumentationRegistry	androidx.test.InstrumentationRegistry
android.support.test.runner.AndroidJUnit4	androidx.test.ext.junit.runners.AndroidJUnit4	androidx.test.runner.AndroidJUnit4
android.support.design.math.MathUtils	com.google.android.material.math.MathUtils	(missing)
android.support.constraint.helper.Flow	androidx.constraintlayout.helper.widget.Flow	(missing)
android.support.v4.media.AudioAttributesCompatApi21	androidx.media.AudioAttributesCompatApi21	(missing)
android.support.v7.widget.PositionMap	androidx.recyclerview.widget.PositionMap	(missing)
android.support.design.animation.AnimationUtils	(missing)	com.google.android.material.animation.AnimationUtils
android.support.design.widget.SnackbarContentLayout	(missing)	com.google.android.material.snackbar.SnackbarContentLayout
android.support.media2.BaseRemoteMediaPlayerConnector	(missing)	android.support.media2.BaseRemoteMediaPlayerConnector

Algorithm 1: Cosine Similarity Analysis

Input: Two Jimple class C_0 and C_1
Output: Cosine similarity score S

Function `splitWords`(C)
  Use regex to remove package names;
  return C.`split ('\W+')`;
Function `termFrequencyAnalysis`(L)
  $vector \leftarrow$ empty dictionary;
  foreach $word$ ***in*** L **do**
    if $word$ ***is not in*** $frequency$ **then**
      $vector[word] \leftarrow 1$;
    else
      $vector[word] \leftarrow vector[word] + 1$;
    end
  end
  return $vector$;
Function `expandDimension`($vector1, vector2$)
  $wordSet \leftarrow vector1.keyset \cup vector2.keyset$;
  foreach $word$ ***in*** $wordSet$ **do**
    if $word$ ***is not in*** $vector1$ **then**
      $vector1[word] \leftarrow 0$;
    if $word$ ***is not in*** $vector2$ **then**
      $vector2[word] \leftarrow 0$;
  end
  return $vector1, vector2$;
Function `calculateSimilarity`($vec1, vec2$)
  $similarity = \frac{\sum_{i=1}^{n} vec1_i \cdot vec2_i}{\sqrt{\sum_{i=1}^{n} vec1_i^2} \cdot \sqrt{\sum_{i=1}^{n} vec2_i^2}}$;
  return $similarity$;
$L_0, L_1 \leftarrow$ `splitWords`(C_0, C_1);
$A, B \leftarrow$ `termFrequencyAnalysis`(L_0, L_1);
$A, B \leftarrow$ `expandDimension`(A, B);
$S \leftarrow$ `calculateSimilarity`(A, B);
return S;

Next, we aim to utilize a similarity algorithm to calculate the similarity of the existing mappings, in order to examine what insights we can gain from the code before and after the migration. The similarity algorithm we have chosen is the cosine similarity algorithm, which is commonly used to determine the similarity between two vectors in a high-dimensional space [29]. It calculates the cosine of the angle between the two vectors, indicating how close they are related. It is widely utilized in various fields such as information retrieval, text mining, and recommendation systems. Its scale ranges from 0 to 1, where 0 represents orthogonality or decorrelation, and values in between indicate moderate similarity or dissimilarity. In text matching scenarios, the attribute vectors A and B typically represent the term frequency vectors of the documents. Cosine similarity can be seen as a method to normalize document length during the comparison process. The formula for cosine similarity is:

$$Similarity(A, B) = \frac{\sum_{i=1}^{n} A_i \cdot B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Where A_i and B_i are components of vector A and B.

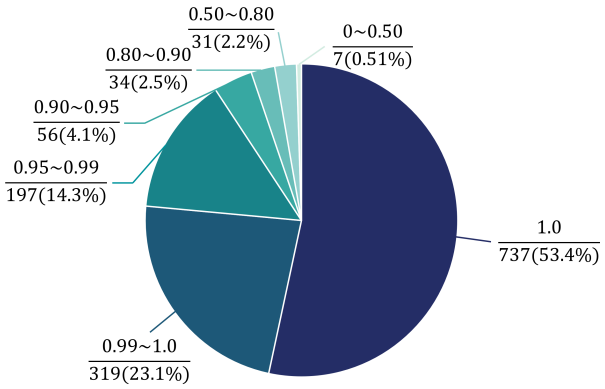

Figure 5: Cosine similarity of WEB mapping

The process of obtaining code similarity using cosine similarity is shown in Algorithm 1. We utilized this algorithm to compute the code similarity of WEB mapping. Our initial step is to employ Soot to decompile the AAR files and obtain the corresponding Jimple class representations. Then we can calculate the similarity score between two classes. By employing this approach, we calculated the similarity scores of the Support classes and their corresponding AndroidX classes listed in the WEB mapping. We visualized the results of the computation in Figure 5.

From the illustration, it is evident that more than half of the classes remain the same code before and after the migration. Approximately 75% of the classes demonstrate a code similarity score of 0.99 or higher before and after migration. Furthermore, 91% of the classes exhibit a code similarity of over 0.95 after class migration. Additionally, the majority of classes in Support remain virtually unchanged before and after

migration, with only a small portion (approximately 5% of the classes) undergoing significant modifications.

> ***Finding 8:*** Most AndroidX classes exhibit a high degree of similarity before and after migration, with only a small number of classes undergoing significant modifications.

3.3.4 Reconstructing missing mappings: The mapping table provided by Android Developers [12] contains 1,549 mappings from Support to AndroidX. However, through Table III, we have found 2,325 outmost classes in Support, indicating that some Support classes have not been officially mapped. We intend to investigate whether these classes have been migrated. Therefore, we utilized the cosine similarity algorithm for matching unmapped Support classes. Specifically, we investigated whether the mapped Support classes have their corresponding AndroidX classes. We attempt to find the most fitting AndroidX class to serve as the mapping for each Support class that lacks a mapping relationship. The procedure was as follow:

1) **Decompilation and preprocessing**: Employing Soot, we decompile all Support classes as well as AndroidX classes to obtain their Jimple IRs. We then merge the code of inner classes into the outer classes. Note that the order of merging is not important, as the cosine similarity algorithm does not take it into account.
2) **Iteratively searching for the best match**: For each Jimple IR of the Support classes, we iterate through all of AndroidX classes and use Algorithm 1 to find the one with the highest similarity score.
3) **Manual validation**: For each Support class, we manually review whether its best matching AndroidX class indeed establishes a mapping relationship. Those classes that fail to establish such a relationship are eliminated.

Finally, we have identified a total of 579 additional mapping relationships that are not covered by the official web class mapping. Each of these mapping entries has undergone meticulous manual review to ensure their utmost accuracy. After that, there are only approximately 197 Support classes that do not have a corresponding mapping relationship. As a result of our work, we have significantly increased the mapping coverage from 66.62% to 91.53%.

> ***Finding 9:*** The WEB mapping covers approximately 70% of the Support classes, yet there are still some omissions. We have discovered an additional 579 mapping entries to supplement the WEB mapping.

The 579 mappings we have recovered exhibit a high degree of similarity to the WEB mapping in terms of their naming pattern. The majority of these mappings involve replacing `android. support` with `androidx` and making adjustments to the length and content of package names. Additionally, there are some mappings that even involve changes to the class names, such as `android.support.media2.IMediaSession2` becoming `androidx.media2.IMediaSession`. Table V shows a partial of additional mapping entries that we have found for supplementing the WEB mapping. The whole reconstructed mappings can be found on our data publicity repository. We have already submit a issue about the supplementary mappings on Android Issue Tracker [30]. At the time of paper submission, the issue has not yet received reply from the Android maintainers.

4. Threats to validity

Posts trends bias: Because we only choose the top 20% of the posts in year 2018, 2019, 2020, it is important to acknowledge that this filtering process may not accurately capture the true evolution trend of AndroidX adoption, as the distribution of post categories in the top 20% might not matches the distribution below the top 20%. It's also worth noting that this approach might not necessarily reflect the most viewed posts, as developers can refer to previous posts. Unfortunately, obtaining view time data for all users was not feasible, and we had to resort to this alternative method.

Generalizability of APK dataset: The study acknowledges the potential issue of generalizeability concerning the APK dataset used. It is crucial to recognize that the dataset may not cover the entire Android app ecosystem, and some variations may exist in apps that were not considered in this research. The authors encourage further research to validate the findings with diverse APK datasets to enhance the generalizability of the results.

Non-exhaustive collection JAR/AAR files: Another concern is related to the collection of JAR/AAR files concerning AndroidX and Support source code. It was observed that a few JAR/AAR files could not be sourced from the official Google Maven repository or any other place, and few source files downloaded from Google Maven are corrupt and can not be used for further analysis. These limitation might impact the comprehensiveness of the dataset, but we make our best efforts to obtain as many relevant files as possible from reliable sources.

5. Related Work

McDonnell et al. [31] conducted an empirical study about Android API adoption. Their study aimed at understanding the co-evolution behavior of Android APIs and dependent Android apps. It revealed that outdated API references in client apps are prevalent, and the time taken to adopt new API versions is longer for fast-evolving APIs. The paper also explores the defect-proneness of API usage adaptation code. Wei et al. [32], [33] studied the fragmentation of the Android ecosystem, which causes compatibility issues for app developers. The main reason for Android fragmentation is the various device models and OS versions. They conducted a study on real-world issues, and proposed FicFinder, a tool for automatic detection of compatibility issues in Android apps via static code analysis. Xia et al. [34] also investigates the challenges of API compatibility issues faced by Android developers. They explored how developers handle these issues and proposed a tool called RAPID to automate the identification of addressed

TABLE V: Supplement to the WEB mapping (partial)

Support Class	AndroidX Class
android.support.constraint.ConstraintAttribute	androidx.constraintlayout.widget.ConstraintAttribute
android.support.constraint.StateSet	androidx.constraintlayout.widget.StateSet
android.support.wearable.complications.ProviderUpdateRequester	androidx.wear.complications.ProviderUpdateRequester
android.support.v4.media.AudioAttributesImplBase	androidx.media.AudioAttributesImplBase
android.support.multidex.BuildConfig	androidx.multidex.BuildConfig
android.support.multidex.instrumentation.BuildConfig	androidx.multidex.instrumentation.BuildConfig
android.support.media2.IMediaSession2	androidx.media2.IMediaSession

compatibility issues. The study reveals that a significant number of developers do not provide alternative implementations for incompatible API invocations. However, when Google provides replacement recommendations, developers are more likely to offer alternative solutions. Wang et al. [13] conducted an empirical study on Android runtime permission issues (ARP). They manually analyzed Stack Overflow posts and open-source projects to identify 11 prevalent types of ARP issues. They investigated the manifestations, prevalence, potential fixes, and impact of these issues on the Android ecosystem.

Besides Android ecosystem, researchers have also conducted empirical studies for software compatibility issues on the other platforms. Zheng et al. [35] examines the evolution of Python framework APIs and the resulting compatibility issues in client programs. They analyzed 288 releases of six popular Python frameworks and 5,538 open-source projects built on them. They also identified common strategies used by developers to fix compatibility issues. Marcel et al. [36] conducted an empirical study on Swift programming language, in which they analyzed 59,156 Stack Overflow questions and interviewed 12 Swift developers. They found developers' questions about Swift centered on basic language elements and the toolset (compiler, Xcode, libraries).

These papers reveal the common procedures for conducting empirical research about API evolution and compatibility issues. While Support and AndroidX are widely utilized as official libraries to address compatibility, there is a lack of systematic research focused on the Android Support and AndroidX libraries. Our study examined the distribution and migration status of both libraries in top apps, analyzing the common issue types faced by developers when adopting and adapting AndroidX, and try to establish a complete mapping relationship between the Support library and corresponding components in the AndroidX library.

6. Conclusion

In this work, we conducted a comprehensive analysis of the adoption and adaptation of the AndroidX library. First, we built a taxonomy of AndroidX issues from 171 Stack Overflow posts and investigated their reasons and trends. Besides, we investigated 15,334 apps from Google Play and 2,470 apps from F-Droid, through which we gained the AndroidX migration status of current Android apps. Finally, we have compared the class similarity between the original Support Library and the new AndroidX Library, and created a supplementary class mapping, addressing certain omissions in the official mappings. Our research contributes a comprehensive understanding of AndroidX, developers' challenges of using the library, migration status, and class mapping from Support, which provides guidance and knowledge to Android practitioners and researchers.

Acknowledgement

This work is supported by the Guangdong Basic and Applied Basic Research Fund (Grant No. 2021A1515011562).

References

[1] M. E. Joorabchi, A. Mesbah, and P. Kruchten, "Real challenges in mobile app development," in *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, 2013, pp. 15–24.

[2] D. He, L. Li, L. Wang, H. Zheng, G. Li, and J. Xue, "Understanding and detecting evolution-induced compatibility issues in android apps," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 167–177.

[3] L. Wei, Y. Liu, and S.-C. Cheung, "Pivot: learning api-device correlations to facilitate android compatibility issue detection," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 878–888.

[4] Android Developers, "AndroidX Overview," https://developer.android.com/jetpack/androidx, 2022.

[5] Google, "Support Library Introduction," https://developer.android.com/topic/libraries/support-library, May 2020.

[6] Android Developers, "Support Library Revision Archive," https://developer.android.com/topic/libraries/support-library/rev-archive, 2021.

[7] Alan, K. Kam, and L. Bergstrom, "Hello world, androidx," https://android-developers.googleblog.com/2018/05/hello-world-androidx.html, 2018.

[8] Android Developers, "AndroidX versions," https://developer.android.com/jetpack/androidx/versions, 2023.

[9] AndroidX, "ActivityCompat," https://github.com/androidx/androidx/blob/androidx-main/core/core/src/main/java/androidx/core/app/ActivityCompat.java, 2023.

[10] Android Developers, "Migrate to AndroidX," https://developer.android.com/jetpack/androidx/migrate, 2023.

[11] Android Developers, "Support Library Artifact Mappings," https://developer.android.com/jetpack/androidx/migrate/artifact-mappings, 2023.
[12] Android Developers, "Support Library Class Mappings," https://developer.android.com/jetpack/androidx/migrate/class-mappings, 2023.
[13] Y. Wang, Y. Wang, S. Wang, Y. Liu, C. Xu, S.-C. Cheung, H. Yu, and Z. Zhu, "Runtime permission issues in android apps: Taxonomy, practices, and ways forward," *IEEE Transactions on Software Engineering*, vol. 49, no. 1, pp. 185–210, 2023.
[14] M. Linares-Vásquez, G. Bavota, M. Di Penta, R. Oliveto, and D. Poshyvanyk, "How do api changes trigger stack overflow discussions? a study on the android sdk," in *Proceedings of the 22nd International Conference on Program Comprehension*, ser. ICPC 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 83–94.
[15] M. Duijn, A. Kucera, and A. Bacchelli, "Quality questions need quality code: Classifying code fragments on stack overflow," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015, pp. 410–413.
[16] Stack exchange, "2023," https://api.stackexchange.com/docs.
[17] S. Lewis, "Qualitative inquiry and research design: Choosing among five approaches," *Health Promotion Practice*, vol. 16, no. 4, pp. 473–475, 2015.
[18] Google, "Google Issue Tracker," https://issuetracker.google.com/, 2023.
[19] Google, "Jetifier," https://developer.android.com/tools/jetifier, 2023.
[20] AppBrain, "AppBrain: Android app market & app discovery," https://www.appbrain.com/.
[21] AndroZoo, "AndroZoo," https://androzoo.uni.lu/.
[22] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16. New York, NY, USA: ACM, 2016, pp. 468–471.
[23] F-Droid, "F-Droid," https://f-droid.org/, 2023.
[24] Soot, "Soot - A framework for analyzing and transforming Java and Android applications," https://github.com/soot-oss/soot.
[25] Android Developers, "AndroidX migration using Android Studio," https://developer.android.com/jetpack/androidx/migrate#migrate_an_existing_project_using_android_studio, 2023.
[26] Google, "Google Maven Repository," https://maven.google.com, 2023.
[27] Maven, "Maven Repository," https://mvnrepository.com/, 2023.
[28] Android Developers, "CustomTabsClient," https://developer.android.com/reference/androidx/browser/customtabs/CustomTabsClient, 2023.
[29] A. D. Hartanto, Y. Pristyanto, and A. Saputra, "Document similarity detection using rabin-karp and cosine similarity algorithms," in *2021 International Conference on Computer Science and Engineering (IC2SE)*, vol. 1, 2021, pp. 1–6.
[30] "Missing class mappings of Support Library and AndroidX," https://issuetracker.google.com/issues/289442452, 2023.
[31] T. McDonnell, B. Ray, and M. Kim, "An empirical study of api stability and adoption in the android ecosystem," in *2013 IEEE International Conference on Software Maintenance*, 2013, pp. 70–79.
[32] L. Wei, Y. Liu, and S.-C. Cheung, "Taming android fragmentation: Characterizing and detecting compatibility issues for android apps," in *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2016, pp. 226–237.
[33] L. Wei, Y. Liu, S.-C. Cheung, H. Huang, X. Lu, and X. Liu, "Understanding and detecting fragmentation-induced compatibility issues for android apps," *IEEE Transactions on Software Engineering*, vol. 46, no. 11, pp. 1176–1199, 2020.
[34] H. Xia, Y. Zhang, Y. Zhou, X. Chen, Y. Wang, X. Zhang, S. Cui, G. Hong, X. Zhang, M. Yang, and Z. Yang, "How android developers handle evolution-induced api compatibility issues: A large-scale study," ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 886–898.
[35] Z. Zhang, H. Zhu, M. Wen, Y. Tao, Y. Liu, and Y. Xiong, "How do python framework apis evolve? an exploratory study," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2020, pp. 81–92.
[36] M. Rebouças, G. Pinto, F. Ebert, W. Torres, A. Serebrenik, and F. Castor, "An empirical study on the usage of the swift programming language," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, 2016, pp. 634–638.