Annotation practices in Android apps

Ajay Kumar Jha
Department of Computing Science
University of Alberta, Edmonton, Canada
ajaykum1@ualberta.ca

Sarah Nadi
Department of Computing Science
University of Alberta, Edmonton, Canada
nadi@ualberta.ca

***Abstract*—Understanding the adoption and usage of any programming language feature is crucial for improving it. Existing studies indicate that Java annotations are widely used by developers. However, there is currently no empirical data on annotation usage in Android apps. Android apps are often smaller than general Java applications and typically use Android APIs or specific libraries catered to the mobile environment. Therefore, it is not clear if the results of existing Java studies hold for Android apps. In this paper, we investigate annotation practices in Android apps through an empirical study of 1,141 open-source apps. Using previously studied metrics, we first compare annotation usage in Android apps to existing results from general Java applications. Then, for the first time, we study why developers declare custom annotations. Our results show that the density of annotations and the values of various other annotation metrics are notably less in Android apps than in Java projects. Additionally, the types of annotations used in Android apps are different than those in Java, with many Android-specific annotations. These results imply that researchers may need to distinguish mobile apps while performing studies on programming language features. However, we also found examples of extreme usage of annotations with, for example, a large number of attributes, as well as a low adoption rate for most annotations. By looking at such results, annotation designers can assess adoption patterns and take various improvement measures, such as modularizing their offered annotations or cleaning up unused ones. Finally, we find that developers declare custom annotations in different apps but with the same purpose, which presents an opportunity for annotation designers to create new annotations.**

I. Introduction

Annotations are a form of metadata used to associate additional information to program elements. For example, the `@Override` annotation informs the compiler that a method is overridden. Annotations are an integral part of any programming language and can be used for various purposes, such as compiler guidance or run-time processing [1]. Most annotations are predefined by the programming language or frameworks/libraries such as Android or JUnit. Developers can also declare and use custom annotations for their specific needs, such as generating boilerplate code. Researchers have also used annotations for tasks such as code inspection [2], [3], code comprehension [4], or code generation [5].

Studying the adoption and usage of language features (e.g., lambda expressions [6], generics [7], and exception handling [8]) is integral to improving these features and advancing a language [9]–[11]. Similarly, the study of annotations can help practitioners and researchers in improving annotation usage. For example, some practitioners have argued for or against the use of annotations [12]–[15] or warned about overusing annotations, with the argument that they can hinder the readability and maintainability of code [16], [17]. However, most of these arguments are based on anecdotal evidence from personal experiences without being grounded with empirical evidence about wide-spread annotation usage.

Given the above needs, researchers have performed some empirical studies to characterize annotation usage in Java projects [9], [18]–[20]. However, no study exclusively focused on annotation usage in Android apps. Although many Android apps are written in Java, it is not clear if their different nature may lead to different annotation usage patterns. For example, the size of Android apps are typically smaller than general Java projects, which might result in differences in the the prevalence of annotation use. It is also not clear if the same annotations used in Java applications are also used in Android apps, especially since there are Android Application Programming Interfaces (APIs) that are specifically tailored to the resource-constrained mobile environment. Moreover, the Android framework declares various annotations (e.g., `@TargetApi`) that can be used only in Android apps [21]. Thus, the results of existing Java annotation studies [9], [18]–[20] may not necessarily hold for Android apps.

In this paper, we study annotation usage in Android apps. We perform an empirical study of 1,141 Android apps, where we investigate previously studied phenomena and also study new characteristics. More specifically, we answer the following three research questions, and share an artifact [22]:

- ***RQ1: What is the distribution of annotation usage in Android apps?*** Using 10 existing annotation metrics [19], [20], we perform a replication study where we quantitatively investigate annotation usage in Android apps and compare the results to existing Java findings. We also identify apps with extreme usage of annotations [19]. Our results show that, similar to Java, the use of annotations is pervasive in Android apps (99.7% of apps). However, the characteristics of annotation usage in Android apps are different from general Java projects. We find that the median values for annotations per app and annotation per line of code in the studied apps are 86 and 0.02, respectively, which is much less than the counterpart values of 1,707 and 0.4, respectively, for general Java projects [20]. While we find that annotation usage at different granularity levels in Android apps are overall less than general Java applications, there are some apps that demonstrate extreme annotation usage [16], [17] where, for example, an annotation spans up to 73 lines

of code or uses up to 15 attributes.

- ***RQ2: Which annotations are used in Android apps?*** We categorize annotations used in Android apps into annotations from the Java language, Android framework, third-party libraries, or custom defined annotations. By analyzing the number of annotations and their frequency in each category, we can understand which annotations are used and whether Android apps use annotations that are different from those in general Java applications. Our results show that Android apps use 91 unique Android framework annotations and only 31 unique Java annotations. This shows that the majority of unique annotations are specific to the Android framework. However, the adoption rate for most of these annotations is low. We also found some differences due to characteristics of Android apps: while previous work shows that three annotations of the Java Persistence API (JPA) are in the list of top-10 frequently used annotations in Java projects [18], we find that Android apps instead use the lightweight Android library Room [23] for persistence, because JPA cannot be used in Android apps.
- ***RQ3: Why do developers declare their own annotations?*** Previous annotation studies did not investigate why developers declare their own annotations. Not only is this a question that gets asked by real developers [24], but finding these reasons can also help third-party annotation providers improve their annotation designs and offerings by incorporating some of the frequent needs into their annotations. We find six categories of custom-declared annotations, which include data persistence, field validation, and marking elements for specific purposes such as API discoverability [25].

Our results provide an empirical characterization of annotation usage in Android apps, and have several actionable implications for practitioners, tool builders, and researchers. For example, we find that there are a few annotation usages with a high number of attributes, while our investigations show that some annotation designers retroactively modularize annotations or remove attributes from the annotations that declare a large number of attributes. ***Annotation designers can use our empirical data on attribute usage to improve the modularization of their annotations***. Our studied adoption rates can ***help annotation designers to identify and clean up unused annotations***. Our first-time analysis of custom annotations provides ***insights for library designers to offer new annotations***. For example, we find some custom annotations from different apps with the same purpose, such as identifying Java native code; this presents an opportunity for annotation designers to design new libraries for this purpose. Interestingly, we find that many custom annotations are created for data persistence (specifically marking Java classes/fields that correspond to database tables/columns) while libraries for such purposes do exist [23]. Future ***researchers can investigate what motivates developers to create a custom annotation over using an existing library***. Finally, since our results show that annotation usage in Android is different than Java, ***we recommend that researchers consider this distinction in future studies***.

```
//meta-annotations
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface SEConference { //annotation body
    String name();
    String location();
    String year() default "2020";
}
```

Listing 1. Annotation type declaration in Java

II. Background

Annotations in a programming language are used to add metadata to program elements, which can be used for various purposes. For example, a compiler can use the metadata for code inspection and a library, e.g., JUnit, can use the metadata for customizing a program's behavior. *Built-in annotations* are predefined by programming languages, frameworks, and third-party libraries, while *custom annotations* are those declared by developers in their own applications. We now describe the process of declaring, using, and processing Java annotations.

An annotation declaration looks similar to a Java interface declaration, except the keyword *interface* in an annotation declaration is preceded by the "@" sign. Listing 1 shows an annotation declaration named *SEConference*. An annotation declaration has mainly two parts: a body and meta-annotations. An annotation declaration body contains element declarations. As shown in Listing 1, elements look similar to methods but do not have any parameters or body. They mostly act like fields. The elements can have default values and their return types must be one of the following: primitive types, String, enum, Class, annotations, or arrays of any of above. Similar to annotations, *meta-annotations* can be of built-in or custom types. Java has some key built-in meta-annotations such as `@Retention` and `@Target`. The `@Retention` meta-annotation specifies how long the declared annotation should be retained. Depending on its value `SOURCE`, `CLASS`, or `RUNTIME`, the annotation is available in the source file, class file, or at run time, respectively. The `@Target` meta-annotation specifies program elements to which the declared annotation can be associated such as type, method, or field.

Listing 2 shows an example usage of `@SEConference`, which can be associated with Class, interface, or enum program elements since its target element is `TYPE`. The value of each annotation element must be assigned when the annotation is used, unless the annotation element declares a default value.

Annotations are processed differently based on the retention policy. For annotations declared with a `SOURCE` or `CLASS` retention policy, the annotation creator provides an annotation processor to process the declared annotation. The annotation creator first writes an annotation processor and then registers the processor with the Java compiler. During compile-time, the compiler scans the code for the specified annotation and processes the annotation as defined in the annotation processor. For annotations declared with a `RUNTIME` policy, the annotation creator provides reflection code to process the declared annotation. The Java Virtual Machine uses the reflection code to retrieve the metadata and customize the program behavior.

```
@SEConference(name="SCAM", location="Australia")
public class MyConference { ... }
```

Listing 2. Example annotation usage of the annotation declared in Listing 1

III. Related Work

We divide previous work related to annotations into three categories: annotation usage (most related to our work), annotation misuse, and applications of annotations.

a) Annotation usage: Rocha and Valente [18] investigated 106 open-source Java projects to find what types of annotations are frequently used and what types of program elements frequently use them. They found that annotations related to code inspection, testing, and data persistence are frequently used in Java projects, and most of the annotations (over 90%) are used on methods. Lima et al. [19] defined seven annotation metrics to study annotation usage patterns. They analyzed 25 open-source Java projects to calculate frequency threshold values for these metrics. The frequency threshold values represent frequent or rare annotation usage phenomena. Yu et al. [20] investigated 1,094 open-source Java projects to find annotation usage distribution, annotation evolution, and impact of annotations on code error-proneness. Our work is inspired by these three studies. We perform a replication study where we investigate the frequency distribution of annotations by reusing the existing metrics. However, we focus our empirical study on Android apps and compare our findings to the existing knowledge on general Java projects. We additionally investigate different types of Android framework annotations, third-party library annotations, custom annotations and their purpose and usage, all of which are not investigated by these existing studies. Finally, our data set consists of 1,141 open-source Android apps, which is larger than the data set used for the first two studies, and comparable in size to the third.

There are also existing studies of other language features that touch upon annotations. Parnin et al. [7] investigated the adoption of Java generics in 40 open-source Java projects. In the process, they compare the adoption rate of Java generics to that of annotations since both language features were released around the same time. Similarly, Dyer et al. [9] investigated 18 Java language features, including annotations, in 31k open-source Java projects. However, both these studies discuss only the frequency of annotation usage in Java projects, but do not delve into the various types of annotations as our work does.

b) Annotation misuse: An annotation misuse characterizes any use of annotations that do not conform to the given specifications, which may be explicitly defined in documentation or implicitly represented by the annotation declaration. Pinheiro et al. [26] mined GitHub repositories to create a data set of annotation misuse in Java and C# projects, and they designed mutant operators to detect annotation misuse. Cordoba-Sanchez and Lara [27] proposed a domain-specific language and a tool to design and validate Java annotations. Similarly, Darwin [28] proposed a tool to verify the correct usage of annotations in Java. We do not provide tools or techniques to detect annotation misuse. However, our findings can help developers in avoiding annotation misuse.

TABLE I
Descriptive Statistics of the Studied Apps

Size in KLOC		Star Count in GitHub	
Range	**No. of Apps**	**Range**	**No. of Apps**
0≤1	223	0≤50	646
1≤10	600	51≤100	152
10≤25	153	101≤200	127
25≤50	97	201≤500	95
>50	68	>500	121
Reviews in Play Store		**Installs in Play Store**	
Range	**No. of Apps**	**Range**	**No. of Apps**
0≤1.0	54	0≤1,000	181
1.1≤2.0	2	1,001≤10,000	214
2.1≤3.0	21	10,001≤50,000	68
3.1≤4.0	176	50,001≤100,000	92
>4.0	401	>100,000	99

c) Applications of annotations: Eichberg et al. [3] used annotations to verify properties of program elements. Tan et al. [2] used annotations to detect concurrency bugs in the Linux kernel. Sulir et al. [4] investigated the recording of developers' concerns as annotations to improve program comprehension. Although we investigate the purpose of custom annotations, we do not design or propose new use cases for annotations.

IV. Data Set Curation

This section describes our selection of study subjects (i.e., Android apps) and the tools we use to extract annotation usage.

A. Study Subjects

We use F-Droid [29], a repository of free and open-source Android apps, to select apps. We collect URLs of all the apps stored on F-Droid and select only those apps that are hosted on GitHub, resulting in a total of 1,209 apps. We limit our study to the apps hosted on GitHub to filter inactive apps. We further limit our study to apps that are written in Java to allow us to compare our results with the existing studies on annotation usage in general Java projects. Therefore, among the 1,209 collected apps, we remove 45 apps written in Kotlin and 2 apps written in Xtend. We also remove 21 apps that do not have source files or whose source files are archived. This leaves us with 1,141 Android apps that we use for our study; 654 of these apps (57%) are also available on Google Play.

Table I shows the descriptive statistics of the studied 1,141 apps, which include the apps' internal and external properties such as size in lines of code (LOC), star count on GitHub, user reviews on the Play store, and install counts on the Play store. Note that the properties related to the Play store are available only for the 654 apps listed there; these apps belong to 31 different app categories such as tools, games, shopping, communication, etc. Overall, the descriptive statistics show that our data set contains apps with varying size and popularity.

B. Extracting Annotations

Rocha and Valente [18] used *apt* (Annotation Processing Tool) [30] to extract annotation data. However, *apt* has been deprecated since Java SE 7. Moreover, the authors reported that the tool could not retrieve annotations used in anonymous

TABLE II
DESCRIPTION OF ANNOTATION METRICS FROM LIMA ET AL. [19] AND YU ET AL. [20] THAT WE USE IN RQ1

Annotation Metrics Name	Metric Origin	Annotation Metrics Description
Annotations per app	[20]	Number of annotations in an app
Annotations per file	[20]	Number of annotations per line of code for each file
Annotations per program element	[20]	Number of annotations per annotated program element (e.g., method or class)
Attributes in Annotation (AA)	[20]	Number of attributes used in an annotation.
LOC in Annotation (LOA)*	[19]	Number of lines of code occupied by an annotation.
Annotation Nesting Level (ANL)	[19]	Number of nesting levels in an annotation.
Annotations per Program Element (APE)*	[19]	Number of annotations used on a program element.
Annotations in Class (AC)	[19]	Number of annotations in a class.
Unique Annotations in Class (UAC)	[19]	Number of distinct annotations in a class.
Annotation Schemas in Class (ASC)	[19]	Number of different annotation schemas (i.e., packages) in a class.

*Lima et al. [19] used the terms *annotation definition*, *annotation declaration*, and *annotation use* interchangeably when referring to an annotation usage (e.g., Listing 2). For consistency and clarity, we changed the wording of some metric names; for brevity, all metrics simply use the term *annotation* to refer to an annotation usage. We changed the name of *LOC in Annotation Declaration (LOCAD)* to *LOC in Annotation* (LOA) and *Annotation in Element Declaration* (AED) to *Annotations per Program Element (APE)*. Note that Lima et al. calculate APE across *all* program elements not just across *annotated* program elements.

classes, which have been supported since Java SE 8. Lima et al. [19] developed Annotation Sniffer [31] to extract the values of their proposed annotation metrics from source files. However, Annotation Sniffer also cannot retrieve annotations used in anonymous classes; additionally, if a Java source file has more than one class, it retrieves annotations of only one class. Yu et al. [20] implemented their own extractor based on the Spoon library [32]. To avoid the above problems, we also implement our own annotation extraction and analysis tooling, but base it on a more popular library named JavaParser [33].

We use JavaParser APIs to extract annotations from the source code. For each app, we obtain the AST of the Java code. We then identify annotation uses and annotation declarations in the AST. We also identify different program elements on which the annotations are used. Moreover, we analyze the fully qualified name of the annotations and the import statements to distinguish different types of annotations such as Java, Android framework, or third-party library annotations (RQ2). We collect and store all annotation usage data for each app. Finally, we use Guava [34] to calculate the descriptive statistics. All our code and data is available online [22].

V. ANNOTATION USAGE IN ANDROID APPS

In this section, we present the results of our study. For each research question, we describe the motivation behind studying that question. We then describe the method used to answer the question and then describe the obtained results.

A. RQ1: What is the distribution of annotation usage in Android apps?

1) Motivation: Existing studies showed that the use of annotations is pervasive in general Java projects [9], [18]–[20]. However, Android apps are typically smaller in size than other Java applications. For example, the size of 72% of the apps used in this study is less than 10 KLOC whereas the size of only 8-9% of the Java projects used in the existing studies is less than 10 KLOC [18], [19]. Therefore, in this first RQ, we perform a replication study of the quantitative metrics presented in previous work. Our goal is to understand the distribution of annotation usage in Android apps, and determine whether it is different from Java projects. Given existing arguments against excessive annotation usage [16], [17], we are also interested in looking at examples of real Android apps with extreme annotation usage, if any.

2) Method: To analyze the distribution of annotation usage in Android apps, we reuse 10 metrics studied in previous work [19], [20], and described in Table II, and compare our findings to the published findings for general Java projects. We calculate each metric for each app in our data set and report the distribution across all apps. For the seven metrics reused from Lima et al. [19], the original authors used a percentile rank analysis method to understand the typical value of each metric. Such typical values can help identify anomalous usages of annotations, such as overusing annotations, which is commonly frowned upon from a maintenance and readability perspective [16], [17]. To compare our results with theirs, we also follow their analysis method, which works as follows.

For each metric, they first calculate the distribution of the values per project and calculate the corresponding 90th percentile, 95th percentile, and 99th percentile. They then calculate the mean of each percentile value across all the studied projects. For each metric, they define frequency thresholds according to the average percentile value as very frequent (90th percentile), frequent (95th percentile), and less frequent (99th percentile). For example, let us assume that the 90th percentile for the annotations in class (AC) metric from Table II is 10, then we can say that having a class with 10 or less annotations is a very frequent phenomenon. On the other hand, if the 99th percentile is 20 annotations in a class, which means it is a less frequent phenomenon, then we can deduce that classes with more than 20 annotations demonstrate extreme usage. Based on our frequency results, we manually analyze a few extreme cases (i.e., apps with particularly high metric values) to understand extreme annotation usages.

3) Results: We find that 1,137 of the 1,141 apps (99.7%) have at least one annotation, and that there are 382,074 annotation usages in total. We could not find any distinguishing characteristics for the 4 apps not using annotations; we focus on the 1,137 apps with annotations for the rest of the study. Similar to recent findings showing that all studied Java applications contained annotations [20], our results similarly show that annotation usage is also pervasive in Android apps.

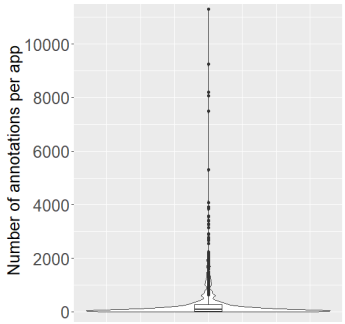

Fig. 1. Annotation density per app for 1,137 studied apps.

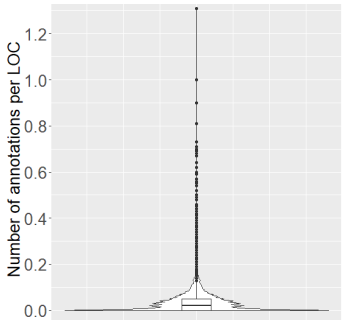

Fig. 2. Annotation density per LOC for 78,791 studied Java files

TABLE III
ANNOTATIONS PER ANNOTATED PROGRAM ELEMENT

Num. Annotations	Num. Elements	Num. Annotations	Num. Elements
1	352,946	6	12
2	12,711	7	4
3	1,052	8	5
4	69	9	6
5	9	>10	3

Density metrics. We first discuss the first three metrics from Table II related to annotation density across different granularity levels. Figure 1 shows the violin plot for the number of annotations per app for the studied 1,137 apps. The number of annotations per app ranges from 1 to 11,314 with a mean and median value 336 and 86, respectively. This median value is much less than the median of 1,707 annotations per app found by Yu et al. [20]. In general, we find that 79% of the apps use less annotations than the mean value, while 84% of the apps have less than 500 annotations. Only 35 apps show extreme values, with more than 2,000 annotations each.

Figure 2 shows the the violin plot for the density of annotations per LOC for the 78,791 Java files in the studied apps. We find that 94% of the files have a density between 0.0 and 0.1, whereas only 29 files have a density larger than 0.5. A density of 0 indicates that a file does not have any annotations while a density of more than 0.5 means that the file contains annotations that occupy more lines of code than the actual code in the file. Overall, the density per LOC ranges from 0.0 to 1.31 for the studied files with a mean and median value of 0.03 and 0.02, respectively. This is notably less than the median value of 0.4 obtained by Yu et al. [20].

Table III shows the density of annotations per program element. We can see that most annotated program elements (96%) have only one annotation. We find only 30 annotated program elements that each have more than 5 annotations, with 13 being the maximal number of annotations on a single program element. Yu et al. [20] also found that most of the program elements (99%) have 1 or 2 annotations. However, they found 2,978 of the 4,462,419 program elements that each have more than 5 annotations, which is much higher than the only 30 program elements we find.

Threshold Values. We now discuss the threshold values for the remaining seven metrics, proposed by Lima et al. [19], from Table II. Table IV shows the threshold values of these metrics which we obtained for Android apps versus the threshold values obtained by Lima et al. [19] for Java projects. We also show the highest metric value in each case. The results in Table IV show that, with only rare exceptions for ASC and ANL, most of the frequency threshold values for Android apps are less than their corresponding frequency values for Java projects. Overall, these results demonstrate that annotation distribution in Android apps across different metrics is generally less than annotation distribution in Java projects. However, it is interesting to note that the highest observed values for the attributes in annotation (AA) and LOC in annotations (LOA) metrics are actually higher than those in Java projects. We investigate these two metrics more closely.

The *attributes in annotations (AA)* metric measures the number of attributes used in an annotation. For example, the `@SEConference` annotation in Listing 2 uses 2 attributes, name and location. Using a high number of attributes in an annotation can affect the readability and maintainability of the annotation [19]. Buse and Weimer [35] found that the number of identifiers in a line of code has strong negative impact on code readability. Thus, it is useful for annotation users to avoid using many attributes. It is also useful for annotation designers to be aware of how their attributes are being used and perhaps redesign the annotation to offer less attributes by, for example, breaking it into two or more annotations.

Among 382,074 studied annotations, we find that 91% do not use any attributes while only 0.4% of the annotations use more than one attribute. The very frequent, frequent, and less frequent threshold values of the AA metric are 0.23, 0.42, and 0.71, respectively. Based on the less frequent threshold, we can conclude that it is rare for annotations in Android apps to use more than one attribute. We, therefore, manually analyze 21 annotations that demonstrate extreme behavior where they use more than 5 attributes. We find that 19 of the 21 annotations are `@ReportsCrashes` annotations of the ACRA library [36], which can have a maximum of 41 attributes. The remaining 2 annotations are `@ModulePrefs` annotations from Google Web Toolkit [37], which can have a maximum of 21 attributes. While some app developers used more than five of these available attributes for both annotations, we also observed that 16 attributes of `@ReportsCrashes` and 14 attributes of `@ModulePrefs` are not used at all in the studied apps. In general, while attributes provide some flexibility in customizing the annotation's behavior, they may reduce its modularity. Therefore, if an annotation contains a large number

TABLE IV
THRESHOLD VALUES OF THE ANNOTATION METRICS FROM LIMA ET AL. [19], AS DEFINED IN TABLE II. ↑ SHOWS ANDROID THRESHOLDS THAT ARE HIGHER THAN THEIR JAVA COUNTERPARTS, ↓ SHOWS THOSE LOWER, AND − SHOWS VALUES WITH NO CHANGE.

Metrics	**Android Apps**				**Java Projects** Lima et al. [19]			
	Very Frequent	Frequent	Less Frequent	Highest Value	Very Frequent	Frequent	Less Frequent	Highest Value
AA	↓ 0.23	↓0.42	↓0.71	↑15.00	1.00	1.00	2.00	9.00
LOA	− 1.00	−1.00	↓1.05	↑73.00	1.00	1.00	2.00	58.00
ANL	−0.00	−0.00	↓0.00	↓1.00	0.00	0.00	0.08	4.00
APE	↓0.50	↓0.88	↓1.07	↓13.00	1.00	1.00	2.00	27.00
AC	↓7.26	↓10.26	↓17.06	↓269.00	11.00	20.00	62.00	729.00
UAC	↓2.22	↓2.74	↓3.84	↓144.00	3.00	4.00	9.00	375.00
ASC	↑1.61	↓1.77	↓2.01	↓9.00	1.50	1.80	2.40	13.00

of attributes and it is being used by developers, annotation designers should consider remodularizing the annotation by looking at our metric results related to the usage of available attributes. For new annotations, annotation designers may consider limiting the number of attributes they offer to less than the frequent threshold value.

The *LOC in annotation (LOA)* metric measures the number of lines of code occupied by an annotation. For example, the `@SEConference` annotation in Listing 2 uses only 1 line of code. Among the studied annotations, we find that 99.9% annotations have a LOA value of 1. The very frequent, frequent, and less frequent threshold values are 1.00, 1.00, and 1.05, respectively. This indicates that most annotations simply span one line, which is what one would expect. Interestingly, we find a few number of annotations that have a large LOA value reaching up to 73, which is actually higher than the highest value observed in Java projects. Such extreme LOA values can affect the readability and maintainability of the annotations [19]. Although breaking up a statement across multiple lines improves readability, an annotation occupying a large number of lines increases the average number of identifiers per line, which negatively affects readability [35].

We manually analyze 10 annotations that span more than 20 LOC. We find that 3 of these 10 annotations are from the ACRA library, such as `@ReportsCrashes` and `@AcraCore` annotations. As previously mentioned, these annotations also have a high AA value (4-10); naturally, using multiple attributes forces the annotation to span multiple lines. The remaining 7 annotations are of type `@Module`, `@IntDef`, and `@JsonPropertyOrder`. Although these 7 annotations have a very high LOA value reaching up to 73, they only have 1 or 3 attributes. However, these attributes expect a long list of values, which again results in spanning multiple lines. In general, it seems that the number of attributes or the values expected by the attributes affects the number of lines an annotation spans. A Pearson correlation test on the studied annotations (n=382,074) does indeed show a statistically significant weak correlation between AA and LOA metrics (*r=0.18, p-value<2.2e-16*).

Interestingly, the recommendation for annotation designers to modularize their annotations and avoid a large number of attributes has been practiced in ACRA [36]. Note that two annotations from this library, `@ReportsCrashes` and `@AcraCore`, have extreme AA and LOA metric values in our data. The documentation [38] shows that the library evolved through ten versions, starting from one annotation, `@ReportsCrashes`, to sixteen different annotations. `@ReportsCrashes` had 41 attributes in the initial version (4.5.0). However, in version 5.1.3, the `@ReportsCrashes` annotation was removed and fifteen new annotations were introduced, out of which seven annotations, `@AcraCore`, `@AcraHttpSender`, `@AcraDialog`, `@AcraMailSender`, `@AcraLimiter`, `@AcraToast`, and `@AcraNotification`, directly resulted from the modularization of the `@ReportsCrashes` annotation. While this is an anecdotal example, it shows that annotation designers realize the need for remodularization. By looking at our metric results and data, annotation designers can find potential modularization and re-design opportunities for their annotations.

4) Discussion: The results of RQ1 show that there are differences in annotation usage between Android apps and general Java applications. In general, the density metrics suggest that the density of annotations in Android apps is less than that in Java projects. The seven other metrics in Table IV showed similar trends, where the vast majority of threshold values in Android were less than their Java counterpart. However, our data reveals that despite Android apps generally using less annotations, extreme annotation usage still exists. In general, our new Android threshold values can help app developers and annotation designers know the typical values for these annotation metrics and discover potential problems in their design, such as number of attributes. Similarly, app developers can use these metric thresholds to know if their annotation usage is different from typical usage.

Answer to RQ1: Annotation usage is pervasive in Android apps (99.7%), but annotation density is notably less than in general Java projects. This is also true for other metrics, but we still found extreme cases of annotation usage.

B. RQ2: Which annotations are used in Android apps?

1) Motivation: So far, we have studied various metrics related to annotation usage, but not the particular types of annotations used. We want to understand if the types of annotations used in Android apps is different than Java. Understanding the annotation types used is also important for practitioners. For example, the Android framework and third-party library annotation designers can understand the adoption

TABLE V
TOP 10 FREQUENTLY USED ANNOTATIONS

Annotation	Category	Frequency (%)
@Override	Java	257,578 (67.41%)
@NonNull	Android	33,610 (8.80%)
@Test	Library	17,476 (4.57%)
@Nullable	Android	14,391 (3.77%)
@SuppressWarnings	Java	6,830 (1.79%)
@BindView	Library	4,589 (1.20%)
@SuppressLint	Android	2,819 (0.74%)
@Inject	Java	2,652 (0.69%)
@TargetApi	Android	2,644 (0.69%)
@RunWith	Library	1,447 (0.38%)

pattern of their annotations, which in turn can help them in designing new annotations or cleaning up unused ones.

2) Method: We first differentiate Java annotations from Android framework annotations. To do so, we categorize annotations from packages starting with `java` or `javax` as Java annotations and annotations from packages starting with `android`, `androidx`, or `com.android` as Android framework annotations. Additionally, to provide meaningful insights to annotation developers about potential opportunities for new annotations or for improving annotation adoption, we further categorize the remaining annotations into third-party libraries and custom annotations. We categorize annotations as custom annotations if they are declared in the apps in which they are used, and then categorize any remaining uncategorized annotations as third-party libraries. We calculate the frequency of each annotation found in our data set across all apps and then determine the top 10 frequently used annotations.

3) Results: We find 845 unique annotations with 382,074 annotation usages. Among these, 539 annotations appear only once in 539 different apps. Only 88 annotations are used in more than 10 apps and only 12 annotations are used in more than 100 apps. This suggests that only a small number of annotations are frequently used across apps. We also find that Java annotations constitute 71% of the total frequency of all the annotations used in the studied apps. This is followed by Android annotations at 17% and library annotations at 11%. We find that using custom annotations is very rare, accounting only for 1% of the total number of annotation usages.

Table V shows the top 10 most frequently used annotations in our apps, with their corresponding category. The frequency of these top 10 annotations constitutes 90% of the total frequency of all the annotations used in our apps. Out of our top 10 list, two Java annotations and one third-party library annotation (`@Override`, `@Test`, `@SupressWarnings`) have also been found as frequent Java annotations by Rocha and Valente [18]. However, unlike their results, which also found three different Java persistence annotations and four different third-party library annotations, our results do not show frequent use of these annotations. Instead, 4 of our top 10 frequently used annotations are Android framework annotations and 1 annotation (`@BindView`) is an Android-specific third-party library annotation.

We now dive deeper into each annotation category. We start with Java annotations. We find 31 unique Java annotations that belong to 7 different packages. However, only 12 of these annotations, belonging to 3 different packages, have been used in more than 10 apps. We show these annotations in Table VI. We find that Java annotations are mainly used in the apps to indicate overridden methods, suppress compiler warnings, indicate deprecated methods, declare annotations, and inject dependencies. Although previous results indicate mostly similar usages of annotations in Java projects [18], one notable difference we see is that unlike Java projects which frequently use Java Persistence annotations for managing databases, Android apps do not use these annotations.

TABLE VI
JAVA ANNOTATIONS USED IN $>$ 10 APPS

Package Name	Annotations
java.lang	@Override, @SuppressWarnings, @Deprecated, @SafeVarargs
java.lang.annotation	@Retention, @Target, @Documented
javax.inject	@Inject, @Singleton, @Named, @Qualifier, @Scope

TABLE VII
ANDROID FRAMEWORK ANNOTATIONS USED $>$ 10 APPS

Package Name (android.)	Annotations
support.annotation	@NonNull, @Nullable, @StringRes, @ColorInt, @IdRes, @DrawableRes, @VisibleForTesting, @RequiresApi, @ColorRes, @IntDef, @LayoutRes, @WorkerThread, @Keep, @MainThread, @UiThread, @AttrRes, @CallSuper, @StyleRes, @IntRange, @StringDef, @DimenRes, @CheckResult, @RawRes, @XmlRes, @ArrayRes, @MenuRes
annotation	@SuppressLint, @TargetApi
arch.persistence.room	@Query, @ColumnInfo, @Ignore, @Entity, @PrimaryKey, @Index, @Insert, @Dao, @Database
support.test.filters	@SmallTest, @LargeTest
test.suitebuilder.annotation	@LargeTest
webkit	@JavascriptInterface

Next, we analyze the Android framework annotations. We find a total of 91 unique Android framework annotations belonging to 16 different packages. However, only 41 of these annotations are used in more than 10 apps. These 41 Android framework annotations belong to 6 different packages, shown in Table VII. Most of the frequently used Android framework annotations belong to `android.support.annotation` and `android.annotation`, and are used mainly for checking null conditions, suppressing lint warnings, handling OS fragmentation, validating resource types, and handling threads. The remaining frequently used annotations are mainly used for managing databases, testing, and exposing methods to JavaScript. As the package names in Table VII show, Android apps use annotations from various Android APIs and libraries that are tailored to the mobile environment, such as annotations from the Room library in place of Java persistence APIs.

Next, we analyze third-party library annotations. We find a total of 184 unique annotations belonging to 34 annotation libraries. Out of these, 13 different annotation libraries containing 127 unique annotations are used in more than 10 apps. Table VIII shows these 13 annotation libraries. The first column shows the package name of the library, the second column shows the main purpose of the library, the third column shows the number of the library's unique annotations used in our apps, the fourth column shows the total frequency of the library's annotations in our apps, and the last column shows

TABLE VIII
ANNOTATIONS FROM THIRD-PART LIBRARIES USED IN MORE THAN 10 APPS

Annotation Library	Main Purpose	Unique Annotations	Total Frequency	No. of Apps
org.junit, org.junit.runners	Testing	17	21,561	362
butterknife	Binding views	20	6,126	77
org.robolectric.annotation	Testing	5	521	46
retrofit2.http	Type-safe HTTP client	20	2,300	44
org.acra.annotation	Reporting crashes	7	61	43
dagger	Dependency injection	13	1,274	40
org.mockito	Testing	4	1,138	33
com.google.gson.annotations	Serializing/deserializing Java objects to/from JSON	3	1,066	33
org.greenrobot.eventbus, org.greenrobot.greendao.annotation	Components communication, mapping objects to SQLite	8	495	30
com.fasterxml.jackson.annotation	JSON processor	17	1,412	27
org.jetbrains.annotations	Code inspection (null check)	6	372	25
org.powermock	Testing	3	54	13
com.bumptech.glide.annotation	Media management and image loading	4	17	13

the number of apps that use annotations from the library. Although third-party annotations constitute only 11% of the total frequency of all annotations used in the apps, Table VIII shows that some of the annotation libraries are used in a large number of apps. For example, annotations from the *JUnit* and *butterknife* library appear in 362 and 77 different apps, respectively. In Table V, we also see that the `@Test` and `@RunWith` annotations from *JUnit* and the `@BindView` annotation from *butterknife* are among the top 10 frequently used annotations. Given the purpose of each annotation library shown in Table VIII, we can conclude that app developers use annotation libraries mainly for testing, binding views, injecting dependencies, and processing JSON and XML data.

Finally, although we find 591 custom annotations declared in 107 apps, only 504 of them (representing 60% of total unique annotations) are used. However, the frequency of the custom annotations used in the studied apps is rare. They only constitute 1% of the total frequency of used annotations. RQ3 studies the purpose of these custom annotations.

4) Discussion: Although Java and the Android framework define hundreds of annotations, only a few of these annotations are frequently used in Android apps. For example, the 10 annotations listed in Table V constitute over 90% of the total frequency of all the annotations used in the studied apps. Moreover, less than 50% of the unique Java annotations and the unique Android framework annotations in our data set are used in more than ten apps. This indicates a low adoption rate for the majority of Java and Android framework annotations we observe. Despite the low adoption rate, our results still bring good news for Android framework annotation designers: a large number of unique Android framework annotations are used in the studied apps, which suggests that app developers are looking beyond frequently used annotations. Tool support that recommends relevant annotations to developers based on their code could be one option to increase the adoption rate.

Our results also show that app developers use various third-party annotation libraries. However, we note that the most frequently used annotation libraries are either already popular in Java projects, such as JUnit and Retrofit, or are endorsed in the official Android developers' website, such as Robolectric and Dagger. Official endorsement may help in improving the adoption rate of third-party annotation libraries.

Answer to RQ2: The adoption rate of most of the Java and Android framework annotations, as well as custom annotations, is very low. Except for a few testing and binding views-related annotations, app developers rarely use third-party library annotations.

C. RQ3: Why do developers declare their own annotations?

1) Motivation: RQ2 showed that there is a total of 591 custom annotations in our data, 504 of which get used. In this RQ, our goal is to understand what app developers use these custom annotations for. While the purpose of frequently used built-in annotations can be found in official documentation sources, it may not be clear to developers why they might need to define a custom annotation [24]. There is currently, to the best of our knowledge, no empirical studies of custom annotations. Thus, results of RQ3 can help app developers in understanding typical needs for custom annotations. Moreover, annotation designers can design new annotations based on the common reasons for creating custom annotations.

2) Method: First, we identify all the custom annotation declarations in the studied apps. We then identify the custom annotations that are used to customize the Java or Android framework annotations by analyzing their meta-annotations. For each meta-annotation of the custom annotations that are not `@Retention`, `@Target`, or `@Documented` annotation, we check the Java or Android framework documentation to identify their purposes. Finally, to categorize and identify the purpose of the custom annotations that are not used to customize the Java or Android framework annotations, we follow an open coding approach [39] as follows.

For each annotation, the first author first manually analyzed the comments written by developers in the annotation declaration as well as the code that declares the annotation and any code that uses the annotation. The first author then wrote a short descriptive phrase about the purpose of the annotation, e.g., *"Tagging a field to be validated for positive numeric values"*. The second author then independently repeated this task and marked any annotations with which they disagreed with the provided descriptions or for which the description was empty and required discussion. In total, we had 14 entries that required discussion. We do not measure an inter-rater agreement, because we did not have pre-defined categories

or true/false labels. After the discussion and ensuring each annotation has a descriptive phrase, both authors performed card sorting together. They start by grouping annotations with similar descriptive phrases and then iterate again to combine or divide groups. Since existing studies [18]–[20] do not analyze custom annotations, we do not compare results in this RQ.

3) Results: We find 591 custom annotation declarations in 107 apps. Among these, 87 annotations do not actually get used at all. For example, the AndFHEM app [40] declares 8 custom annotations but only uses 1 of them in the code. Therefore, we focus on the 504 custom annotation declarations that do get used. Among the 504 custom annotation declarations, we identify 454 custom annotation declarations (90%) that are used to customize the Java or Android framework annotations, which we refer to as *Java/Android custom*. We find that 371 of those are used to declare a data type similar to the `enum` data type in Java. These custom annotations are used as interfaces for the `@IntDef` and `@StringDef` Android framework annotations. The `@IntDef` and `@StringDef` framework annotations denote that the value of the annotated element should be one of the explicitly named integers or string constants. These framework annotations are used as meta-annotations of the custom annotations to create an effect similar to Java's `enum`. However, the custom annotations created with `@IntDef` or `@StringDef` annotations have a performance advantage over `enum` [41]. Listing 3 shows a custom annotation declaration and use, taken from SoundWaves app [42]. The custom annotation `@Action` uses the `@StringDef` annotation to create a data type that can have five different string values. The use of the `@Action` annotation on `getAction()` method then ensures that the method returns one of the five string values. Among the remaining 83 Java/Android custom annotations, we find 57 qualifier annotations and 26 scope annotations. *Qualifier annotations* are used to distinguish different instances of objects of the same type, whereas *scope annotations* are used to create custom scopes within a program. These annotations use `@Qualifier` and `@Scope` Java annotations as their meta-annotations to define custom qualifiers and scopes, respectively.

The remaining 50 custom annotation declarations (10%) are not used for customizing the Java or Android framework annotations. Based on our open coding of these annotations, we identify six reasons for declaring the annotations as follows. We find 14 annotations that are declared for *data persistence*, such as specifying a table or column name or tagging a field that can be writable. We also find 11 annotations that are declared for *testing*, such as filtering test classes or package containing test classes. For example, Listing 4 shows the `@TestScanPackage` custom annotation from the Nextcloud [43] app, which is used to identify packages containing test suites. We also find 10 annotations that are declared to *distinguish program elements that are used for specific purposes*, such as identifying program elements that are used by Java Native Interface (JNI) framework or indicating that the program elements' visibility can be made public to allow API discoverability [25]. We find 7 annotations that are used

```
//annotation declaration
@StringDef({DOWNLOAD, DELETE, PLAY, NEW, FLATTR})
@Retention(RetentionPolicy.SOURCE)
public @interface Action {}
//annotation use
@Action
public String getAction() { return action;}
```

Listing 3. Custom annotation declaration and use in SoundWaves app

for *validating fields* such as checking positive numeric values or empty values or indicating a required field. We find 4 annotations used for *modifying app run-time behavior*, such as adding navigation tabs or setting preference values. Finally, we find 2 annotations that are declared for *documenting developer concerns or comments* in the source code [4].

4) Discussion: Our results show that app developers declare most of the custom annotations (90%) in the studied apps for customizing the Java or Android framework annotations. For example, 371 custom annotations declare `@IntDef` and `@StringDef` Android framework annotations. They declare only 50 custom annotations exclusively for their specific needs. Interestingly, there are annotation libraries available for the same purpose as the purpose used to declare some of the custom annotations. For example, some of the custom annotation declarations we find related to persistence exist in libraries for Android such as Room [23]. In the future, it would be interesting to ask developers why they choose to declare their custom annotation versus use an existing library that provides this annotation. On the other hand, we find cases of custom annotations, such as identifying Java native code, where there is no corresponding annotation library available. This presents an opportunity for annotation library designers to design new annotations.

Answer to RQ3: 90% of custom annotations are declared to customize Java/Android annotations such as `@IntDef`, `@Qualifier`, and `@Scope`. The remaining 10% are used for data persistence, testing, distinguishing program elements, validating fields, or modifying the app's run-time behavior.

VI. Actionable Implications of Our Results

We now discuss implications of our results for practitioners (annotation designers and clients) and researchers.

a) Annotation Designers: Annotation designers need to understand how annotations are used in practice in order to improve their adoption and facilitate their usage. Our results can help annotation designers in evaluating adoption patterns, which in turn can help them in improving usage of the annotations. By using our tooling or data set, annotation designers can check which annotations are not used or rarely used in practice and perform maintenance activities, such as clean up unused annotations or improve documentation for the unused annotations. For example, Android annotations such as `@InterpolatorRes`, `@RepetitiveTest`, `@Beta`, or `@RequiresDevice` have been used in only one app and the `@RepetitiveTest` annotation has already been deprecated.

Annotation designers can also use our results on different annotation frequency metrics, use our data set to find annotation usage in practice, or apply our tooling on new data in order to improve the design of new and existing annotations. Moreover, our results on custom annotations present opportunities for annotation designers to create new annotations, such as for identifying program elements that use Java Native Interfaces (JNI). We also find that app developers declare various custom annotations for data persistence, although a library named Room [23] is available for data persistence in Android apps. This presents an opportunity for annotation designers to create new annotations or extend current ones.

b) Annotation Clients: RQ2 results can serve as a source of information for annotations used in practice and their popularity, which may help developers decide if certain annotations are relevant for them to use. Besides, our results from RQ3 help in filling the information gap related to when developers can/should create custom annotations [24]. Finally, our results show that only a few Android annotations are used frequently, and four Android annotations listed in the top 10 most frequently used annotations (Table V) can be either directly inferred and inserted by the Android Studio (`@Nullable` and `@NonNull`) or indirectly inferred from the Android Studio lint warnings (`@SuppressLint` and `@TargetApi`). The results indicate that the usage of Android annotations is significantly influenced by the annotation inference and placement abilities of the Android Studio. Therefore, to increase usage of other annotations, tool and IDE builders may consider providing support for automatically inferring and recommending annotations to developers.

c) Researchers: Our results show that annotation usage is different in Android apps from general Java applications. Mobile apps have certain restrictions, such as size, and also use different libraries that affect usage of programming language features. Therefore, researchers performing studies on programming language features may want to distinguish mobile apps if the language is not exclusively used in the mobile environment. Our results also raise new research questions such as why developers create custom annotations, such as for persistence, when there are existing libraries for them.

VII. Threats to Validity

a) Construct validity: In RQ1 and RQ2, we compare our findings to those by previous work on annotations in Java projects. We note that the data set of 1,094 Java projects used by Yu et al. [20] also contains Android apps. Specifically, we analyzed their data set and found 217 Android apps. However, we do not compare our findings only with Yu et al., but we also compare them with two other papers on annotation usage in Java projects [18], [19] whose data sets do not contain any Android app. Given the differences we found across various metrics, we can safely conclude that annotation usage in Android apps has differences to annotation usage in Java apps.

b) Internal validity: We used various APIs of the JavaParser library [33] to collect data about annotation usage. JavaParser is a widely used library; we also manually validated some of the results to gain confidence in the tool. We did not explicitly remove cloned apps; however, there were only 11 cloned apps in our data set, with many modifications to the source code. Although we filtered inactive apps, some of the studied apps might have become inactive since we downloaded the apps. In RQ3, we manually analyzed some of the collected data to understand the purposes of declaring and using annotations. In the absence of comments, both authors had to use their best judgement based on the provided code. Manual tasks are subject to mistakes, and we may have misinterpreted the purpose of some annotations. We release all the artifacts used in this study for further validation [22].

```
//annotation declaration for testing in Nextcloud
@Retention(RetentionPolicy.RUNTIME)
public @interface TestScanPackage {
    public String value();
}
//reflection code for annotation processing
TestScanPackage annotation =
    clazz.getAnnotation(TestScanPackage.class);
if (annotation == null) {
    throw new InitializationError("No package given ...");}
return annotation.value();
```

Listing 4. Custom annotation taken from Nextcloud app and used for testing

c) External validity: Given that we need to analyze source code to understand the purposes of annotation declaration and usage, we limited our work to open-source apps. To find such apps, we used all the apps from F-Droid [29], which has also been extensively used in previous work [44]–[47]. We further limited our study to the apps hosted on GitHub to filter inactive apps. As a result, we analyzed annotation usage only in 1,141 apps, which is a small number of apps in comparison to millions of apps available in Google Play [48]. Although the results of this study might not generalize beyond the apps that we studied, 57% of the studied apps are also available in Google Play, indicating that these are real apps. Moreover, our data set is much larger than previous annotation studies [18], [19], and the descriptive statistics of the studied apps in Table I suggest that we cover a wide variety of apps.

VIII. Conclusion

In this paper, we performed an empirical study to understand annotation usage in Android apps and whether it is different from general Java projects. We investigated three research questions: annotation distribution at various granularity levels, usage of different types of annotations and their adoption frequency, and purpose of custom annotations. Our study generates various findings about annotation distribution, types, adoption, and declaration in Android apps, which can help app developers and annotation designers in improving the usage of annotations in Android apps. For example, designers can improve the annotation usage by designing annotations needed by developers, adjusting the design of new and existing annotations, and cleaning up unused annotations.

Acknowledgment

This research was undertaken thanks to funding from the Canada Research Chairs program.

REFERENCES

[1] Oracle, "Java annotations," https://docs.oracle.com/javase/tutorial/java/annotations/, 2020.

[2] L. Tan, Y. Zhou, and Y. Padioleau, "acomment: mining annotations from comments and code to detect interrupt related concurrency bugs," in *Proceedings of the 2011 33rd International Conference on Software Engineering (ICSE)*. IEEE, 2011, pp. 11–20.

[3] M. Eichberg, T. Schäfer, and M. Mezini, "Using annotations to check structural properties of classes," in *Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering*. Springer, 2005, pp. 237–252.

[4] M. Sulír, M. Nosál', and J. Porubän, "Recording concerns in source code using annotations," *Computer Languages, Systems & Structures*, vol. 46, pp. 44–65, 2016.

[5] H. Zhang, Z. Chu, B. C. d. S. Oliveira, and T. v. d. Storm, "Scrap your boilerplate with object algebras," in *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2015, pp. 127–146.

[6] P. M. Uesbeck, A. Stefik, S. Hanenberg, J. Pedersen, and P. Daleiden, "An empirical study on the impact of c++ lambdas and programmer experience," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 760–771.

[7] C. Parnin, C. Bird, and E. Murphy-Hill, "Adoption and use of java generics," *Empirical Software Engineering*, vol. 18, no. 6, pp. 1047–1089, 2013.

[8] G. B. de Pádua and W. Shang, "Revisiting exception handling practices with exception flow analysis," in *Proceedings of the 17th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2017, pp. 11–20.

[9] R. Dyer, H. Rajan, H. A. Nguyen, and T. N. Nguyen, "Mining billions of ast nodes to study actual and potential usage of java language features," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 779–790.

[10] L. A. Meyerovich and A. S. Rabkin, "Empirical analysis of programming language adoption," in *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications*, 2013, pp. 1–18.

[11] B. G. Mateus and M. Martinez, "On the adoption, usage and evolution of kotlin features on android development," *arXiv preprint arXiv:1907.09003*, 2019.

[12] A. Warski, "The case against annotation," https://www.yegor256.com/2016/04/12/java-annotations-are-evil.html, 2016.

[13] Y. Bugayenko, "Java annotations are a big mistake," https://blog.softwaremill.com/the-case-against-annotations-4b2fb170ed67, 2017.

[14] Stack Overflow, "Xml configuration versus annotation based configuration," https://stackoverflow.com/questions/182393/xml-configuration-versus-annotation-based-configuration, 2008.

[15] Technical Jungle, "Annotations and its benefits in java," http://technicaljungle.com/annotations-in-java-intro-benefits-avoidance/, 2020.

[16] Stack Overflow, "Arguments against annotations," https://stackoverflow.com/questions/1675610/arguments-against-annotations, 2009.

[17] G. Riegler, "Be a better developer," http://www.beabetterdeveloper.com/2013/12/an-annotation-nightmare.html, 2013.

[18] H. Rocha and M. T. Valente, "How annotations are used in java: An empirical study." in *Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2011, pp. 426–431.

[19] P. Lima, E. Guerra, P. Meirelles, L. Kanashiro, H. Silva, and F. F. Silveira, "A metrics suite for code annotation assessment," *Journal of Systems and Software*, vol. 137, pp. 163–183, 2018.

[20] Z. Yu, C. Bai, L. Seinturier, and M. Monperrus, "Characterizing the usage, evolution and impact of java annotations in practice," *IEEE Transactions on Software Engineering*, 2019.

[21] Android Developers, "Android annotations," https://developer.android.com/reference/androidx/annotation/package-summary, 2020.

[22] —, "Data set: Annotation practices in android apps." https://figshare.com/articles/dataset/Annotation_practices_in_Android_apps/12782468, 2020.

[23] Android Developers, "The room persistence library," https://developer.android.com/jetpack/androidx/releases/room, 2020.

[24] Stack Overflow, "Use of custom annotations," https://stackoverflow.com/questions/31103302/use-of-custom-annotations, 2015.

[25] A. L. Santos and B. A. Myers, "Design annotations to improve api discoverability," *Journal of Systems and Software*, vol. 126, pp. 17–33, 2017.

[26] P. Pinheiro, J. C. Viana, M. Ribeiro, L. Fernandes, F. Ferrari, R. Gheyi, and B. Fonseca, "Mutating code annotations: An empirical evaluation on java and c# programs," *Science of Computer Programming*, vol. 191, p. 102418, 2020.

[27] I. Córdoba-Sánchez and J. de Lara, "Ann: A domain-specific language for the effective design and validation of java annotations," *Computer Languages, Systems & Structures*, vol. 45, pp. 164–190, 2016.

[28] I. Darwin, "Annabot: A static verifier for java annotation usage," *Advances in Software Engineering*, vol. 2010, 2010.

[29] F-Droid, "Free and open source android app repository," https://f-droid.org/, 2020.

[30] Oracle, "Annotation processing tool (apt)," https://docs.oracle.com/javase/7/docs/technotes/guides/apt/GettingStarted.html, 2020.

[31] P. Lima, "Annotation sniffer," https://github.com/phillima/asniffer, 2018.

[32] R. Pawlak, M. Monperrus, N. Petitprez, C. Noguera, and L. Seinturier, "Spoon: A library for implementing analyses and transformations of java source code," *Software: Practice and Experience*, vol. 46, no. 9, pp. 1155–1179, 2016.

[33] JavaParser, "Parser and abstract syntax tree for java," https://github.com/javaparser/javaparser, 2020.

[34] Guava, "Google core libraries for java," https://github.com/google/guava, 2020.

[35] R. P. Buse and W. R. Weimer, "Learning a metric for code readability," *IEEE Transactions on Software Engineering*, vol. 36, no. 4, pp. 546–558, 2009.

[36] ACRA, "A library enabling android applications to automatically post their crash reports to a report server." https://github.com/ACRA/acra, 2020.

[37] Google Web Toolkit, "A development toolkit for building and optimizing complex browser-based applications." http://www.gwtproject.org/overview.html, 2020.

[38] ACRA, "Javadoc acra." http://www.acra.ch/javadoc/, 2020.

[39] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Transactions on software engineering*, vol. 25, no. 4, pp. 557–572, 1999.

[40] andFHEM App, "An application to control devices using an fhem home automation server." https://github.com/klassm/andFHEM, 2020.

[41] W. Karim, "Intdef and stringdef in android," https://wajahatkarim.com/2018/05/intdef-and-stringdef-in-android/, 2018.

[42] A. P. L. Böttiger, "Soundwaves podcast player." https://play.google.com/store/apps/details?id=org.bottiger.podcast, 2020.

[43] Nextcloud App, "The android client for nextcloud." https://github.com/nextcloud/android, 2020.

[44] Y. Zeng, J. Chen, W. Shang, and T.-H. P. Chen, "Studying the characteristics of logging practices in mobile apps: a case study on f-droid," *Empirical Software Engineering*, vol. 24, no. 6, pp. 3394–3434, 2019.

[45] M. Nayebi, H. Farrahi, and G. Ruhe, "Analysis of marketed versus not-marketed mobile app releases," in *Proceedings of the 4th International Workshop on Release Engineering*, 2016, pp. 1–4.

[46] B. Xu, L. An, F. Thung, F. Khomh, and D. Lo, "Why reinventing the wheels? an empirical study on library reuse and re-implementation," *Empirical Software Engineering*, vol. 25, no. 1, pp. 755–789, 2020.

[47] R. Coppola, L. Ardito, and M. Torchiano, "Characterizing the transition to kotlin of android apps: a study on f-droid, play store, and github," in *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics*, 2019, pp. 8–14.

[48] Statista, "Number of app in google play store," https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/, 2020.