Understanding Reuse in the Android Market

Israel J. Mojica Ruiz*, Meiyappan Nagappan*, Bram Adams†, Ahmed E. Hassan*
*Software Analysis and Intelligence Lab (SAIL) †Maintenance, Construction and Intelligence Lab (MCIS)
Queen's University, Kingston, Canada École Polytechnique de Montréal, Canada
Email: mojica, mei, ahmed@cs.queensu.ca Email: bram.adams@polymtl.ca

***Abstract*—Mobile apps are software products developed to run on mobile devices, and are typically distributed via app stores. The mobile app market is estimated to be worth billions of dollars, with more than hundred of thousands of apps, and still increasing in number. This explosion of mobile apps is astonishing, given the short time span that they have been around. One possible explanation for this explosion could be the practice of software reuse. Yet, no research has studied such practice in mobile app development. In this paper, we intend to analyze software reuse in the Android mobile app market along two dimensions: (a) reuse by inheritance, and (b) class reuse. Since app stores only distribute the byte code of the mobile apps, and not the source code, we used the concept of Software Bertillonage to track code across mobile apps. A case study on thousands of mobile apps across five different categories in the Android Market shows that almost 23% of the classes inherit from a base class in the Android API, and 27% of the classes inherit from a domain specific base class. Furthermore, on average 61% of all classes in each category of mobile apps occur in two or more apps, and 217 mobile apps are reused completely by another mobile app in the same category.**

I. Introduction

Mobile apps are applications developed to run on mobile devices. These mobile apps are available commonly through app stores maintained by platform developers such as Apple, Google, Microsoft, or Research in Motion. Today, there are hundreds of thousands of mobile apps across various app stores, and this number has been increasing exponentially since Apple opened the first app store in 2008. There is no decrease anticipated in the rate of growth of the mobile apps market [1] [2]. For example, the Android Market started with 2,300 apps in March 2009, and the last update, in January of 2012, indicates that there are currently more than 380,000 apps available [3].

There are various potential reasons for this current explosion in the number of available mobile apps. One of them could be the large increase in the number of developers for these platforms [4], as well as the availability of decentralized mobile apps stores [5], or the ease of building new apps [6], which attracts many new developers to develop an app. A more fundamental reason why so many mobile apps are developed, might be the use of proven software engineering practices, such as code reuse [7] [8]. Such practices have been analyzed in depth on desktop and server software systems [7] [8] [9] [10], and these principles form the core of any university degree on software engineering. Given the low threshold for entering the mobile app market, it is not clear how such practices are being followed for mobile app development.

This paper analyzes the adoption of code reuse for mobile app development. Frakes and Kang define software reuse as "the use of existing software or software knowledge to construct new software" [9]. Research has shown that the judicious usage of code reuse tends to build more reliable systems and reduce the budget of the total development [7] [8] [9]. Various types of software reuse exists, like inheritance, code, and framework reuse [10], each having its own advantages and disadvantages.

In this paper we focus on exploring the extent of reuse among mobile apps. In order to comprehend the software reuse in the Android Market, we make use of the Software Bertillonage technique introduced by Davies *et al.* [11], because the app stores only distribute the bytecode of the mobile apps, and not the source code. Software Bertillonage generates a class signature from each class's bytecode, then compares these signatures across different apps.

We focused our study on the apps available in the Android Market [12] since mobile devices running the Android OS together have approximately 48% of the market of smartphone devices in US [13], and approximately 68% in China [14] to date. In order to explore the adoption of reuse by mobile app developers, we conduct a case study on five different categories of apps in the Android Market (Cards & Casino, Personalization, Photography, Social and Weather). In particular, we analyze the mobile apps for reuse along the following two research questions:

- **RQ1: What fraction of the classes reuse code by inheritance?**
  We found that almost 27% of the classes in the thousands of mobile apps in our case study inherit from a domain specific base class. 23% of the classes inherit from Android-specific base classes. Furthermore, nine of the top ten base classes are from the Android API.
- **RQ2: How often do mobile apps reuse classes from other apps?**
  We found that on average 61% of the total number of classes in each category occurs in two or more mobile apps. The classes most often reused are the Ads classes provided by Google. Most of the classes reused are from third party developers like Apache and Google. Furthermore, 217 mobile apps have the exact same set of classes as another mobile app in the same category.

This paper is organized as follows: Section II describes the study design. Section III presents the results of our case

study and discusses our findings. Section IV presents a detailed discussion on a special case of reuse. Section V outlines the threats to validity, and Section VI presents the related work. Finally, Section VII concludes the paper.

II. Study Design

In this section we present the subject systems in our study as well as the approach we followed to extract the required data for our study (Fig. 1).

A. Subject Systems

Since the Android operating system has the highest market share among other competing mobile operating systems [13] [14], we analyzed apps from the Android Market. In the Android Market, it is estimated that two-thirds of all the available mobile apps are free (as in "no cost") [3]. Hence, we limited the apps in our case study to only the free ones. The Android Market categorizes the applications under 34 different categories (including 8 Games-subcategories). Across these 34 categories there are more than 380,000 mobile apps [3]. Since app stores including the Android Market do not provide an interface for mass download of mobile apps, we had to manually download them. Hence, we restricted our study to just five categories.

The five studied categories are: Cards and Casino, Personalization, Photography, Social, and Weather. We selected these five categories because we expected to find a higher proportion of reuse in them in comparison with the rest of categories, given us a best case scenario. For example, the category of Photography focuses only on two kinds of apps in particular: apps to take pictures in devices with camera, and apps to show a collection of pictures. Another example is the category of Weather apps, which focuses on apps of weather forecasting. We felt that the similarity and simplicity of the goals of these apps will very likely lead to a large amount of reuse between them.

B. Data Extraction

Each mobile app in the Android Market is packaged as an Android Package file (APK). In this subsection, we explain the process that we use to extract the Java Archive file (Jar) from the APKs, i.e., the first three steps in Fig. 1.

1. Download APK: We manually downloaded every application in the selected categories from the Android Market, onto an actual Android device. For the five categories under study, we selected every single free application. To the best of our knowledge, there is no information available about how many mobile apps exists for each category in the Android Market. Hence, we continued downloading different mobile apps until no additional apps could be found anymore.

2. Backup APK: We backed up the applications from the Android device to our server with the help of a tool called the Astro File Manager [15]. However, the Astro File Manager was not able to backup mobile apps that were marked as private apps, which are mobile apps that were copy protected [16]. Table I shows the total number of APKs[1] that we were able to backup to our server using the Astro File Manager.

TABLE I
Set of Android Applications Under Analysis

Category	Number of applications under study
Cards and Casino	501
Personalization	1,115
Photography	1,034
Social	1,119
Weather	554

The process of downloading and backing up the apps required approximately 100 hours.

3. Extract Jar from APK: Since an APK is an Android specific format, we had to extract the Java bytecode (a regular Java Jar archive) from it. We used an existing tool, dex2jar [17], to perform this operation. We had to modify the names of some APKs in our case study because they made use of special characters that dex2jar was not able to interpret (e.g., Chinese characters).

C. Class Signature Extraction

We used the Software Bertillonage [11] technique to analyze the mobile apps for software reuse. We chose this technique instead of a code clone detection technique, because the Android app store (like any app store) does not provide access to the source code of the mobile apps, only the bytecode. Furthermore, we did not know beforehand the kinds of reuse applied by mobile apps, i.e., whether developers literally cloned whole classes or rather heavily customize the cloned code. A more lightweight technique like software bertillonage is designed to work with this kind of uncertainty. Below, we briefly explain the Software Bertillonage technique, and how we use it to extract the class signatures for our analysis. This is the fourth step in Fig. 1.

4. Generate class signatures: Davies *et al.* [11] proposed a technique called Software Bertillonage in order to identify the origin (the provenance) of a Jar file in a set of Jar files. Software Bertillonage generates a class signature for each class contained in a Jar file. Then, the class signatures are compared across Jar archives to find similar Jar archives through signature matching.

We developed a tool to obtain the class signatures from the Jar archives of the mobile apps based on a slightly modified version of Software Bertillonage, which we explain below. A class signature typically includes the fully qualified name (name of the class along with the package that it is in) of a Java class. The fully qualified name is chosen to avoid that two different classes, in two different packages, that by chance have the same name, have the same signatures. However, it is important to note that if developers copy and paste code, and alter the namespace of the original class, Software Bertillonage will not be able to build the same signature. Despite these limitations, Software Bertillonage is an intuitive technique for

[1]List of apps used: http://sailhome.cs.queensu.ca/replication/ICPC2012/

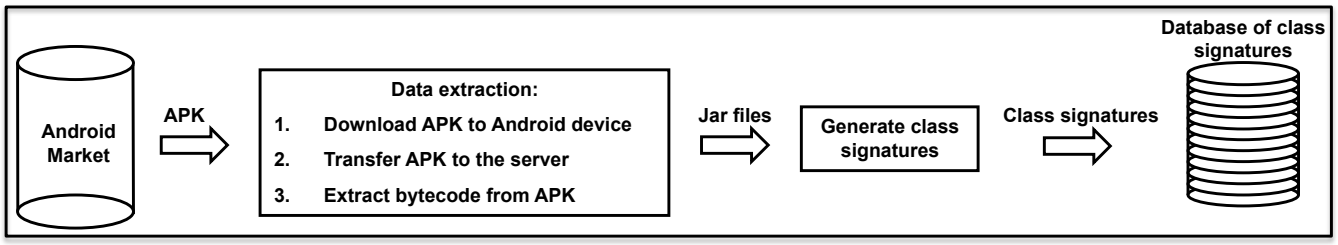

Fig. 1. Steps for the generation of class signatures from mobile apps in the Android Market.

```
package p.s;

public class ClassName extends j.l.E implements x.y.Z{

     public C(){
         //Class' constructor
     }
     syncronized static int a(java.lang.String s) throws
         pack.subPackage.K{
           /*[compiled byte code]*/
     }
}
```

sClass = public class p.s.ClassName extends E implements Z
sM1 = public C()
sM2 = default synchronized static int a(String) throws K
Bertillonage class signature = <sClass,<sM1,sM2>>

Fig. 2. Decompiled Java class, Class and Method lines, and Class Signature

our purposes, since it is lightweight and only requires the bytecode to work.

Class signature example: In Fig. 2 we show a decompiled version of a class (ClassName) and its corresponding class signature on the right. A class signature consists of a *sClass* with the *sM* of each method in the class. The first step of our algorithm produces the *sClass*, which is composed of the package name and class name. The second step produces a sorted set of method signatures *sM1* and *sM2* (one for each method). This set of method signatures form the second component of the class signature. Note that if the method is neither public, nor private, nor protected, default will be added for visibility.

The three steps in the algorithm are:

(a) Extract the package and class line.
(b) Extract and sort the methods in alphabetical order.
(c) Remove methods introduced to implement non-generic interfaces.

In step (b), Davies *et al.* did not sort the methods. However, to avoid that two classes are identical except for the order of their methods are not recognized to be identical, we decided to sort the methods in alphabetical order.

The above algorithm is repeated for each class contained in each Jar file. The result of the execution of this algorithm is a set of class signatures. We refer the reader to the original paper [11] for further details about this algorithm.

We implemented a tool that applies this Software Bertillonage technique with slight modifications, in order to identify the software reuse among a set of mobile apps in the Android Market. The tool was developed using bcel-5.2 [18], the Apache library Davies *et al.* [11] used to analyzed Java binary files.

TABLE II
SET OF CLASS SIGNATURES BY CATEGORY

Category	Total Number of Class Signatures	Mean	Median
Cards and Casino	59,208	118	65
Personalization	52,425	47	11
Photography	88,156	85	26
Social	250,063	223	93
Weather	84,205	152	45

D. Database of Class Signatures

The set of class signatures generated from the jar files of the mobile apps using the above steps was stored in a database for our analysis. During the analysis, we found a list of classes named with a single letter (e.g. *a, b, c*). It was unclear whether these classes were introduced by the developers (as we do not have the source code), or by the compiler. Consequently, we chose to omit classes named with a single letter, since we could not decipher the purpose of these classes from their name. Furthermore, in Android apps developers write an XML file that is compiled into a class called R (Resource) in the APK. This class is present in all Android apps that have a user interface. Hence, we decided to omit this class in our analysis.

The total number of class signatures for each category is shown in Table II. Additionally, the mean and the median of the number of class signatures per app are also shown in Table II for each category. As we can see in Table II, the median number of class signatures per app in Personalization was just 11 while the median number of class signatures in the Social category was almost eight times as much. This shows that the 5 categories each provide a slightly different view about mobile apps.

III. Case Study

The goal of our study is to analyze and understand reuse in the mobile apps of the Android Market. Software bertillonage allows us to study two dimensions of reuse: inheritance level reuse, and class level reuse. This section presents the approach and results of our case study for the two research questions.

A. RQ1: What fraction of the classes reuse code by inheritance?

Motivation: In this research question, we want to analyze the extent of inheritance present in mobile apps of the Android Market [12]. Prior research has shown several advantages (improved conceptual modeling) [19] and disadvantages (tight coupling to base class) [20] of inheritance in OO programming. By identifying which base classes are inherited more often, the developers can allocate appropriate resources to make them more reliable, efficient and modular. Hence, in this research question we want to determine:

(a) What proportion of classes in each category inherit a base class?

(b) What are the top base classes that are inherited?

Approach: In Java, all classes inherit at least from the Object class. Hence, in order to answer question (a), we removed from our set of signatures all those classes that are just inheriting from the Object class (for example public class package.ClassName extends Object). However, we decided to keep any other class, such as Enum or Exception, because the developers explicitly have to code the inheritance from these classes. We used the package name of the base class to identify if it was part of the platform (Android API) or a domain specific package.

In order to answer question (b) we need to determine the base class in each class signature. To determine the base class, we mined the class signature for the base class by searching for the keyword 'extends'. An example of a class line of a class signature is:

```
public class com.google.ads.AdView extends RelativeLayout
```

Therefore the *AdView* class reuses via inheritance the *RelativeLayout* base class that is part of the *android.widget* package. Then, we grouped identical base classes, and calculated the frequency of them. In the results, we will present the top ten base classes in the five categories of Android apps under analysis.

Results: (a) Table III shows the percentage of classes in each category that inherit from the platform base classes and non-platform base classes. Overall, about 50% of classes in each category inherits from a base class. In two of the five categories, the classes in the mobile apps inherit more from the platform base classes, and in two other categories they inherit more from the non-platform domain specific base classes.

(b) On investigating which base classes were inherited the most we found that the "Activity" class in the Android API is the most popular with about 24,266 of the class signatures (across the five categories) inheriting from it. The other base classes in the top 10 most frequently inherited base classes are in Table IV. The second column of the table presents the percentage of classes (total classes that inherit = 266,782) across the five categories that inherit the particular base class. Nine of the ten classes are provided by Google as part of the Android API. The SimpleNode class was from the *org.apache.james.mime4j* package.

TABLE III
Inheritance by Category (percentages calculated with respect to the total number of classes that inherit a base class in the mobile apps of a specific category)

Category	Percentage of base classes from the Android API	Percentage of base classes from other domain specific classes
Cards and Casino	21%	29%
Personalization	29%	26%
Photography	29%	19%
Social	19%	31%
Weather	25%	25%

TABLE IV
Top 10 base classes (percentages calculated with respect to the total number of classes that inherit a base class in the mobile apps across all five category)

Class name	Percentage of classes that inherit from the base class
android.app.Activity	9.1%
java.lang.Enum	3.4%
android.content.BroadcastReceiver	2.6%
android.widget.RelativeLayout	2.3%
java.lang.Exception	1.6%
android.widget.BaseAdapter	1.2%
java.lang.Thread	1.1%
android.os.AsyncTask	0.9%
android.os.LinearLayout	0.8%
org.apache.james.mime4j.field.address.parser.SimpleNode	0.8%

Since it can be expected that mobile apps inherit from platform classes, we also examined the top 10 non-Android API base classes that were inherited. The ranks of the top 10 third party base classes, in the global ranking of base classes, ranged from 10 to 42. The other base classes between ranks 10 and 42 were all from Android. Even among the top 10 third party base classes in Table V, only one (WxFrameworkException by WSI), was from a closed source API. Thus, we were able to identify that most classes in mobile apps inherit, when they do so, from base classes that are available in open source freely available APIs.

Discussion: Prior research looked at the extent of reuse via inheritance in four open source Java projects (JHotDraw, Log4J, ArgoUML, and Azureus) [21]. The results from this research shows that the percentage of classes that inherit from base classes are between 29 and 36% (in ArgoUML and JHotDraw respectively). In comparison, the percentage of reuse via inheritance in each category of mobile apps under study is about 50%.

TABLE V
TOP 10 BASE CLASSES THAT ARE NOT PART OF THE ANDROID API (PERCENTAGES CALCULATED WITH RESPECT TO THE TOTAL NUMBER OF CLASSES THAT INHERIT A BASE CLASS IN THE MOBILE APPS ACROSS ALL FIVE CATEGORY)

Class name	Percentage of classes that inherit from the base class
org.apache.james.mime4j.field.address.parser.SimpleNode	0.77%
com.adwhirl.adapters.AdWhirlAdapter	0.65%
org.apache.http.entity.mime.content.AbstractContentBody	0.44%
org.apache.james.mime4j.field.AbstractField	0.43%
com.wsi.android.framework.wxdata.exceptions.WxFrameworkException	0.39%
org.apache.james.mime4j.field.ParseException	0.38%
org.jivesoftware.smack.packet.IQ	0.36%
gnu.expr.ModuleBody	0.35%
com.qq.taf.jce.JceStruct	0.34%
com.wsi.android.framework.wxdata.exceptions.WxFrameworkException	0.33%

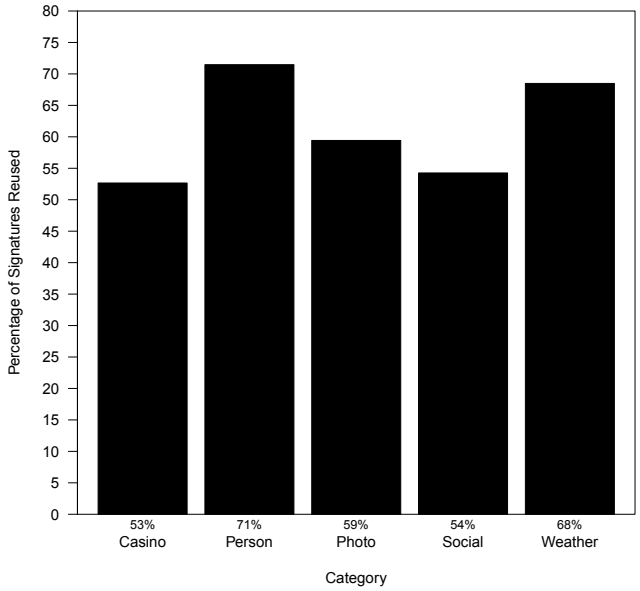

Fig. 3. Proportion of Class Signatures reused per category

B. RQ2: How often does a class from a mobile app gets reused in another app?

Motivation: This question tries to gain a better understanding into types of classes that are being reused across mobile apps. By identifying classes that are reused often (similar to operational profiling), the developers of such classes will be able to allocate appropriate resources to make these classes more reliable and efficient. Hence, in this question we want to determine:

(a) What is the proportion of classes in each category that are reused at least once, i.e., occur in two or more apps?
(b) What is the proportion of classes in each app that are reused in at least one of the remaining apps in the same category?

Approach: In this research question, we used the technique of signature matching proposed by Davies *et al.* [11] to identify reuse of class signatures across the mobile apps in each category. In order to answer question (a), for each category, we calculated the proportion of class signatures that occurs in two or more apps. We took this proportion's complement to obtain the Proportion of Class Signatures Reused (PCSR):

$$PCSR = 1 - \frac{Total\ Number\ of\ Unique\ Class\ Signatures}{Total\ Number\ of\ Class\ Signatures}$$

A high value for the PCSR indicates that the reuse of class signatures is high in that category.

Question (b): The PCSR is only a measure of reuse of class signatures among all mobile apps of a particular category. Question (b) wonders what happens with the occurrence of the class signatures at the app level, i.e., what percentage of class signatures in each mobile app occurs in other mobile apps of the same category. In order to answer this question, we used another measure called *"Global Reuse"* denoted as *Global(A)*, for a particular mobile app *A*. *Global(A)* is the proportion of class signatures found in a mobile app *A* that are also found in at least one other app in the same category as *A*. We define *Global Reuse* in a mobile app as:

$$global(A) = \frac{|s(A) \cap s(\bar{A})|}{|s(A)|}$$

Where: A = Current mobile app under consideration.
$\bar{A}$ = The apps other than A, in the same category as A.
s(A) = Set of class signatures in A.
s($\bar{A}$) = Set of class signatures in $\bar{A}$.

When *Global(A)* = 1, it means that every class signature in *A* occurs in at least one other mobile app in the same category (not necessarily all in the same mobile app). When *Global(A)* = 0, it means that no class signature in the mobile app *A* occurs in any other mobile app in the same category. Therefore, using this measure we are able to identify the commonly used class signatures in each mobile app and the extent to which they are reused.

Results: (a) Fig. 3 shows the Proportion of Class Signatures Reused for each category. Our results indicate that on average 61% of class signatures of a category are reused signatures. This high percentage of reuse indicates that very few classes are unique to a mobile app. We also observe that the Personalization category has the highest proportion of class signatures that was reused, followed by Weather and Photography. Social, and Cards & Casino have the lowest proportion of class signatures that was reused.

(b) In order to analyze the distribution of class signatures among individual mobile apps in each category we calculated the *Global Reuse* of each app in all the five categories. We plotted the cumulative values of *Global Reuse* per app in Fig. 4. The *x-axis* in Fig. 4 is the percentage of classes in each app that are reused in another mobile app in the same category.

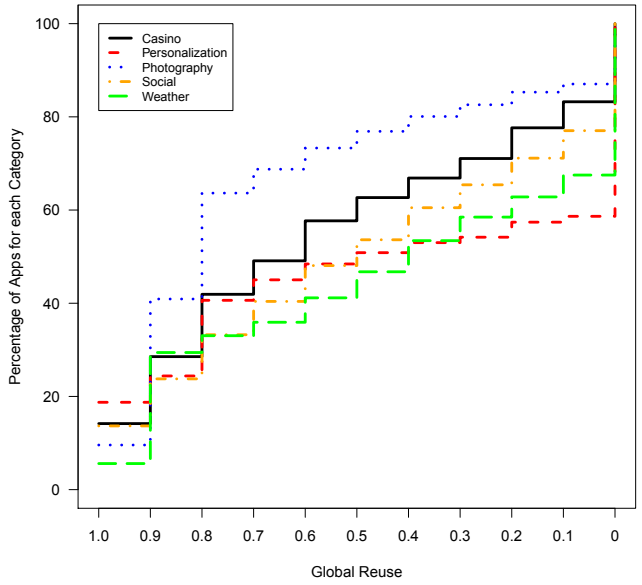

Fig. 4. Cumulative *Global Reuse* per app among the different apps in each category.

The *y-axis* denotes the percentage of apps (cumulative) in each category that has a particular value of *Global Reuse*. Each category has its own step function in Fig. 4. We grouped all the apps in ten intervals of *Global Reuse* with each interval having a width of 10%. That is, all the mobile apps in the category with *Global Reuse* in the interval [1, 0.9) are grouped together in one group. All the mobile apps with *Global Reuse* in the interval [0.9, 0.8) are grouped together in another group, and so on.

Fig. 4 shows that Personalization has the highest percentage of apps with *Global Reuse* = 1, that is every class signature in these mobile apps occurs at least in one other mobile app in the Personalization category. The Weather category has the lowest number of mobile apps with *Global Reuse* =1. However, at the same time Personalization has the highest percentage of apps with *Global Reuse* = 0, while Photography is the category with lowest percentage of apps with *Global Reuse* = 0.

Discussion: Prior research on identifying reuse in open source projects, found that 49% of the analyzed projects (38,700), had at least 80% of their files in common with other projects [22]. From Fig. 4, we can also observe that anywhere between 20 to 40 % of the mobile apps in each category have 80% or more class signatures that occurs in another mobile app of the same category. However, at 30% code reuse, prior research shows that only 53% of the files were in common. In the case of mobile apps, between 50 and 80% of them have at least 30% of classes in common.

To better understand the reasons for reuse, we took a closer look at the class signatures that were found in two or more apps. First we extracted the name of their Project/Organization from the fully qualified name of the class in the class signature. The top ten projects/organizations are presented in Table VI.

TABLE VI
TOP 10 PROJECTS/ORGANIZATIONS

Project / organization name	Number of unique classes in the project/organization	Total number of reused classes from the project/organization
Apache	4,370	86,489
Google	4,821	39,973
Wsi	311	21,205
Anddev	1,162	20,716
twitter4j	1,363	10,781
Gnu	700	10,220
Android	1,117	7,580
Admob	426	7,023
Jivesoftware	716	6,355
Millenialmedia	388	6,140

We noticed that some of the classes have behind a community of developers providing many open source libraries (for example: Apache, Anddev). Only one of them (Wsi) was a company that did not provide its code as open source. From column three we can see that the libraries from the Apache community, are reused the most. We can also see that these ten developers (in Table VI) account for 216,482 of the class signatures (almost 50% of the total class signatures across the five categories under study). However, from column two we can identify that most of the unique classes come from the libraries in Google Code.

The high number of Google packages reused are part of Google AdMob Ads API (com.google.ads). The purpose of this package is to add a banner on top of the apps [23]. The second popular package by Google is part of Google Analytics API (*com.google.android.apps.analytics*). The objective of this package is to provide an API for collecting information about the use of mobile apps. [24].

In the case of Apache, its most popular package in reuse in the Android apps is its package to support the MIME protocol (*org.apache.http.entity.mime*) [25]. The second most popular package from Apache is the Apache James Mime4J (*org.apache.james.mime4j*). The main purpose of this package is to proportion a parser for e-mail message streams, in two different formats: plain rfc822 and MIME format [26].

WSI is a company that focus on the commercialization of weather forecast [27]. The WSI packages most reused are *com.wsi.android.framework.wxdata.tiles.TileDownload* and *com.wsi.android.framework.wxdata.geodata.controller.GeoDataCollectionUpdater*.

Next, we wanted to identify the top five classes that were reused. Table VII shows that the AdSize class from Google is the most popular class. It was used in 1,525 apps across the five categories of apps under study. All five of the top five class signatures are from the ads package. The reason for this is the business model used by mobile app vendors. Typically, the free version of an app contains less functionality or shows advertisements. To enable the latter, apps make use of existing ad libraries. Some of the other non-Google classes that were reused often were the ads class from the AirPush API [28], and classes from the Facebook API.

TABLE VII
TOP FIVE CLASSES ACROSS THE 5 CATEGORIES OF MOBILE APPS

Class name	Number of apps used in
com.google.ads.AdSize	1,525
com.google.ads.AdRequest	1,280
com.google.ads.InstallReceiver	1,280
com.google.ads.InterstitialAd	8,29
com.google.ads.util.AdUtil	7,27

TABLE VIII
TOP 6 PURPOSES OF THE CLASSES

Tag/Purpose	Number of classes
Util	33
UI	25
Image	24
Network	24
Ads	22
I/O	22

Finally, we wanted to determine the purpose of the most reused classes whose signatures were reused so often. Hence, we decided to qualitatively analyze the classes based on their class signatures. Since we could not manually tag all 190K classes, we picked a statistically valid random sample with a confidence level of 95% and a confidence interval of 5%. For this, we had to randomly choose 383 signatures. We then manually tagged each class with the purpose of the class. This tagging was based on the name of the class, the package it was in, the classes that it extended, and the interfaces that it implemented. Then we aggregated these tags according to their purpose.

As can be seen in Table VIII, of the sample of 383 classes, 33 of them (10+-5%) were Utility classes. The other tags in the top 5 positions are classes for user interface, image handling, network, advertisements, and input/output.

An example of a highly reused Util class is *org.anddev.andengine.util*. The main goal of this package is to provide libraries for developing videogames in Android [29]. One of the most reused classes of UI is *com.wsi.android.framework.ui.overlay.item.GeoObjectOverlay Item* by WSI [27]. For the purpose of Image we can mention the class *org.anddev.andengine.opengl.texture.atlas.bitmap.source.Inter nalStorageFileBitmapTextureAtlasSource* by AndEngine [29]. In Network, the most reused class is *org.apache.http.conn.params.ConnRouteParams*. This class is part of the HttpComponents package that provides support for the HTTP protocol [30]. In the Ads category, we identified *com.admarvel.android.ads.AdMarvelAnalyticsEvents* developed by AdMarvel [31]. Finally, for I/O, Apache proportions libraries as part of its project Commons IO. One of these classes identified is *org.apache.commons.io.output.ByteArrayOutputStream*, this is an alternative implementation of the standard java.io.ByteArrayOutputStream class [32].

IV. DISCUSSION

In the second research question (Fig 4) we identified mobile apps that have *Global Reuse* = 1, i.e., every class signature in the mobile app is present in another mobile app in the same category. This is a special case of reuse, since the mobile apps that we studied are end user mobile apps and not just libraries that are intended for reuse. To better understand this special case of reusing classes, we performed additional analysis of this subset of mobile apps. Hence, in this analysis we want to determine:

(a) How many pairs of mobile apps have identical set of classes?
(b) Who are the developers of such mobile apps?

Motivation: In cases where two or more mobile apps have an identical set of signatures, it is possible that a common framework of classes is being used, especially since the final products are different in terms of their look and feel or purpose. Cardino *et al.* suggest the various advantages and disadvantages of framework reuse [33]. They show that reusing a set of classes can increase productivity. However, the cost of building and adapting a framework can be very expensive. Since there is no current research that discusses the extent of framework reuse in mobile apps, we intend to study this type of reuse. Also, since the mobile apps that we are considering in this section are identical to each other with respect to the set of signatures, we wonder if the reuse is done by the same developer or different developers.

Approach: In order to answer question (a), we introduce a new measure called *Local Reuse* of a mobile app, denoted as *local(A,B)*, in each category. For a pair of mobile apps A and B, *Local Reuse* is the proportion of class signatures found in a A, that also occur in B. Unlike *Global Reuse*, we performed a pair wise comparison of mobile apps to determine *Local Reuse*, which is defined as:

$$local(A,B) = \frac{|s(A) \cap s(B)|}{|s(A)|}$$

A high value of *Local Reuse* means that a high number of class signatures are being reused in another app. Since *Local Reuse* is calculated for every pair wise combination of mobile apps, we only considered the largest value of *Local Reuse* for each mobile app.

Note that we only consider pairs of apps that both have *Local Reuse* = 1 to address question (a). This means that both mobile apps have the same number of classes and each class signature in one mobile app is identical to a signature in the other mobile app.

In order to answer question (b), we looked for the developers of those applications with at least one *Local Reuse* = 1 in the Android Market. In order to find the mobile app in the Android Market we had to first look for the application package information contained in the AndroidManifest.xml file in each APK. We extracted the AndroidManifest.xml from the APK for every mobile app with *Local Reuse* = 1, using the apktool [34]. We looked for the package information in the

AndroidManifest.xml file, and we used it to locate the mobile app in the Android Market (thus, the developer).

Results: The results for questions (a) and (b) are presented in Table IX. We were able to find different subsets of apps with *Local Reuse* = 1, i.e, sets of mobile apps that were all identical to each other. We present the total number of subsets in each category in column two of Table IX. We also present the number of mobile apps present in each subset, along with the number of signatures in each of them. On the whole, there were 48 sets of mobile apps that had the same set of class signatures. These 48 sets amounted to 217 (5% of total mobile apps studied) individual mobile apps across the five categories. We notice from Table IX that the Photography category has the largest number of subsets, namely 18. Also the same category has the most number of mobile apps that have a *Local Reuse* = 1, namely 70.

For question (b), we found that in 42 out of the 48 sets, all the mobile apps were developed by the same developer. In the other six subsets, different developers built mobile apps that were identical to each other. Most of these cases were found in the social category.

Discussion: Although, out of the 4,323 mobile apps, only 217 of them were identical to at least one other mobile app, this is a significant number. We found that three of the six sets of apps that were developed by more than one developer (organization), were from companies that provide mobile solutions to other companies. For example, Forum Runner [35] develops and commercializes libraries to manage a forum within a mobile app. They publish apps in the Android Market for different commercial sectors. Other companies in this category use their libraries to generate apps. The remaining three sets of apps were built from sets of open source libraries (from Apache and twitter4j).

Furthermore, even in the 42 sets of mobile apps that were each developed by a single developer, we found that a framework was used. For example in the weather category, there were 11 mobile apps with the same set of class signatures that were all developed by the same developer. When we looked at the actual mobile app in the Android Market, we found that each app basically was the same weather app, but for different metropolitan cities in Europe. Since we know that the developer used the same set of class signatures and that the apps look the same, we can infer that the source code was changed minimally. We think that the change could be the weather service that is called to get the data for the corresponding cities.

Hence, in mobile apps that are identical to another mobile app, it seems like one of the following three types of frameworks is used: (a) private closed source owned by companies for their own purposes, or (b) private closed source owned by companies to develop solutions for their clients, or (c) public open source collection of libraries.

V. Threats to Validity

This section discusses the main threats to validity that can affect the study we performed.

A. Internal Validity

The total number of mobile apps under analysis represents a confidence level of more than 99% with a confidence interval of 2%, based on the last report of the total number of apps available in the Android Market. This limited subset of mobile apps was used instead of the whole set, since each app had to be manually downloaded. To back up of mobile apps, we used the widely popular tool Astro File Manager. We could not get applications marked as private apps by the Astro File Manager.

Our tool to generate class signatures made use of bcel-5.2, the same library used by Davies *et al.* [11]. Also, we made use of popular open source tools like dex2jar and apktool. We had to discard class files named with just one letter (e.g. a, b, c) because we could not decipher the purpose of these classes from their name. Also, we excluded mobile applications with *Global Reuse* = 1 that had in total one class signature.

B. External Validity

Our study analyzes free (as in "no cost") mobile apps in five different categories of the Android Market. Some developers may have uploaded their applications in the wrong category (i.e., an app that displays a snow city theme in the category of Weather).

To find out if our results apply to other app stores and mobile platforms, we need to perform additional studies on those environments. Also, in order to generalize our results, we require an analysis of a more extended number of apps across all categories available in the store.

C. Construct Validity

Our results are based on a modified version of the Software Bertillonage algorithm, and the Global and *Local Reuse* metrics. The modifications were made to identify the cases where the developer modifies the order in which the methods are declared in the class that is being reused. Also, *Local Reuse* is the same metric used in [11] (Inclusion Index), while *Global Reuse* is a complementary metric of *Local Reuse*.

VI. Related Work

In this section, we discuss related work about Software Bertillonage, OO reuse, Clone detection and research done on APK files.

A. Software Bertillonage

Davies *et al.* [11] created a technique named Software Bertillonage to find the origin (provenance) of a set of 51 Jar files compared to the public repository of 130,000 Jar files provided by Maven2. We used the same technique to look for patterns that pointed to software reuse among a set of Android apps.

TABLE IX
NUMBER OF SETS OF MOBILE APPS WITH IDENTICAL CLASS SIGNATURES AND THEIR DEVELOPER INFORMATION

Category	# subsets in which mobile apps are identical	# mobile apps across all subsets	# sets in which all apps have same developer	# sets in which all apps do not have same developer
Cards and Casino	4	13	3	1
Personalization	14	56	13	1
Photography	18	70	17	1
Social	8	57	5	3
Weather	4	21	4	0

B. OO reuse

Conte *et al.* [37] analyzed the workload of java applications. They collected Java applications on the Internet between 1997 and 1998. They analyzed the workload of these applications based on code reuse at the program, class, and method level. In contrast, we analyzed the inheritance and class level reuse of Android applications.

Mockus pointed out the lack of any work that quantified code reuse in open source software outside the area of code clones. He quantified and identified code reuse in a large set of large open source projects, such as different Linux distributions [22]. In our study, however, we analyzed closed source, mobile applications of the Android Market.

Michail built a tool named CodeWeb to analyze the use of library reuse on source code through data mining [38] [39]. Besides library reuse, we looked for inheritance reuse. Our goal was to analyze and measure OO reuse on a large and popular mobile app store.

Denier *et al.* presented a model through which they analyzed the base classes, and classified these classes depending on the number of methods [21]. While they concentrated their efforts in presenting a model for the understanding of the inheritance within a large software product, we have analyzed and quantified the reuse by inheritance among a large set of mobile apps.

Heinemann *et al.* analyzed 20 different projects in order to discover the extent of reuse among open source Java projects. They classified the type of reuse in two categories: white-box reuse, the type of reuse that includes the source code; and black-box reuse, the one that reuse software in binary form [40]. In contrast, we studied a larger number of clouse-source (but free) mobile apps, in order to discover the reuse along two dimensions: inheritance and class reuse.

C. Clone Detection

Roy *et al.* presented a study of identifying function clones in open source software through the use of the NICAD tool [41]. Kapser *et al.* classified code clones in three groups (forking, templating, and customization) [42]. Hemel *et al.* used binary clone detection in order to find software license violations [43]. Similarly, our focus in this paper is to identify software reuse, instead of identifying new clone detection techniques. We used Software Bertillonage instead of clone detection techniques since the app stores do not give access to the source code. Decompiling the bytecode for the full source code is not as accurate as decompiling to only get the class signatures. Hence, we chose the more accurate and lightweight Software Bertillonage technique.

D. Research on Android Packages (APK) files

To the best of our knowledge, Shabtai *et al.* [44] have been the first researchers to conduct a formal study on APK files. They applied Machine Learning techniques to features extracted from Android applications in order to classify two types of Android apps: tools and games. While they sought to identify common features in Android applications, the present study measures and analyzes the different levels of reuse in the Android Market.

Syer *et al.* compared the source code of mobile apps available for the Android and BlackBerry platforms [45]. They conducted their research on three open source projects available in both platforms. They focus their research on the comparison of size and churn of the code base and the dependency of the code on the platform. In our research, we have conducted a study in a larger set of closed source Android mobile apps, in order to analyze reuse by inheritance and reuse of classes.

VII. CONCLUSION

This paper analyzed Android applications to identify the amount of code reuse in them. We employed several techniques based on the concept of Software Bertillonage to find patterns that indicate how frequently software is reused among all the apps, either via inheritance or via class reuse.

In summary, we found that almost 50% of the classes in the mobile apps of the five categories under study inherit from a base class. We found that most of the base classes that are reused were from the Android API or a third party API.

Additionally, we identified that on average 61% of the classes in each category under study appears in two or more apps of that category. We also found that 20-40% of the applications had 80% or more of their class signatures in at least one other mobile app in the same category. We found that the top classes are those that handle notifications of advertisements and that more than half of the classes that were found in two or more apps were from just ten projects/organizations. In our qualitative analysis, we identified that the top purpose of these classes were Util classes.

In conclusion, our study provides an initial insight into the development practices of mobile apps. We conclude that software reuse is prevalent (compared to regular open source software) in mobile apps of the Android Market. Developers

reuse software through inheritance, libraries and frameworks. Most of these are from publicly available open source artifacts. Developers of these highly reused artifacts can use our results to make them more reliable and efficient. The results also help the developers of mobile apps, to understand the risk they run when the library/framework they build on, changes.

REFERENCES

[1] "Gartner Says Worldwide Mobile Application Store Revenue Forecast to Surpass $15 Billion in 2011," Gartner Inc. [Online]. Available: gartner.com/it/page.jsp?id=1529214

[2] Quirolgico, S.; Voas, J.; Kuhn, R.;, "Vetting Mobile Apps," IT Professional , vol.13, no.4, pp.9-11, July-Aug. 2011.

[3] Van Agten, T.;, "Google Android Market Tops 400,000 Applications | Distimo," [Online]. Available: http://www.distimo.com/blog/2012_01_google-android-market-tops-400000-applications/, Retrieved February 2, 2012.

[4] Stackpole, B.;, "Your next job: Mobile app developer? - Computerworld," [Online]. Available: http://www.computerworld.com/s/article/9217885/Your_next_job_Mobile_app_developer_?taxonomyId=11&pageNumber=1, Retrieved February 10, 2012.

[5] Rowinski, D.;, "[Infographic] History of Mobile App Stores," [Online]. Available: http://www.readwriteweb.com/archives/infographic_history_of_mobile_app_stores.php, Retrieved April 16, 2012.

[6] Lohr, S.; "Google's Do-It-Yourself App Creation Software," [Online]. Available: http://nytimes.com/2010/07/12/technology/12google.html, Retrieved February 10, 2012.

[7] Basili, V.; Rombach, H.D.;, "Support for Comprehensive Reuse," Software Engineering Journal, IEE British Computer Society, vol. 6(5): 303-316, September 1991

[8] Basili, V.; Briand, L.; Melo, W.;, "How Reuse Influences Productivity in Object-Oriented Systems," Communications of the ACM, vol. 39(10): 104-116, October 1996.

[9] Frakes, W.B.; Kang, K.; , "Software reuse research: status and future," Software Engineering, IEEE Transactions on , vol.31, no.7, pp. 529- 536, July 2005.

[10] Ambler, S. W. (1998, Jun 01). The various types of object-oriented reuse. Computing Canada, 24(21), 24-24.

[11] Davies, J.; German, D.M.; Godfrey, M.W.; Hindle, A.; , "Software bertillonage: finding the provenance of an entity," In Proceedings of the 8th Working Conference on Mining Software Repositories (MSR '11).

[12] Apps on Android Market, https://market.android.com/, Retrieved December 22, 2011.

[13] comScore Reports January 2012 U.S. Mobile Subscriber Market Share - comScore, Inc, http://www.comscore.com/Press_Events/Press_Releases/2012/3/comScore_Reports_January_2012_U.S._Mobile_Subscriber_Market_Share, Retrieved April 11, 2012.

[14] Lee, M.;, "Android beats Apple in China mobile platform race: report | Reuters," [Online]. Available: http://www.reuters.com/article/2012/04/10/net-us-android-apple-idUSBRE8390AK20120410, Retrieved April 11, 2012.

[15] ASTRO File Manager. http://www.metago.net/astro-file-manager.php, Retrieved December 22, 2011.

[16] Private Apps, http://sourceforge.net/userapps/mediawiki/infidel/index.php?title=Android, Retrieved December 22, 2011.

[17] Dex2jar, http://code.google.com/p/dex2jar/, Retrieved December 22, 2011.

[18] Apache Commons BCEL, http://commons.apache.org/bcel/, Retrieved December 22, 2011.

[19] Taivalsaari. A;, "On the notion of inheritance," ACM Comput. Surv. 28, 3 (September 1996), 438-479.

[20] Holub, A.;, "Why extends is evil - JavaWorld.com," [Online]. Available: http://www.javaworld.com/javaworld/jw-08-2003/jw-0801-toolbox.html, Retrieved February 17, 2012.

[21] Denier, S.; Gueheneuc, Y.-G.; , "Mendel: A Model, Metrics, and Rules to Understand Class Hierarchies," Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on , vol., no., pp.143-152, 10-13 June 2008.

[22] Mockus, A.; , "Large-Scale Code Reuse in Open Source Software," Emerging Trends in FLOSS Research and Development, 2007. FLOSS '07. First International Workshop on , vol., no., pp.7, 20-26 May 2007.

[23] Google AdMob Ads Android Fundamentals - Google AdMob Ads SDK - Google Developers, https://developers.google.com/mobile-ads-sdk/docs/android/fundamentals, Retrieved April 17, 2012.

[24] Google Analytics SDK for Android - Google Analytics - Google Code, http://code.google.com/apis/analytics/docs/mobile/android.html, Retrieved April 17, 2012.

[25] org.apache.http.entity.mime (HttpMime 4.1.3 API), http://hc.apache.org/httpcomponents-client-ga/httpmime/apidocs/org/apache/http/entity/mime/package-summary.html, Retrieved April 17, 2012.

[26] Apache James Mime4J - Mime4J, http://james.apache.org/mime4j/, Retrieved April 17, 2012.

[27] WSI Media Products - WeatherActive Mobile, http://www.wsi.com/products-media-mobile-weather-active.htm, Retrieved April 17, 2012.

[28] Android Ad Network | Push Notification ad network | Mobile Ad network | Android app monetization, http://www.airpush.com/, Retrieved January 04, 2012.

[29] AndEngine - Free Android 2D OpenGL Game Engine, http://www.andengine.org/, Retrieved April 17, 2012.

[30] Apache HttpComponents, http://hc.apache.org/, Retrieved April 17, 2012.

[31] AdMarvel - Home, http://www.admarvel.com/index.php, Retrieved April 17, 2012.

[32] Commons IO Overview, http://commons.apache.org/io/, Retrieved April 17, 2012.

[33] Cardino, G.; Baruchelli, F.; Valerio, A.;, "The evaluation of framework reusability". SIGAPP Appl. Comput. Rev. 5, 2 (September 1997), 21-27.

[34] Android-apktool - A tool for reengineering Android apk files - Google Project Hosting, http://code.google.com/p/android-apktool/, Retrieved January 04, 2012.

[35] Forum Runner - vBulletin / XenForo / myBB / IP.Board / phpBB Forum iPhone App, http://www.forumrunner.com/, Retrieved April 16, 2012.

[36] Building and Running | Android Developers, http://developer.android.com/guide/developing/building/index.html, Retrieved April 16, 2012.

[37] Conte, M.T.; Trick, A.R.; Gyllenhaal, J.C.; Hwu, W.W.;, "A study of code reuse and sharing characteristics of Java applications," Workload Characterization: Methodology and Case Studies, 1998 , vol., no., pp.27-35, 1999.

[38] Michail, A.; , "Data mining library reuse patterns in user-selected applications," Automated Software Engineering, 1999. 14th IEEE International Conference on. , vol., no., pp.24-33, Oct 1999.

[39] Michail, A.; , "CodeWeb: data mining library reuse patterns," Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on , vol., no., pp. 827- 828, 12-19 May 2001.

[40] Heinemann, L.; Deissenboeck, F.; Gleirscher, M.; Hummel, B.; Irlbeck, M.;, "On the extent and nature of software reuse in open source Java projects," In Proceedings of the 12th international conference on Top productivity through software reuse (ICSR'11), Klaus Schmid (Ed.). Springer-Verlag, Berlin, Heidelberg, 207-222.

[41] Roy, C.K.; Cordy, J.R.; , "An Empirical Study of Function Clones in Open Source Software," Reverse Engineering, 2008. WCRE '08. 15th Working Conference on , vol., no., pp.81-90, 15-18 Oct. 2008

[42] Kapser, C.; Godfrey, M.W.; , "Cloning Considered Harmful," Reverse Engineering, 2006. WCRE '06. 13th Working Conference on , vol., no., pp.19-28, Oct. 2006.

[43] Hemel, A.; Trygve Kalleberg, K.; Vermaas, R.; Dolstra E.; "Finding software license violations through binary code clone detection," In Proceedings of the 8th Working Conference on Mining Software Repositories (MSR '11).

[44] Shabtai, A.; Fledel, Y.; Elovici, Y.; , "Automated Static Code Analysis for Classifying Android Applications Using Machine Learning," Computational Intelligence and Security (CIS), 2010 International Conference on , vol., no., pp.329-333, 11-14 Dec. 2010.

[45] Syer, M.D.; Adams, B.; Zou, Y.; Hassan, A.E.;, "Exploring the Development of Micro-Apps: A Case Study on the BlackBerry and Android Platforms," In Proceedings of the 11th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2011).