Revisiting Android Reuse Studies in the Context of Code Obfuscation and Library Usages

Mario Linares-Vásquez[1], Andrew Holtzhauer[1], Carlos Bernal-Cárdenas[2], Denys Poshyvanyk[1]
[1]The College of William and Mary, Williamsburg, VA, USA
[2]Universidad Nacional de Colombia, Bogotá, Colombia
mlinarev@cs.wm.edu, asholtzhauer@cs.wm.edu, cebernalc@unal.edu.co, denys@cs.wm.edu

ABSTRACT

In the recent years, studies of design and programming practices in mobile development are gaining more attention from researchers. Several such empirical studies used Android applications (paid, free, and open source) to analyze factors such as size, quality, dependencies, reuse, and cloning. Most of the studies use executable files of the apps (APK files), instead of source code because of availability issues (most of free apps available at the Android official market are not open-source, but still can be downloaded and analyzed in APK format). However, using only APK files in empirical studies comes with some threats to the validity of the results. In this paper, we analyze some of these pertinent threats. In particular, we analyzed the impact of third-party libraries and code obfuscation practices on estimating the amount of reuse by class cloning in Android apps. When including and excluding third-party libraries from the analysis, we found statistically significant differences in the amount of class cloning 24,379 free Android apps. Also, we found some evidence that obfuscation is responsible for increasing a number of false positives when detecting class clones. Finally, based on our findings, we provide a list of actionable guidelines for mining and analyzing large repositories of Android applications and minimizing these threats to validity.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement

General Terms

Measurement

Keywords

Android, class cloning, third-party libraries, obfuscated code, reuse

MSR'14, May 31 – June 1, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2863-0/14/05...$15.00
http://dx.doi.org/10.1145/2597073.2597109

1. INTRODUCTION

Developing mobile applications differs from desktop and web applications in several dimensions. It is not only about the revenue models or the size of the applications, but also about programming practices, hierarchy and structure of development teams, as well as other factors. For example, testing mobile applications highly benefits from crowdsource-based approaches that assume testing newly released applications (or updates) on different devices, operating systems, and under different connection modes (e.g., offline, WiFi, GSM). Also, reuse in mobile applications (hereinafter referred as apps) is ongoing and widespread, in particular because the apps are highly dependent on the APIs [24, 19, 18] and the distribution model in markets allows developers to sell apps several times by (re)packaging them with different GUI elements/data (e.g., the same travel guide for different cities).

As of today, only a handful of papers have analyzed mobile apps and their ecosystems to understand the factors that distinguish mobile apps and their development processes from desktop and web applications [5, 9, 12, 17, 18, 19, 24, 25]. For example, Minelli and Lanza [17] and Syer *et al.* [25] suggest that practices for desktop and server-based applications may not necessarily apply to mobile apps. Most of these related papers used a similar approach that consists of analyzing the code or metadata available in public markets. In the particular case of Android, APK (Application PacKage) files have been analyzed. It should be noted that these studies use executable files of the apps (APK files), instead of source code because of availability issues – most of the free apps available at the Android official market are not open-source, but can still be downloaded and analyzed in APK format. During the analysis, the APK files are converted to JAR files or decompiled to Java source code. *However, the building and packaging model of Android apps (APK files) may introduce some threats to validity of the results of empirical studies.*

As described in the Android developer guide [6], JAR libraries referenced by the source code of Android apps are imported into APK files at build time. Therefore, when the Android build system converts the .class files into a DEX file, a converter tool is called to extract .class files from JAR libraries and consider them as local .class files compiled from the application source code. Consequently, when converting APK files to JAR files or to Java source code, all the files are under the same root directory (app classes and third-party libraries), thus following the Java rules for organizing files under packages; in other words, there is no file system distinction between libraries and application-specific code. Even in the case of open source apps, according to [17, 24]

some apps include the source code of third-party libraries. In addition, obfuscation is a common practice recommended in the Android developer guide [7] to protect security protocols and other application components from reverse engineering attacks. Also, obfuscation is used to hide illegal reuse and avoid licensing issues [22, 10].

Including the code of third-party libraries in the APK files and ignoring obfuscation practices are threats to validity of empirical studies using APK files, in particular the ones aimed at analyzing class cloning/reuse in Android apps. For example, because of the build process, it is not possible to distinguish directly between code referenced as a library and code that was copied and modified from other applications or third-party libraries. Also, signature-based techniques for detecting class cloning, such as Software Bertilonage [3, 2], are sensitive to obfuscation, mainly to transformations such as renaming, ordering (e.g., changing order or methods, or changing order of parameters in methods), and aggregations (e.g., inline and outline methods, cloning methods) [1]. In general, the study by Schulze and Meyer showed that obfuscation by renaming identifiers reduces the effectiveness of text-based clone detectors [22].

Previous studies have not considered the impact of obfuscated code and third-party libraries on the measurements of class cloning in Android apps. Only a recent study by Mojica *et al.* [18] removed obfuscated classes from their dataset when computing the amount of class cloning on Android apps. Therefore, in this paper we provide empirical evidence on how third-party libraries and obfuscated code can impact reuse measurements. We computed the amount of classes reused on a large set of 24,379 free apps downloaded from Google play, including/excluding third-party libraries, and including/excluding obfuscated apps (that we detected using our algorithms). For detecting clones we used a signature-based approach as in [19, 18]. For detecting apps with obfuscated code we used a simple heuristic we defined after manually inspecting a large sample of obfuscated apps.

The results of this study show that there are significant and large differences, in terms of statistical significance and effect size, between the amount of class signatures reused in Android apps when including and excluding third-party libraries. Moreover, although the impact of obfuscated code is negligible when detecting cloned classes in Android apps, we found evidence of false positives declared as clones by the signature-based approach. Therefore, researchers analyzing/mining APK files should consider carefully when to include/exclude third-party libraries and obfuscated code, in particular for studies that use lexical information extracted from the files (i.e., identifiers) and signatures, or studies aimed at measuring similarities among apps.

Structure of the paper. Section 2 presents previous empirical studies that used Android apps. Section 3 defines our empirical study and the research questions, and provides details about the data extraction process and analysis method. Section 4 reports the study results, and discusses them from a quantitative and qualitative point of view. Section 5 discusses the threats that could affect the validity of the results. Section 6 concludes the paper and outlines directions for future work.

2. ANALYZING ANDROID APPS

Several recent papers have analyzed software evolution- and maintenance-related aspects in Android apps. Most of these studies used apps downloaded from Google Play and extracted bytecode from the APK files. The extraction process includes a transformation process from DEX to Java bytecode. This transformation process generates a set of .class files in a directory structure that follows the Java package guidelines. Therefore, the files belonging to third-party libraries and to the app's main package are organized using folders representing the packages hierarchy inside a single JAR file. In the following subsections we briefly describe the studies and summarize them in Table 1.

2.1 Reuse in the Android Market

Mojica Ruiz *et al.* [19] were the first to report on the volume of reuse in Android apps. Two dimensions of reuse were analyzed: reuse by inheritance and class reuse (from other applications). About 4,000 Android apps were manually downloaded from Google Play to measure the percentage of classes that were totally reused (cloned) by other apps and the top base classes that were inherited from third-party libraries and platform APIs (Android and Java). Mojica Ruiz *et al.* [19] analyzed the reuse by class cloning in Android apps, by using class signatures as proposed by Davies *et al.* [2, 3]. The main conclusion of their study is that almost 50% of the classes in the apps inherit from a base class, and most of the reused classes are in the Android APIs. The same study was recently extended in [18] with more than 200K apps from GooglePlay. The results on the extended study showed that about 84% of the classes are reused across all the categories of apps. However, both studies included the code belonging to third-party libraries when measuring the percentage of class cloned in the apps; and only the latter [18] considered the impact of obfuscated classes. Dresnos [5] also used method signatures to detect similar Android apps, where the signatures included string literals, API calls, exceptions, and control flow structures. However, the study does not report on the impact of obfuscated code or third-party libraries on their experiments.

Syer *et al.* [24] analyzed dependencies, source code, and churn metrics of three open source apps (i.e., Wordpress, Google Authenticator, and Facebook SDK) in Android and BlackBerry. Although they reported the findings in terms of apps dependency on predefined categories (e.g., language, user interface, platform, third-party), they analyzed different dimensions of reuse (i.e., inheritance, interface implementation, API calls) by counting the number of dependencies on each category and the proportion of platform and user interface dependencies out of the total number of dependencies. Their main conclusions were that Android apps require less source code but have larger files than in BlackBerry, and depend more on the Android APIs. During the analysis, the authors distinguished project-specific files from the source code of third-party libraries, and explicitly mentioned that *"apps often include, customize and maintain the source code of third party libraries"*[24].

Minelli and Lanza [17] proposed a visualization-based analysis for mobile apps using Samoa, which is an interactive tool that uses historical and structural information from the apps. Although the tool is not focused on a specific design aspect as reuse, the authors used the Average Hierarchy Height (AHH) and Average Number of Derived Classes (ANDC) metrics to study inheritance in Android apps. Moreover, they identified that some apps reuse libraries by copying the entire code instead of referencing JAR files. Some of the

Table 1: Recent studies of Android apps analyzed aspects or purpose, number of apps, and number of Android categories covered. We use NR to distinguish the cases where the number of domain categories is not reported. The last two columns list if the study considered the impact of third-party libraries (TPL) or the impact of obfuscated code (OBF): YES means the study considered the factor (NO is the opposite); NI stands for those cases where TPL and OBF factors do not impact the results.

Study	Purpose	#apps	#cat.	TPL	OBF
Shabtai *et al.* [23]	Apps categorization	2,285	2	NO	NO
Syer *et al.* [24]	Dependencies analysis	3	NR	YES	NI
Sanz *et al.* [21]	Apps categorization	820	7	NO	NO
Dresnos [5]	Detection of similar apps	2	1	NO	NO
Mojica Ruiz *et al.* [19]	Reuse by inheritance and code cloning	4,323	5	NO	NO
Minelli and Lanza [17]	Visualization based analysis	20	NR	NI	NI
Mojica Ruiz *et al.* [18]	Reuse by inheritance and code cloning	> 200K	30	NO	YES
Syer *et al.* [25]	Size, dependencies and defect fix time	15	NR	NO	NI
McDonnell *et al.* [14]	API instability and adoption	10	7	NI	NI
Linares-Vásquez *et al.*[12]	Apps success and API change/bug proneness	**7**,097	30	NI	NI

findings help to describe the programming model of Android apps (e.g., complexity of mobile apps is mostly attributed to the dependency on third-party libraries), however, only 20 open source apps were used in the study. Although the authors recognize the fact that the source code of third-party libraries is copied in some cases into the apps, they do not mention explicitly if the tool (Samoa) distinguishes between project-specific and third-party library files.

2.2 Other Studies Using Android Apps

Syer *et al.* [25] analyzed 15 open source apps to investigate the differences of mobile apps with five desktop/server applications. The comparison was based on two dimensions: the size of the apps and the time to fix defects. The study suggests that mobile apps are similar to UNIX utilities in terms of size of the code and the development team. However, it is not clear if the analyzed apps included the source code of third-party libraries. Also, the findings suggest that mobile app developers are concerned with fixing bugs quickly: over a third of the bugs are fixed within one week and the rest are fixed within one month.

Categorization of Android applications has been explored using machine-learning techniques [21, 23]. Shabtai *et al.* [23] categorized APK files into two root categories of the Android market ("Games" and "Applications") using attributes extracted from DEX files and XML data in the APK files. Sanz *et al.* [21] used string literals in classes, ratings, application sizes, and permissions to classify 820 applications into several existing categories. In both cases [21, 23], some of the extracted features could be obfuscated and could also belong to third-party libraries. Therefore it is possible that the results of the study were impacted by the effect of obfuscated code and third-party libraries.

McDonnell *et al* [14] analyzed the evolution of Android APIs (i.e., frequency of changes) and the reaction of client code to API evolution. For the latter purpose, they analyzed 10 open-source Android applications from 7 domains to investigate into: (i) degree of dependency on Android APIs; (ii) lag time between a client API reference and its most recent available version; (iii) adoption time of new APIs; (iv) the relation between API instability and adoption; and (v) the relationship between API updates and bugs in client code. Also, Linares-Vásquez *et al.* [12] analyzed the impact of the Android APIs change- and fault-proneness on the success of 7,097 apps from Google Play. In both studies [12, 14], because they analyzed calls to the Android API, there was not an impact on the results by the effect of third-party libraries or obfuscated code.

3. STUDY METHODOLOGY

The *goal* of this study is to understand to what extent obfuscated code and third-party libraries could affect the studies on reuse by class cloning. The *context* consists of 24,379 free Android apps from the Google Play Market, and the *perspective* is that of researchers interested in defining guidelines for empirical studies based on Android apps. Table 2 reports characteristics of the apps that we analyzed. For each category considered in our study (e.g., photography, medical, games, etc), the table lists (i) the number of apps analyzed from the category (column #apps), (ii) the size range of the analyzed apps in terms of number of classes (column #classes), and size in terms of thousands of lines of code including third-party libraries (KLOC).

3.1 Research Questions

In the context of our study, we formulated the following research questions:

- **RQ$_1$**: *Do third-party libraries impact the measurement of class cloning?* This research question aims at investigating if the amount of class cloning in Android apps is mainly due to the dependability on the third-party libraries or the apps' classes. Specifically, we test the following null hypothesis:

Table 2: Characteristics of the apps (grouped by category) used in our study.

Category	#apps	Classes	KLOC
Arcade	826	5-566	625-20K
Books and reference	719	5-73	7K-639K
Brain	1021	5-572	5K-16K
Business	2047	5-551	64K-105K
Cards	495	8-633	30K-60K
Casual	840	6-566	60K-77K
Comics	57	10-392	251-20K
Communication	479	5-11	419-667K
Education	1572	5-119	9K-58K
Entertainment	2809	2-11	850-61K
Finance	586	5-1583	220-9K
Health and fitness	310	6-104	8K-26K
Libraries and demo	244	1-499	32K-338K
Lifestyle	1621	2-572	7K-16K
Media and video	644	5-572	8K-35K
Medical	102	5-105	6K-26K
Music and audio	1562	3-683	8K-14K
News and magazines	1015	5-280	26K-96K
Personalization	1055	2-126	12K-54K
Photography	595	6-155	111-31K
Productivity	639	5-111	11K-34K
Racing	456	15-280	26K-169K
Shopping	200	5-7	138-151K
Social	522	5-318	48K-122K
Sports	1158	5-280	7K-16K
Sports games	498	6-572	26K-52K
Tools	1421	4-65	7K-58K
Transportation	149	6-57	10K-202K
Travel and local	681	5-257	6K-16K
Weather	56	16-30	2K-22K
Total	**24,379**	**1-1583**	**111-667K**

H_{0_1}: There is no significant difference between the amount of cloned classes in Android apps when considering third-party libraries and when excluding those libraries from the analysis.

- **RQ_2**: *Does obfuscated code impact the measurement of class cloning?* This research question aims at investigating if obfuscated apps should be considered when computing the amount of classes reused between Android apps. Specifically, we test the following null hypothesis:

  H_{0_2}: There is no significant difference between the amount of cloned classes in Android apps when considering obfuscated apps and when excluding those apps from the analysis.

The **dependent variable** for both research questions is represented by the amount of reuse by class cloning, which is estimated as the *Proportion of Class Signatures Reused (PCSR)* per category in our dataset (Section 3.3).

The **independent variable** for **RQ_1** is the set of .class files of the apps under study including third-party libraries, and excluding those libraries. For **RQ_2** the **independent variable** is the set of .class files of the apps under study including and excluding obfuscated apps.

3.2 Data Extraction Process

We downloaded (randomly) free mobile apps from Google Play as APK files, then we converted the APK files into JAR files using the following procedure: (i) unzip APK files by using the *apktool*[1] tool, which reveals the compiled Android application code file (note that an APK is just a set of zipped DEX files); then (ii) translate DEX files from the Dalvik bytecode to Java bytecode files (i.e., .class) using the *dex2jar*[2] tool (see Figure 2).

3.2.1 Reuse by Class Cloning Detection

For computing reuse via class cloning we relied on the Software Bertillonage technique [3, 2] to identify when a class is cloned across several apps, by comparing the classes' signatures. We built class signatures using the Apache Commons BCEL Java library[3] as in [3, 2]. Consequently, a class signature is a file with three parts: **class header**, **attributes signatures** sorted alphabetically, and **methods signatures** sorted alphabetically. The format of each part is as follows:

- The **class header** is defined by the following expression: <*modifiers*> <*class_name*> extends <*base_class*> implements <*interfaces_separated_by_comma*>. We avoided including the java.lang.Object class in the list of base classes.

- Each **attribute signature** is defined by the following expression: <*modifiers*> <*attribute_type*> <*attribute_name*>.

- Each **method signature** is defined by the following expression: <*modifiers*> <*return_type*> <*method_name*> (<*argument_types*>).

Similarly to [3, 2, 19, 18], the parts corresponding to the base class and interfaces are optional in the class header, and the names of classes/types do not include the package in any of the parts. Figure 1 presents an example of a class signature.

In order to detect reuse by class cloning, we needed to find if any signature file's contents were exactly the same as the contents of another signature file. Even if we used certain optimizations on our comparisons to prevent redundant comparisons, it would be extremely time-consuming to repeatedly compare files directly. In order to overcome this obstacle, we opted to read in each signature file, and created an MD5 hash from each class signature. We created a large hash map which used the MD5 hash as the key, and contained a list of signature names for the value. For every signature file, we checked if the hash already existed as a key in the hash map. If it did, we appended the name of the signature file to the end of the list in the respective value. If not, we added the key/value pair to the hash map. Thus, once we finished adding every signature file's hash and name to the map, we were able to distinguish the cloned files from the originals.

3.2.2 Detecting Obfuscated Apps

One of the authors manually inspected the source code (after decompilation) of 120 apps (i.e., two obfuscated and two non-obfuscated apps per category) to identify patterns in the identifiers of obfuscated classes. To decompile the apps we extracted .class files from the JAR files by using the *7zip*[4]

[1] http://code.google.com/p/android-apktool/
[2] http://code.google.com/p/dex2jar/
[3] http://commons.apache.org/proper/commons-bcel/
[4] `http://www.7-zip.org/`

```
public ZzActivity extends Activity implements View
private Button button1
public static int count
public int radioCheck
default static void <clinit>()
public void <init>()
public void onCheckedChanged(RadioGroup,int)
public void onClick(View)
public void onCreate(Bundle)
public boolean onKeyDown(int,KeyEvent)
```

Figure 1: Class signature example for the class zz.zzz.ZzActivity in the Android zz.zzz App.

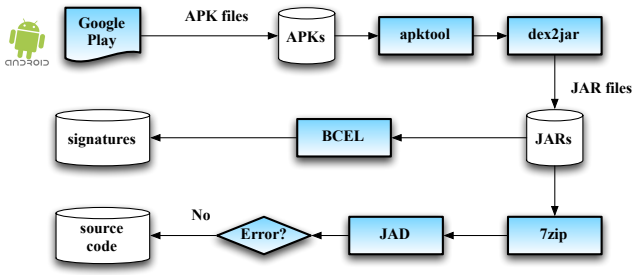

Figure 2: Source code and JAR files extraction process from APK files

tool and then we decompiled the .class files to Java source code using the JAD decompiler[5]. During decompilation, we discarded any apps that did not decompile correctly (see Figure 2). At the end, we were able to decompile 24,379 apps successfully.

After decompiling and manually inspecting the apps, we found that all the apps with obfuscated identifiers always have a class `a.java`, because of the renaming algorithm of the obfuscation tool used for Android apps transforms identifiers using a lexicographic order. Therefore, to detect apps with obfuscated identifiers we looked for apps with a class `a.java` in the main package. We decided to use this simple heuristic because we were interested only in the impact of identifier obfuscation in the class cloning estimation using signatures. Using this method we found 415 apps with obfuscated code. The distribution of apps with obfuscated code per category is depicted in Figure 3.

To validate the accuracy of the method, another author of the paper manually verified the true positive rate (TPR) and false positive rate (FPR) of the heuristic for detecting obfuscated classes, by using a validation set of apps. The validation set was built using the following guidelines:

- The apps were sampled by one of the authors (not the same author performing the validation)
- The validation set includes two apps classified as obfuscated and two apps classified as non-obfuscated for each category (i.e., 120 apps).
- The apps in the validation set were different from the ones inspected manually for identifying patterns in the identifiers of obfuscated classes

For the validation we followed these definitions[6]:

[5] `http://www.varaneckas.com/jad/`

[6] We were interested in the correctness of the heuristic for classifying apps in the positive set (i.e., obfuscated), and in the negative set (i.e., non-obfuscated), and in the general accuracy of the heuristic. Therefore, we used TPR, FPR, and ACC instead of precision.

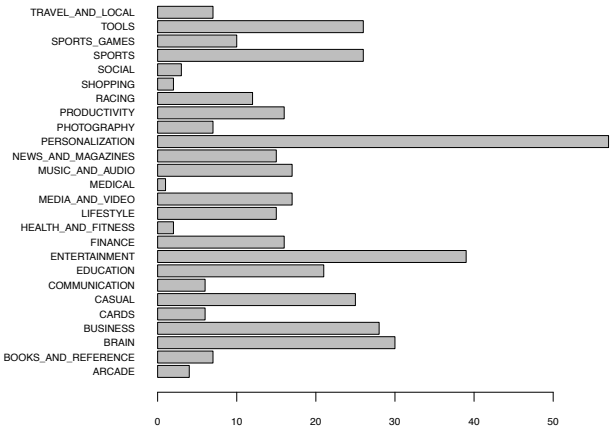

Figure 3: Distribution of obfuscated apps per category.

- **True positives (TP):** number of obfuscated apps classified correctly by the heuristic
- **True negatives (TN):** number of non-obfuscated apps classified correctly by the heuristic
- **False positives (FP):** number of non-obfuscated apps classified incorrectly by the heuristic (i.e., classified as obfuscated)
- **False negatives (FN):** number of obfuscated apps classified incorrectly by the heuristic (i.e., classified as non-obfuscated)
- **True positive rate (TPR), a.k.a., recall:** $TP/(TP+FN)$
- **True negative rate (TNR):** $TN/(FP+TN)$
- **Accuracy (ACC):** $(TP+TN)/(TP+TN+FP+FN)$

The results of the manual validation were 60 true positives and 60 true negative, which accounts for a TPR equal to 1, a TNR equal to 1, and an ACC equal to 1. Therefore, our simple heuristic for detecting obfuscated apps is accurate and correct in a sample of 120 apps, which ensures a confidence interval of 8.93% with a confidence level of 95%.

3.3 Analysis Method

For measuring the amount of reuse by class cloning we used the *Proportion of Class Signatures Reused* ($PCSR$) proposed by Mojica Ruiz *et al.* [18, 19]. $PCSR$ calculates the proportion of class signatures that are clones (i.e., they appear in multiple apps belonging to a set of apps). Given a set of apps A, the number of Unique Class Signatures (UCS) in an app $a_i \in a$ ($a \subset A$), and C the set of all class signatures of the apps in a, the $PCSR$ of a subset of apps a is defined as follows:

$$PCSR(a, A) = 1 - \frac{\sum_{i=1}^{|a|} UCS(a_i, \{A - a_i\})}{|C|} \quad (1)$$

We defined a unique class signature in a_i as a signature that does not appear in the rest of apps in A ($\{A - a_i\}$ in equation 1). Thus, the higher the $PCSR$, the higher the reuse in a subset of apps a (e.g., apps in the category *Arcade*)

when compared to all the apps in A (e.g., all the apps in our dataset). Consequently, in order to compare the impact of third-party libraries on the measurement of reuse by class cloning (RQ_1) we computed the $PCSR$ per category (i.e., $PCSR$ of class signatures of apps belonging to a specific category that are cloned in all the 24,379 apps) including the class signatures of the third-party libraries ($PCSR_{+TPL}$); we also computed the $PCSR$ per category excluding class signatures of the third-party libraries ($PCSR_{-TPL}$). To compare the impact of obfuscated apps on the measurement of reuse by class cloning (RQ_2) we computed $PCSR$ per category excluding obfuscated apps ($PCSR_{-OBF}$), and excluding classes signatures of third-party libraries and obfuscated apps ($PCSR_{-(TPL,OBF)}$).

To validate that the results of our research questions are statistically significant in the 30 categories of Google play we used the Mann-Whitney test [4]. We compared $PCSR_{+TPL}$ to $PCSR_{-TPL}$ for H_{0_1}; and $PCSR_{+TPL}$ to $PCSR_{-OBF}$, and $PCSR_{-TPL}$ to $PCSR_{-(TPL,OBF)}$[7] for H_{0_2}. We also computed the Cliff's delta d effect size [8] to measure the magnitude of the difference in the three cases. We followed the guidelines in [8] to interpret the effect size values: negligible for $|d| < 0.147$, small for $0.147 \leq |d| < 0.33$, medium for $0.33 \leq |d| < 0.474$ and large for $|d| \geq 0.474$. We are not assuming population normality and homogeneous variances, therefore we choose non-parametric methods (Mann-Whitney test and Cliff's delta). To distinguish app-specific classes from third-party-library classes, we extracted the package name (i.e., main package) from the `AndroidManifest.xml` file that resided with every application we downloaded. Then, we considered all the classes inside the main package and its sub-packages as app-specific classes; classes outside the main package were considered as classes from third-party libraries.

3.4 Replication Package

The data set used in our study is publicly available at http://www.cs.wm.edu/semeru/data/MSR14-android-reuse/. In particular we provide: (i) the list (and URLs) of the studied 24,379 apps; (ii) the list of apps labeled manually as obfuscated and non-obfuscated; (iii) the dataset used for training the classifiers; and (iv) the results of the classification process and the manual validation.

4. ANALYSIS OF THE RESULTS

This section reports the results aimed at answering the two research questions formulated in Section 3.1. Table 3 summarizes the results for RQ_1 and RQ_2. In particular, the table lists the number of the proportion of class signatures reused ($PCSR$) in our dataset per category, when considering third-party libraries ($+TPL$), excluding third-party libraries ($-TPL$), excluding obfuscated apps ($-OBF$),and excluding third-party libraries and obfuscated apps (i.e., *-(TPL, OBF)*); Table 3 also lists the differences between the $PCSR$ values ($\Delta PCSR$). In addition, Figure 4 depicts the change ratio (i.e., reduction) of number of cloned signatures detected in the 30 categories, when comparing the initial dataset to the dataset without third-party libraries, and when comparing the dataset without third-party libraries to the dataset without libraries and without obfuscated apps.

[7]Note that the apps used for computing $PCSR_{+TPL}$ and $PCSR_{-TPL}$ include obfuscated apps.

4.1 Impact of Third-Party Libraries

Excluding the libraries from the PCSR computation reduces notoriously the number of classes detected as clones. On average, 87.66% less signatures are detected as clones (see Figure 4 boxplot *+TPL to -TPL*), with a median of 90.70%, a minimum reduction of 67.08% (in the category *Health and fitness*), and a maximum reduction of 97.48% (in the category *Casual*). A similar behavior (i.e., reduction in all the categories) is reflected in the $PCSR$ computation (see Table 3). The average reduction of $PCSR$ when comparing $PCSR_{+TPL}$ to $PCSR_{-TPL}$ is 37.30%, with a median of 37.94%, a minimum reduction of 7.45% (in the category *Business*), and a maximum reduction of 71.82% (in the category *Finance*).

That reduction in the number of class signatures detected as clones is large and significant. The Mann-Whitney test applied to the PCSR of signatures including third-party libraries ($PCSR_{+TPL}$) and the $PCSR$ of signatures excluding third-party libraries ($PCSR_{-TPL}$) reports a p-value= 9.123e-13, and the Cliff's delta was 0.9267 with a 95% confidence interval [0.8083, 0.9730]. Therefore, we can reject our null hypothesis H_{0_1}, that is, there is statistically significant difference between the two groups, and the magnitude of the difference is large (Cliff's delta > 0.474).

The significant reduction of the $PCSR$ and the number of clones when excluding the signatures of third-party libraries from the analysis shows that most of the clones are detected in the signatures of the libraries, and it suggests that most of the code in APK files belongs to the libraries. Figure 5 depicts the change ratio (i.e., reduction) of the number of class signatures when comparing the datasets including and excluding third-party libraries. On average, 82% of the signatures are reduced when excluding third-party libraries, with a median reduction of 81.23%, a minimum reduction of 63.82% (in the case of apps in the Category *Medical*), and a maximum reduction of 93.13% of the signatures (in the category *Arcade*).

We also analyzed which third-party library class signatures appeared most often, and attributed them to their respective third-party libraries. By doing so, we found that the most common third-party library class signature is `com.google.ads.AdActivity` of the `com.google.ads` package. This class is found in 8,008 apps from our dataset, and is by far the most common third-party library class. The second most common is `com.facebook.android.FacebookError`, from the `com.facebok.android` package. This class signature was found in 6,652 apps. Finally, the third most common is `org.mcsoxford.rss.Dates` (and 22 other classes from this same package), which all appeared 4,880 times each.

A significant number of apps in our dataset utilize the Google Ads third-party library, potentially as a source of revenue due to all the apps in our dataset being free. Also free apps have the option for Facebook integration. Finally, the commonality of `org.mcsoxford.rss` demonstrates that many apps try to integrate with RSS feeds, and this third-party library is described as a "lightweight Android library to read parts of RSS 2.0 feeds." [8]

[8]https://github.com/ahorn/android-rss

Table 3: Summary of results for RQ_1 and RQ_2. The PCSR and the difference between PCSR are listed by category.

Category	$PCSR$				$\Delta PCSR$ in percentage		
	$(1) + TPL$	$(2) - TPL$	$(3) - OBF$	$(4) - (TPL, OBF)$	$\frac{(1)-(2)}{(1)}$	$\frac{(1)-(3)}{(1)}$	$\frac{(2)-(4)}{(2)}$
Arcade	0.879688807	0.351430128	0.829749587	0.353	60.05%	5.68%	-0.49%
Books and reference	0.902051778	0.764008747	0.865465015	0.766	15.30%	4.06%	-0.32%
Brain	0.887832825	0.430272179	0.856644088	0.443	51.54%	3.51%	-2.98%
Business	0.87552092	0.81029478	0.83328423	0.819	7.45%	4.82%	-1.03%
Cards	0.839701923	0.383950449	0.803221733	0.394	54.28%	4.34%	-2.52%
Casual	0.879285526	0.316837771	0.848328398	0.334	63.97%	3.52%	-5.29%
Comics	0.920239358	0.569789675	0.895015907	0.570	38.08%	2.74%	0.00%
Communication	0.721917416	0.267079672	0.698035799	0.278	63.00%	3.31%	-4.02%
Education	0.930289647	0.752647301	0.89829617	0.758	19.10%	3.44%	-0.70%
Entertainment	0.926801039	0.778399296	0.890272919	0.782	16.01%	3.94%	-0.45%
Finance	0.73930074	0.2083375	0.671912979	0.214	71.82%	9.12%	-2.72%
Health and fitness	0.942137572	0.858483567	0.910759035	0.858	8.88%	3.33%	0.06%
Libraries and demo	0.945508264	0.58816772	0.906561089	0.588	37.79%	4.12%	0.00%
Lifestyle	0.895974257	0.670682596	0.854727031	0.673	25.14%	4.60%	-0.39%
Media and video	0.82493961	0.350162866	0.778602469	0.356	57.55%	5.62%	-1.57%
Medical	0.892509122	0.79831534	0.868709734	0.797	10.55%	2.67%	0.14%
Music and audio	0.876889856	0.772113587	0.872644424	0.782	11.95%	0.48%	-1.30%
News and magazines	0.898420806	0.570090694	0.863682988	0.579	36.55%	3.87%	-1.53%
Personalization	0.916485781	0.579639994	0.882627703	0.597	36.75%	3.69%	-2.97%
Photography	0.841369352	0.488375841	0.802766729	0.493	41.95%	4.59%	-0.93%
Productivity	0.760950939	0.372573998	0.697382465	0.386	51.04%	8.35%	-3.65%
Racing	0.92630846	0.544961203	0.914853858	0.570	41.17%	1.24%	-4.66%
Shopping	0.778630803	0.241201949	0.722952071	0.248	69.02%	7.15%	-2.80%
Social	0.891405177	0.769676122	0.874447338	0.771	13.66%	1.90%	-0.20%
Sports	0.913609539	0.70980359	0.8654784	0.721	22.31%	5.27%	-1.52%
Sports games	0.928972353	0.515475313	0.900676647	0.520	44.51%	3.05%	-0.81%
Tools	0.798852806	0.423019698	0.719613626	0.439	47.05%	9.92%	-3.86%
Transportation	0.818652745	0.396924049	0.755777412	0.397	51.51%	7.68%	0.00%
Travel and local	0.913801067	0.727011219	0.859122395	0.727	20.44%	5.98%	0.06%
Weather	0.949381457	0.660390516	0.90783172	0.660	30.44%	4.38%	0.00%

Summarizing, the results of our $\mathbf{RQ_1}$ show that *considering third-party libraries when computing class cloning in Android apps impacts the results, in the sense that because of the wide usage of third-party libraries, a significant number of clones are detected between the apps. Therefore, an actionable guideline when analyzing APK files is: consider carefully if third-party libraries should be included or not in the specific analysis; in particular, when analyzing class cloning between Android apps, researchers should justify the decision of including/excluding third-party libraries libraries in the class cloning measurements.*

4.2 Impact of Obfuscated Apps

Excluding obfuscated apps also reduced the number of signatures detected as clones, and consequently $PCSR$. The Mann-Whitney test applied to the PCSR of signatures including third-party libraries ($PCSR_{+TPL}$) and the $PCSR$ of signatures excluding obfuscated apps ($PCSR_{-OBF}$) reports a p-value= 0.009604, and the Cliff's delta was 0.3866667 with a 95% confidence interval [0.08998, 0.62031998]. Therefore, we can reject our null hypothesis H_{0_2}, i.e. there is a statistically significant difference between the two groups, and the magnitude of the difference is medium ($0.33 \leq |d| < 0.474$).

On average (see Table 3) there is a reduction of 4.55% in the $PCSR$, with a median of 4.09%, a minimum reduction of 0.48% (*Music and Audio*), and a maximum reduction of 9.92% (*Tools*). This reduction is explained due to the number of signatures belonging to obfuscated apps (we found 415 obfuscated apps out of 24,415). When excluding obfuscated apps (see Figure 5) 8.25% of the signatures were reduced on average (median = 7.31%, min. = 0%, max=21.38%), which represented an average reduction in the number of signatures detected as clones (see Figure 4) of 12.40% (median = 11.43%, min. = 2.74%, max=23.98%).

However, when comparing the impact of obfuscated code in the $PCSR$ excluding the signatures of third-party libraries (i.e., -TPL to -(TPL, OBF)) the Mann-Whitney reports a p-value=0.8187, and we obtained a Cliff's delta = -0.0356. In this case there is no significant difference (p-value > 0.05) and the magnitude of the difference is negligible ($|d| < 0.147$). When removing the obfuscated apps from the set of signatures that does not include third-party libraries there is an average reduction in the number of signatures detected as clones of 0.63% (median = 0%, min. = 0%, max=3%), and an average reduction in the number of signatures of 2.23% (median = 1%, min. = 0%, max=8%). However, in most of the categories (23 out of 30) removing the obfuscated apps increases the $PCSR$ (see Table 3). For example, there is a change in the $PCSR$ of the category *Casual* from 0.3168 ($PCSR_{-TPL}$) to 0.334 ($PCSR_{-(TPL,OBF)}$), which accounts for an increment of 5.29%.

An explanation for those cases is the impact of the reduction of the signatures in the $PCSR$ computation. Equation

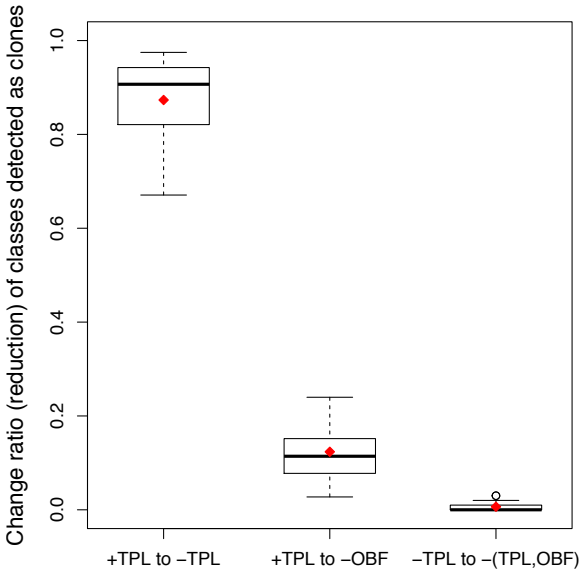

Figure 4: Boxplots for the change ratio of number of clones (signatures) when (1) comparing the dataset with third-party libraries and without third-party libraries (i.e., +TPL to -TPL); (2) comparing the dataset with third-party libraries, and the dataset without obfuscated apps (i.e., +TPL to -OBF); and (3) comparing the dataset without third-party libraries, and the dataset without third-party libraries and without obfuscated apps (i.e., -TPL to -(TPL, OBF)). Red diamonds represent the mean (average).

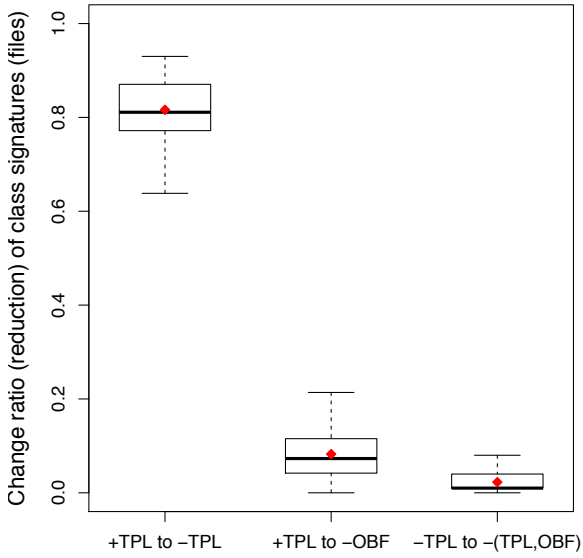

Figure 5: Boxplots for the change ratio of number of signatures when (1) comparing the dataset with third-party libraries and without third-party libraries (i.e., +TPL to -TPL); (2) comparing the dataset with third-party libraries, and the dataset without obfuscated apps (i.e., +TPL to -OBF); and (3) comparing the dataset without third-party libraries, and the dataset without third-party libraries and without obfuscated apps (i.e., -TPL to -(TPL, OBF)). Red diamonds represent the mean (average).

1 is equivalent to the ratio between the number of signatures detected as clones and the total number of signatures. In the case of apps in the category *Casual* for $PCSR_{-TPL}$ there were 11,851 signatures detected as clones out of 37,404 signatures ($PCSR_{-TPL} = 11,851/37,404 = 0.3168$), and for $PCSR_{-(TPL,OBF)}$ there were 11,539 signatures detected as clones out of 34,589 signatures ($PCSR_{-(TPL,OBF)} = 11,539/34,589 = 0.334$). That increment of 5.29% in the $PCSR$ is explained in the fact that proportionally the reduction of the signatures is bigger compared to the reduction of clones, which means that most of the clones were detected between the non-obfuscated apps. However, there were some signatures detected as clones between the obfuscated apps.

Regarding detecting cloned classes in the dataset of apps tagged as obfuscated, we inspected manually the signatures and we found that there are some false positives. That is, there are classes that are marked as clones of other classes based on their class signatures, but further analysis of the content of the class demonstrated that this is not always true. We were able to find multiple examples of this occurring fairly easily, and we believe that there could be many more false clone detections in our obfuscated dataset as a result of this observation. The first example comes from the apps with package names *bagins.football* and *com.antivirus*. In both apps we found two obfuscated classes that were detected as cloned signatures: `/bagins/football/c/c.java` and `/com/ antivirus/core/b/ c.java`. Both of these files have the same signatures and thus method names, but these methods do different things. For instance, in *bagins.football* the values() method creates a new array and performs a `System.arraycopy` into it, whereas in *com.antivirus* the method only has a statement returning a casted variable with `.clone()`.

Another example we found is between the apps *com.agilesoft resource* and *com.ableon.team.barcelona*. Both apps have a class called `h.java` inside their main package, and both classes have a `void run()` method. However, the run function in `h.java` of *com.agilesoftresource* is simply a one-line statement:

```
"AppManagerMain.a(AppManagerMain.e(g.a(a))).
        refreshPackList();"
```

whereas the `run` method of *com.ableon.team.barcelona* is 15 lines long and makes calls to the `javax.microedition.khronos.egl` API and performs an obfuscated conditional:

```
"if(a.isVisible() && g.c(a).eglGetError() = 12302)"
```

Some cloned classes appear in more than two apps. One such example is the class `as.java`, which appears in three apps: *balofo.game.movie*, *com.application.fotodanz*, and *com. advancedprocessmanager*. For each app, this class has the `onClick()` method, but the code it executes is unique in each case. For *balofo.game.movie*, the method simply performs a `dialoginterface.cancel()`; for *com.application.fotodanz*, the method executes no code; and for *com.advancedprocess manager*, we get an "obfuscated" one-line of code:

```
"ak.a(aq.a(a)).a()"
```

The examples described before show how using class signatures to detect clones between obfuscated classes is not accurate because it is prone to false positives. However, there are also cases of true positives. We have noted that if apps share a main package or developer "keyword" in the app's package name then it is likely that the files are indeed clones. For instance, *com.appmakr.app247821* and *com.appmakr.app153560* both have a class called `c.java` that were located within different directories inside each main package respectively, but were still detected as clones. Due to each app's package name sharing the term *appmakr*, we assume it's likely for these two files to be legitimate clones; upon further inspection, each file is 43 lines long and both files are the exact same, character for character, except for one line which references the main package name (*app247821* or *app153560* respectively). Therefore, these files are correctly detected as legitimate clones.

Finally, we should note that because we only tried to find cloned classes that reside within the main package of the app, we've extracted the package name from the `AndroidManifest.xml` file that resided with every application we downloaded. Thus, sometimes a cloned class may appear to lie in a package different from the source directory, but is in fact within the proper main package. For instance, another cloned class was `q.java` which appears in apps *com.atomimbh.app*, *com.BeltzandRuth*, and *com.bangladeshfreegoimbh.app*. We noted that both the first and last apps in this list seem to follow the trend of having a shared keyword (`*imbh.app`), but one of the apps doesn't follow this pattern. However, for this app the actual location of this cloned class is found in *com.BeltzandRuth/src/com/bemyvalentineimbh/app/*, which does share the similar keyword as the other two apps. Upon analyzing the Android manifest for this app, the main package is indeed *com.bemyvalentineimbh.app*. Thus, upon further inspection of the `q.java` class, we noted that all three apps have a similar implementation of both methods inside the class, where both *com.BeltzandRuth* and *com.bangladeshfreegoimbh.app* contained exactly the same implementations, and *com.atomimbh.app* contained the exact implementation of one method and a functionally similar implementation of another method (only a few lines had their order changed).

Summarizing, the results of our **RQ**$_2$ show that *considering obfuscated apps when computing class cloning in Android apps impacts the results, in the sense that signatures in obfuscated classes introduce false positives in the cloned signatures detection. Although the impact of obfuscated code is not as significant as the impact of considering third-party libraries in the cloned signatures detection, researchers should be careful when considering obfuscated code in their experiments using APK files. Therefore, an actionable guideline when analyzing APK files is: consider carefully if obfuscated apps (or obfuscated code) should be included or not in the specific analysis; in particular, when analyzing class cloning between Android apps, researchers should justify the decision of including/excluding obfuscated code in the measurements.*

5. THREATS TO VALIDITY

Threats to construct validity concern the relationship between theory and observation, and it is essentially due to the measurements/estimates on which our study is based. We assumed that class signatures are representative of the actual source code files as in previous studies [2, 3, 18, 19]. However, we cannot state that the code inside the source files is exactly the same based solely on matching signatures. Instead, the methods may have been named similarly or may have had the same parameters. Therefore, it is likely that there is much more source code reuse occurring that we have been unable to detect in the case of class cloning. As this is an initial study of reuse, for future work we plan to obtain more exact results, by considering also the source code.

Threats to *conclusion validity* concern the relationship between treatment and outcome. Our conclusions are supported by appropriate, non-parametric statistics (Mann-Whitney test). In addition, the practical significance of the observed differences is highlighted by effect size measures (Cliff's d).

Threats to internal validity concern factors that can affect our results. Our heuristic for identifying obfuscated apps could fail if the renaming strategy did not follow a lexicographic order (i.e., the first letted used to obfuscate identifiers is *a*) or the obfuscation is different to renaming transformation. However, we manually inspected a sample of apps classified by the heuristic and we obtained a true positive and true negative rates equals to 1, which represents an accuracy of 100%.

Threats to *external validity* concern the generalization of our findings. Our analysis is limited to Android free apps. Dependency of commercial apps on third-party libraries could be different, for example, libraries for advertisements might not be widely used in commercial apps. Also, it is possible that the commercial apps have more obfuscated code. Therefore, our findings may not necessarily hold for commercial apps. Regarding the size of our dataset (24,379 apps), the set of analyzed apps is a small percentage of the existing apps in Google Play (more than 1 million of apps reported by the AppBrain website[9]). However, our sample covers all the domain categories in Google Play with a significant number of apps compared to other studies using Android apps (see Table 1). In future studies, we are also planning on using diversity measures to guide the selection of apps to maximize generalizability of the case studies [20]. Finally, our conclusions may not be valid for apps developed for other mobile platforms (e.g., iOS).

6. CONCLUSION AND FUTURE WORK

Although APK files have been used in several studies for analyzing Android apps and their development processes, the building process used to generate those files introduces some threats to the validity of the results in the studies. In particular, we analyzed 24,379 APK files downloaded from Google Play to measure the impact of third-party libraries and obfuscated code on class cloning measurement. We found that excluding third-party libraries reduces on average 87.66% of the signatures detected as clones, and the difference is large and statistically significant when comparing the proportion of class signatures reused (PCSR) in our dataset including and excluding the libraries. Concerning the impact of obfuscated files, it is significantly different but the difference is medium on the computation of the PCSR. We found a few of the obfuscated apps and evidence of false positives detected as clones by the signature-based method (Software Bertilonage).

[9] `http://www.appbrain.com/stats/number-of-android-apps`

Future studies with significantly higher number of obfuscated apps should analyze the impact of those apps on the results.

Our findings show that empirical studies using APK files should take into account possible impacts of third-party libraries and obfuscated code. Therefore, we suggested two actionable guidelines when analyzing/mining APK files:

1. *Consider carefully if third-party libraries should be included or not in the specific analysis; in particular, when analyzing class cloning between Android apps, researchers should justify the decision of including/ excluding third-party libraries libraries in the class cloning measurements.*

2. *Consider carefully if obfuscated apps (or obfuscated code) should be included or not in the specific analysis; in particular, when analyzing class cloning between Android apps, researchers should justify the decision of including/excluding obfuscated code in the class cloning measurements.*

These actionable guidelines are also pertinent to studies/approaches on software categorization [11, 13, 16], in which the lexical information in bytecode or source code is used to categorize the apps; given the widespread use of third-party libraries, such as Google Ads or Facebook for Android using the identifiers extracted from those libraries can reduce the variance and consequently impact the categorization process. In addition, studies aimed at identifying similar apps [15], which use non-textual based detection, should also consider the impact of third-party libraries and obfuscation practices.

7. ACKNOWLEDGEMENTS

We would like to thank Bogdan Dit for his help on previous drafts of this paper. This work is supported in part by the NSF CCF-1016868 and NSF CAREER CCF-1253837 grants. Any opinions, findings, and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors.

8. REFERENCES

[1] C. Collberg, C. Thomborson, and D. Low. A taxonomy of obfuscating transformations. Technical Report 148, Department of Computer Science, The University of Auckland, 1997.

[2] J. Davies, D. M. German, M. W. Godfrey, and A. Hindle. Software bertillonage determining the provenance of software development artifacts. *Empirical Software Engineering*, 2012.

[3] J. Davies, D. M. German, M. W. Godfrey, and A. J. Hindle. Software bertillonage: Finding the provenance of an entity. In *MSR'11*, 2011.

[4] S. D.J. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & All, 2007.

[5] A. Dresnos. Android : Static analysis using similarity distance. In *45th Hawaii International Conference on System Sciences*, pages 5394–5403, 2012.

[6] Google. Building and running, available at `http://developer.android.com/tools/building/index.html`.

[7] Google. Security and design, available at `http://developer.android.com/google/play/billing/billing_best_practices.html`.

[8] R. Grissom and J. Kim. *Effect sizes for research: Univariate and multivariate applications*. Taylor & Francis, New York, NY, 2012.

[9] M. Harman, Y. Jia, and Y. Zhang. App store mining and analysis: MSR for app stores. In *MSR'12*, pages 108–112, 2012.

[10] H. Huang, S. Zhu, P. Liu, and D. Wu. A framework for evaluating a framework for evaluating mobile app repackaging detection algorithms. *Lecture Notes in Computer Science*, 7904:169–186, 2013.

[11] S. Kawaguchi, P. Garg, M. Matsushita, and K. Inoue. MUDABlue: An automatic categorization system for open source repositories. *IJSS*, 79(7):939–953, 2006.

[12] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk. API change and fault proneness: A threat to the success of Android Apps. In *ESEC/FSE'13*, pages 477–487, 2013.

[13] M. Linares-Vásquez, C. McMillan, and D. Poshyvanyk. On using machine learning to automatically classify software applications into domain categories. *EMSE*, 2012.

[14] T. McDonnell, B. Ray, and M. Kim. An empirical study of API stability and adoption in the Android ecosystem. In *ICSM'13*, 2013.

[15] C. McMillan, M. Grechanik, and D. Poshyvanyk. Detecting similar software applications. In *ICSE'12*, pages 364–374, 2012.

[16] C. McMillan, M. Linares-Vásquez, D. Poshyvanyk, and M. Grechanik. Categorizing software applications for maintenance. In *ICSM'11*, pages 343–352, 2011.

[17] R. Minelli and M. Lanza. Software analytics for mobile applications: Insights and lessons learned. In *CSMR*, 2013.

[18] I. Mojica Ruiz, B. Adams, M. Nagappan, S. Dienst, T. Berger, and A. Hassan. A large scale empirical study on software reuse in mobile apps. *IEEE Software Special Issue on Next Generation Mobile Computing*, 2013.

[19] I. Mojica Ruiz, M. Nagappan, B. Adams, and A. Hassan. Understanding reuse in the Android market. In *ICPC'12*, pages 113–122, 2012.

[20] M. Nagappan, T. Zimmermann, and C. Bird. Diversity in software engineering research. In *ESEC/FSE'13*. ACM, August 2013.

[21] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, and P. Bringas. On the automatic categorization of Android applications. In *CCNC*, pages 149–153, 2012.

[22] S. Schulze and D. Meyer. On the robustness of clone detection to code obfuscation. In *IWSC*, pages 62–68, 2013.

[23] A. Shabtai, Y. Fledel, and Y. Elovici. Automated static code analysis for classifying Android applications using machine learning. In *CIS*, pages 329–333, 2010.

[24] D. Syer, B. Adams, Y. Zou, and A. Hassan. Exploring the development of micro-apps: A case study on the Blackberry and Android platforms. In *SCAM'11*, pages 55–64, 2011.

[25] M. Syer, M. Nagappan, B. Adams, and A. Hassan. Revisiting prior empirical findings for mobile apps: An empirical case study on the 15 most popular open-source Android apps. In *CASCON 2013*, 2013.