A Large Scale Empirical Study on Software Reuse in Mobile Apps

Israel J. Mojica[1], Bram Adams[2], Meiyappan Nagappan[1], Steffen Dienst[3], Thorsten Berger[4], Ahmed E. Hassan[1]

[1] *SAIL, Queen's University, Canada*

[2] *MCIS, Polytechnique Montréal, Canada*

[3] *BIS, University of Leipzig, Germany*

[4] *Process and System Models group, IT University of Copenhagen, Denmark*

mojica@cs.queensu.ca, bram.adams@polymtl.ca, mei@cs.queensu.ca, sdienst@informatik.uni-leipzig.de, thbe@itu.dk, ahmed@@cs.queensu.ca

Abstract

Mobile apps are software products developed to run on mobile devices. In less than five years, the number of apps has grown exponentially to more than one million in the largest mobile app stores. One possible explanation for this exponential growth could be the adoption of well-proven software engineering practices, in particular of software reuse, despite the often conjectured lack of training of mobile app developers. We performed a study on hundreds of thousands of Android apps across 30 different categories to analyze software reuse in the Google Play app store. We found the following about three kinds of reuse: (a) 18.75% of Android app classes inherit from a base class in the Android API, and 35.78% of the classes inherit from a domain-specific base class; (b) 84.23% of classes across all categories of mobile apps occur in two or more apps; (c) 17,109 mobile apps were a direct copy of another app. Overall, app developers perform substantial software reuse, which means that they may benefit from increased productivity, yet at the same time become more dependent on the quality of the reused apps and libraries.

Index Terms

software reuse; mobile apps; Android

I. Introduction

Mobile apps are applications developed to run on mobile devices such as smartphones or tablets. App users typically access these apps through different online app stores, such as the Google Play App Store, the Blackberry App World, the Apple App Store and the Windows Phone App Store, with the Google Play store alone having 700,000 apps at the end of 2012 [1].

The huge market (billions of downloaded apps, with a projected revenue of more than 22 billion dollar by 2016), standardized app stores and feature-packed mobile platforms like Android or iOS have attracted thousands of developers [2]. A survey of 352 app developers [3] has shown that 40% of the developers develop apps outside their main job, 21% work on apps part-time, and 39% made their living through app development. Many of these developers are highly educated (e.g., 33% had some graduate school experience), but not necessarily in software engineering. Hence, the survey concluded that the app developers "often lack expertise in all the aspects of app development needed to be successful". Although half of the developers earned less than $15,000, developers still earned on average $45,000 per year.

In order to understand why, despite a lack of formal training, app developers are able to succeed, we previously have studied the usage of proven software engineering practices by mobile app developers in 4,323 free (as in "no cost") Android apps across 5 categories of apps [14]. In particular, we found that the practice of software reuse ("the use of existing software or software knowledge to construct new software" [7], [8]) is high among app developers (compared to "regular" open source projects), with 61% of the classes on average appearing in two or more apps through inheritance, libraries or frameworks. Surprisingly, we also found 48 clusters of 217 apps that were identical, often consisting of fake apps cobbled together by unscrupulous app developers out of existing apps [4], in order to:

- replace the advertisement libraries in order to steal revenue from the original app developers [5];
- add malicious functions to steal information from the app users (e.g., contact list numbers) [4];
- automatically charge app users without their knowledge (e.g., sending text messages to premium numbers) [6].

In order to analyze whether these phenomena are just specific to the studied apps or generalize to mobile apps in general, this paper extends our previous study to more than 200,000 free Android apps across all 30 app categories from the Google Play app store. Using textual signatures to characterize each Java class, we analyze inheritance and code reuse, as well as the special case of framework reuse where whole apps are being reused.

II. Study Design

Software reuse [7], [8], has been analyzed since 1968, when McIlroy proposed to mass produce software with the help of reusable components. Various types of software reuse exist, like inheritance, code, and framework reuse [9], each having its own advantages and disadvantages. This paper studies inheritance and code reuse, as well as the special case of framework reuse where a whole app is being reused by another app. To analyze these different forms of reuse, we generate a signature for each Java class of a mobile app, then track the signature across all apps. This section details this approach, as well as the data set used in our study.

A. Crawling Google Play App Store

In 2011, two of the authors crawled the official Google Play app store [11] to obtain two data sets used in this study: (1) the binary Android apps in the official Android Package (APK) format, and (2) the apps' metadata, i.e., specific information for each app, such as app name, app package name, app category, version number, developer name, and developer email address.

We study only the free (to download) apps because we required access to the source code or bytecode, which is not possible legally for paid apps without actually paying. However, since the Android platform has the largest user base [12], free apps represent 75% of all Google Play App Store apps[1], and free apps are downloaded more often [13], we believe that our case study space is broad and representative enough to draw valid conclusions.

Google Play classifies the apps in 27 different app categories, and 8 subcategories under Games. The crawling did not fetch two categories, i.e., "Live wallpapers" and "Widgets", and we found two identically named subcategories under Games. This left us with 208,601 apps in 30 categories, as shown in the second column of Table I.

B. Class Signature Extraction

We used the Software Bertillonage [10] technique to analyze the mobile apps for software reuse. We chose this technique instead of a more traditional code clone detection technique, because the Google Play app store (like any app store) does not provide access to the source code of the mobile apps, only to the bytecode. Similarly, hashing-based techniques on bytecode would not work either because developers could change whitespace, comments, the order of methods or even parts of the implementation

[1]http://www.appbrain.com/stats/free-and-paid-android-applications

of methods. Instead, Software Bertillonage only analyzes the interface of classes, i.e., the class name and method prototypes, which can readily be extracted from bytecode.

These are the steps we used to obtain our analysis data:

1) Extracting bytecode from the APKs: We used the dex2jar tool[2] to extract the JAR files (Java Archives) from the APKs.
2) Extracting classes and methods: Using the Apache BCEL library[3], we extracted for each class in the JAR files its fully qualified name (name of the class along with the package that it is in), its base class (if applicable), and the names and parameter types of its methods. This data was stored in a database.
3) Removing obfuscation: Android app source code may be obfuscated using different tools (e.g., ProGuard). Obfuscated classes have names consisting of one or two letters (e.g., "ab"), or three to five identical initial letters (e.g., "aaaa"). We decided to omit these obfuscated classes. Furthermore, we also omitted the class called R (Resource), since this one is compiled automatically from an XML file describing the Graphical User Interface (GUI).
4) Generating class signatures: A Software Bertillonage signature consists of the fully qualified class name, followed by an alphabetical list of the method prototypes. Methods that implement generic interfaces are removed, since such interfaces cannot be processed by the existing tools correctly. Originally, Davies et al. [10] did not sort the methods. However, we decided to change this in order to deal with accidental reordering of methods. The set of generated class signatures was stored in a database for further analysis.
5) Comparing signatures across JAR archives: Textual comparison of the signatures allows to match classes across JAR archives, and hence apps. Note that, since Android apps include the bytecode of the app itself as well as all used libraries into their APK, our analyses automatically consider both source and library code.

The third, fourth and fifth column in Table I show the total, mean and median number of class signatures for each category, after removing obfuscated classes.

III. Case Study

We now use the extracted signatures to analyze inheritance and code reuse, as well as framework reuse of whole apps.

[2]http://code.google.com/p/dex2jar

[3]http://commons.apache.org/bcel/

A. *Inheritance Reuse*

In Java, inheritance is indicated by the `extends` keyword, e.g.: `public class com.google.ads.AdView extends android.widget.RelativeLayout`. Here, the `AdView` class reuses via inheritance the `RelativeLayout` base class that is part of the `android.widget` package. Using the package name of the base class, one can identify if the class is part of the Android API platform[4] or is rather a domain-specific class. Platform base classes also comprise standard Java classes like `java.lang.String` or `java.net.URL`.

The last three columns of Table I show the percentage of classes in each category that inherit from platform (second column) and domain-specific base classes (third column). Since in Java all classes inherit at least from the `java.lang.Object` class, we separated out all those classes that inherit only from the `Object` class (first column).

On average, 54.53% of classes in each category inherit from a base class, while only 18.75% of all classes inherit from the Android platform base classes. In only three out of the 30 categories ("Comics", "Personalization" and "Photography"), the classes in the mobile apps inherit more from the platform base classes than from domain-specific (non-platform) base classes.

The "Activity" class in the Android API is the most popular base class with about 8.40% of the (non-java.lang.Object) class signatures inheriting from it. Table II shows how eight out of the top ten base classes are part of the Android API. The `TwitterResponseImpl` class in the `twitter4j` package is from the twitter4j-core library[5], and the `SerializerBase` class in the `org.codehaus.jackson.map.ser` package is from the jackson-mapper library[6].

Since it is not a surprise that mobile apps inherit from (and hence developers reuse) platform classes like "Activity", we also examined the top 10 domain-specific base classes that were inherited. The ranks of this top 10 of domain-specific base classes, in the global ranking of base classes, ranged from 7 to 26. Among these top classes, we found classes to interface with Twitter, handle JSON data, interpret the Kawa scheme implementation[7] (used by Android App Inventor[8], an application to create apps by dragging and dropping visual components), interpret JavaScript, HTML5 and CSS3 (used by the PhoneGap framework[9] to create

[4]http://developer.android.com/reference/packages.html

[5]http://twitter4j.org/

[6]http://jackson.codehaus.org/

[7]http://www.gnu.org/software/kawa/index.html

[8]http://appinventor.mit.edu/

[9]http://phonegap.com/

Android apps using web programming languages, and by the Rhino project[10] to integrate JavaScript in Android apps), and to display advertisements (provided by the Adwhirl library[11]).

If we compare these findings to prior research on reuse in four open source Java projects [15], we find that about 54.53% of the classes of a mobile app inherits from a base class (other than `java.lang.Object`), compared to between 29 and 36% on open source Java projects. Although the Java case study considered only four systems compared to the many thousands in our analysis, these findings seem to suggest that mobile app developers make more thorough use of inheritance-based reuse.

B. Code Reuse

To understand the degree of code reuse of classes, and which classes are being reused across mobile apps, we measured the proportion of classes in each category that are unique to one app. We took this proportion's complement to obtain the Proportion of Class Signatures Reused (PCSR) of a category :

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

A high value for the PCSR indicates that the reuse of class signatures is high in that category. Figure 1 shows the Proportion of Class Signatures Reused for each category.

Overall, 84.23% of class signatures are reused across all categories. Six out of the 30 categories are above the overall PCSR: Games-Brain & Puzzle 89.84%, Sports 88.65%, Games-Casual 87.78%, Games-Arcade & Action 86.72%, Games-Racing 84.62%, Music & Audio 84.46%. The Comics category is the category with the lowest PCSR with only 62.72%, close to the Games-Cards & Casino with 63.59%.

The high percentage of code reuse across apps indicates that very few classes are unique to a mobile app. In other words, we again find that mobile app developers value reuse.

C. Framework Reuse of Whole Apps

This section focuses on the specific cases of framework reuse where all classes of an app are being reused by another app, or multiple apps have identical sets of classes. Such cases basically correspond to extreme cases of the general definition of framework reuse, where it is said that a set of apps is reusing a common framework of classes if two or more mobile apps have a common set of signatures (for example for persistence of data or look and feel purposes). Cardino et al. showed that framework

[10]https://developer.mozilla.org/en-US/docs/Rhino

[11]http://code.google.com/p/adwhirl/

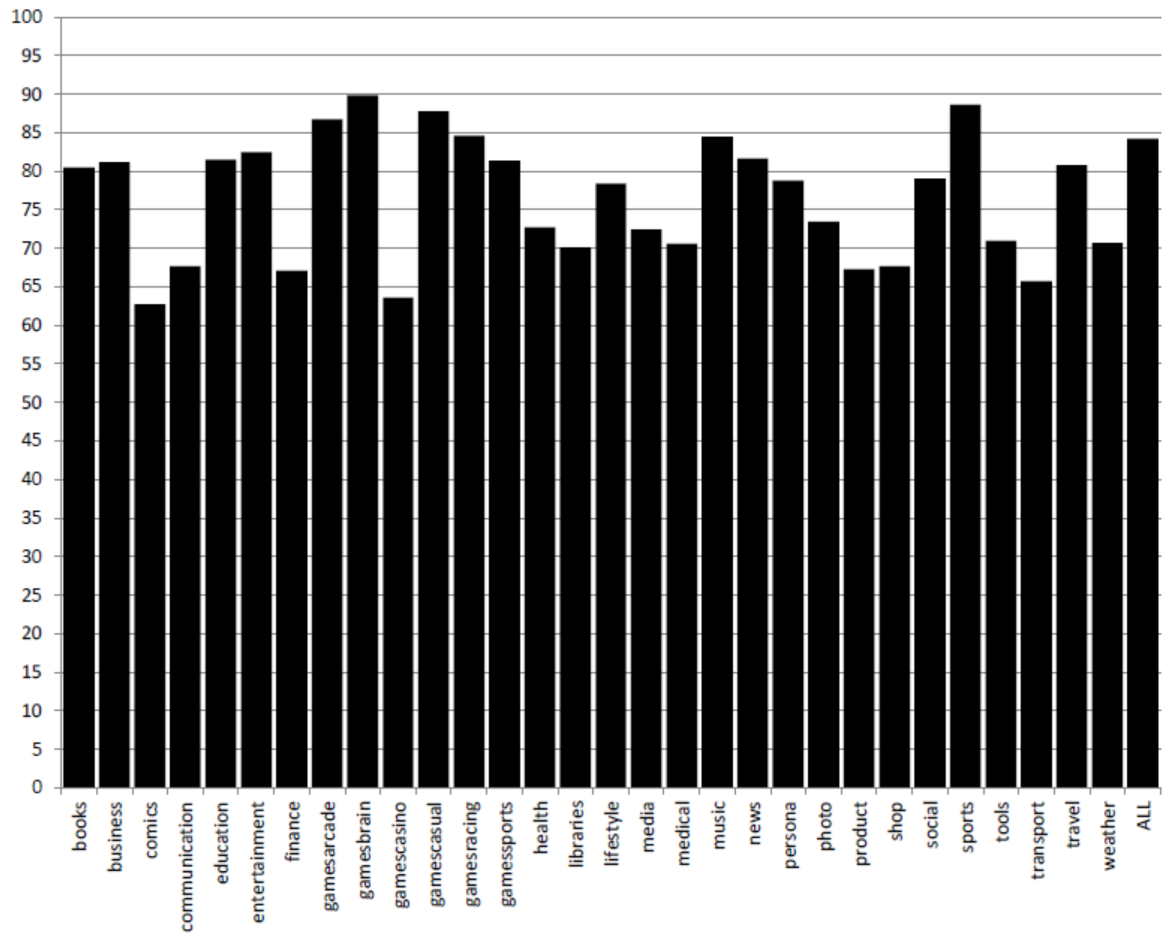

Figure 1: Proportion of class signatures reused per category.

reuse in general can increase productivity, amongst other advantages [16], yet the special case that we consider corresponds to the less recommended practices described in the introduction.

For this analysis, we introduce the Local Reuse of a mobile app, denoted as local(A,B). For a pair of mobile apps A and B, local reuse is the proportion of class signatures found in A, that also occur in B:

$$local(A,B) = \frac{|s(A) \cap s(B)|}{|s(A)|},\ s(X)\ is\ set\ of\ signatures\ in\ app\ X. \tag{2}$$

A high value of local reuse means that a high number of class signatures are being reused in another app. Note that we only consider pairs of apps that both have local reuse = 1, which 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.

There were 1,811 sets of mobile apps that had the same set of class signatures, comprising 17,109 individual mobile apps (8.20% of all studied mobile apps). 30 out of the 1,811 sets comprise apps from different categories. The number of classes (size of app) across the 1,811 different sets of apps with local reuse = 1 varies from one to 1,903 classes. Most clusters contain 1 (81 clusters), 8 (63 clusters) or 25 (33 clusters) reused classes, with a median of 61 reused classes. The top three largest clusters consists of 473 apps with 1 reused class, 339 apps with 8 reused classes, and 334 apps with 20 reused classes. The median number of apps across clusters is 3 apps.

The largest cluster is a set of 473 apps, each of which contains 1 class. Class `isest.nestmi.IsestApp` is reused by 468 apps in the "Games – Arcade & Action" category, four apps in the "Games – Brain & Puzzle" category, and one app in the "Games – Cards & Casino" category. These apps were registered by three different app developers. However, the names of the app developers are similar, i.e., Vital-Game (200 apps), Vital-Game.com (88 apps), and VitalGame (185 apps). Furthermore, the app developer's contact website is the same for the three developers, and states that these apps require Flash to run on Android devices[12]. Hence, the class is basically a wrapper to execute Flash apps.

The cluster with the largest number of classes has 1,903 classes and is a set of 26 apps. All the apps are under the "News & Magazines" category, and are developed by one unique developer (Raycom Media, Inc.), a T.V. broadcasting company. Analyzing the classes used by this set of apps, shows that different open source packages are being reused, such as com.github.droidfu, com.j256.ormlite, com.facebook.android, com.bea.xml.stream, and other packages.

When considering clusters with 100 or more classes and at least 100 apps, we found the following 7 clusters (ordered ascending by the number of classes):

1) 174 apps with 124 classes: These apps are distributed across 20 different categories, especially "News & Magazines" (27 apps), "Business" (24), "Communication" (22), and "Music & Audio" (22). 18 different app developers created these 174 apps, and the contact websites are distinct, which indicates that they are developed by different app developers. 120 apps were created by the AppsBuilder app developer, with package names containing the package com.appsbuilder. AppsBuilder[13] turns out to be a web app to visually build and generate apps without having to program.

[12]http://www.vital-game.com/

[13]http://www.apps-builder.com/

2) 154 apps with 161 classes: All these apps are under the “Books & Reference” category, and were created by a Chinese developer (Lexun). These apps have been removed from the Google Play app store, likely by Google.
3) 178 apps with 170 classes: All these apps are under the “Books & Reference” category, and were created by another Chinese developer (3GQA Dev Team). Again, all the apps from this app developer were removed from the Google Play app store, likely by Google.
4) 106 apps with 178 classes: These apps are distributed across 13 different categories, especially “Sports” (50 apps) and “News & Magazines” (35). These apps were created by 38 different app developers, especially What Ho What Ho. These apps were built using the AppYet[14] web app for building Android apps without programming.
5) 103 apps with 235 classes: These apps are distributed across the “Games-Casual” (99 apps) and “News & Magazines” (4) categories, and were created by three different app developers. These apps are using packages such as jp.co.mediaship.andapp, jp.co.milog, and others. Apps from this cluster were removed from the Google Play app store, likely by Google.
6) 112 apps with 262 classes: All the apps in this cluster are in the “Sports” category. These apps were created by only one app developer (Pliabull Technologies), integrating different packages such as javax.mail, com.sun.mail, com.millennialmedia, com.google.android.apps.analytics, and other packages.
7) 232 apps with 431 classes: These apps are distributed across the “Lifestyle” (169 apps), “Entertainment” (59) and “Education” (4) categories. The creator of these apps has three different identifiers, however only one contact website, which again indicates that these apps were built by the same app developer (Quipper). These apps were removed from the Google Play app store, likely by Google. The apps used different packages such as com.admob, com.facebook, org.codehaus.jackson, and others.

Wrapping up, mobile apps that are identical to other mobile apps seem to belong to one of the following four types of framework reuse: (a) reuse of private closed source classes owned by companies for their own purposes, (b) reuse of private closed source classes owned by companies to develop solutions for their clients, (c) reuse of a public, open source collection of libraries, or (d) use of automatic mobile app builders.

[14]http://www.appyet.com/

IV. Discussion & Conclusion

In order to understand the exponential growth of mobile apps, despite the lack of formal software engineering training of most app developers, we analyzed the application of one of the most basic software engineering practices, i.e., reuse. We used the Software Bertillonage technique to find patterns that indicate how frequently mobile apps perform inheritance, code and framework reuse on more than 200,000 free Android apps, across 30 categories.

We found that almost 54% of the classes in the mobile apps of the thirty categories under study inherit from a base class, especially Android platform classes and domain-specific API classes. Furthermore, overall 84.23% of the classes are reused across all the categories of apps, which is even higher than our previous findings on a smaller sample of mobile apps [14]. Finally, more than 17,000 apps are almost identical to other apps, either because they are generated automatically, reuse the same open source libraries, or because developers willingly capitalized on a proven app for additional revenue. App stores like Google Play are quick to remove the latter kind of apps.

The fact that software reuse, in the form of inheritance, class, and library reuse, is prevalent in mobile apps of the Google Play app store, means that app developers reap all the typical reuse benefits, such as improved productivity, higher quality software and faster time-to-market [7], [8], even though many did not receive a formal training in software engineering [3]. It is not clear whether this successful reuse is due to the quality of mobile platforms, development tools, app stores or (a combination of) other factors. Possible other factors could be the relatively small size of the mobile app code base and development team, although in recent work we have found that for these characteristics, mobile apps behave identically to small Unix utility applications [18]. In any case, there is evidence that mobile platforms encourage reuse by making frequently reused apps and libraries a part of the mobile platform itself. This is for example what happened to the JSON data format support on the Blackberry platform [17]. User studies with app developers are needed to understand how they reuse code and whether the way in which they approach reuse is different from developers of non-mobile apps.

At the same time, mobile apps also inherit the disadvantages of reuse, such as increased dependencies on the reused classes and a potentially large amount of effort needed to integrate a reused class in the mobile app. For example, an app reusing a low-quality library runs a higher risk that bugs or incompatibilities of the library harm the app's quality and reliability. More research is needed to analyze this negative impact on mobile apps in the long term, as well as to analyze other forms of reuse such as the general case of framework reuse.

References

[1] Rosen, K.; , “Google Play Has 700,000 Apps, Tying Apples App Store,” http://mashable.com/2012/11/01/google-apps-tie-apple/, Retrieved November, 2012.

[2] Columbus, L.; , “Roundup Of Mobile Apps & App Store Forecasts, 2013,” http://www.forbes.com/sites/louiscolumbus/2013/06/09/roundup-of-mobile-apps-app-store-forecasts-2013/, Retrieved June, 2013.

[3] Craven, A.; , “A demographic and business model analysis of today’s app developer,” http://appdevelopersalliance.org/files/pages/GigaOMApplicationDevelopers.pdf, Retrieved September, 2012.

[4] Abendan, O.; , “Fake Apps Affect ANDROID OS Users,” http://about-threats.trendmicro.com/us/webattack/72/Fake%20Apps%20Affect%20Android%20OS%20Users, Retrieved February, 2013.

[5] Zhou, W.; Zhou, Y.; Jiang, X.; Ning, P.; , “Detecting repackaged smartphone applications in third-party android marketplaces,” In Proc. of the 2nd ACM Conf. on Data and Application Security and Privacy (CODASPY ’12), pp.317-326.

[6] Warman, M.; , “Fake Android apps scam costs 28,000,” http://www.telegraph.co.uk/technology/news/9286538/Fake-Android-apps-scam-costs-28000.html, Retrieved May, 2012.

[7] Basili, V.R.; Briand, L.C.; Melo, W.L.; , 1996. “How reuse influences productivity in object-oriented systems,” Commun. ACM 39, no.10, pp.104-116, October 1996.

[8] Frakes, W.B.; Kyo Kang; , “Software reuse research: status and future,” IEEE Transactions on Software Engineering, vol.31, no.7, pp.529–536, July 2005.

[9] Ambler, S.W.; , “The various types of object-oriented reuse,” Computing Canada, 24, 24–24, June 1998.

[10] Davies, J.; German, D.M.; Godfrey, M.W.; Hindle, A.; , “Software bertillonage: finding the provenance of an entity,” In Proc. of the 8th Working Conf. on Mining Software Repositories (MSR ’11), pp.183–192.

[11] Dienst, S.; Berger, T.; , “Static analysis of app dependencies in android bytecode,”. Technical Note, 2012.

[12] Kellogg, D.; , “40 Percent of U.S. Mobile Users own Smartphones; 40 Percent are Android,” http://www.nielsen.com/us/en/newswire/2011/40-percent-of-u-s-mobile-users-own-smartphones-40-percent-are-android.html, Retrieved November, 2012.

[13] Gartner, Inc.; , “Gartner Says Free Apps Will Account for Nearly 90 Percent of Total Mobile App Store Downloads in 2012,” http://www.gartner.com/newsroom/id/2153215, Retrieved November, 2012.

[14] Mojica-Ruiz, I.J.; Nagappan, M.; Adams, B.; Hassan, A.E.; , “Understanding reuse in the Android Market,” In Proc. of the IEEE 20th Intl. Conf. on Program Comprehension (ICPC ’12), pp.113–122.

[15] Denier, S.; Gueheneuc, Y.-G.; , “Mendel: A Model, Metrics, and Rules to Understand Class Hierarchies,” In Proc. of the 16th IEEE Intl. Conf. on Program Comprehension (ICPC ’08), pp.143–152.

[16] Cardino, G.; Baruchelli, F.; Valerio, A.; , “The evaluation of framework reusability,” SIGAPP Appl. Comput. Rev. 5, pp.21–27, September 1997.

[17] Syer, M. D.; Adams, B.; Hassan, A. E.; Zou, Y.; , “Exploring the Development of Micro-Apps: A Case Study on the BlackBerry and Android Platforms,” In Proc. of the Intl. Working Conf. on Source Code Analysis and Manipulation (SCAM ’11), pp.55–64.

[18] Syer, M. D.; Nagappan, M.; Adams, B.; Hassan, A. E.; , “Revisiting Prior Empirical Findings For Mobile Apps: An Empirical Case Study on the 15 Most Popular Open Source Android Apps,” In Proc. of the IBM CASCON Conf. (CASCON ’13), to appear.

Table I: Characteristics and inheritance results of the Android apps under analysis.

Category	Total # distinct apps	Total # classes	Mean # classes	Median # classes	%base classes java.lang.Object	%base classes platform	%base classes domain-specific
Books & Reference	12,769	1,048,042	82.08	24	44.95%	25.96%	29.10%
Business	8,490	1,572,701	185.24	66	46.61%	17.98%	35.41%
Comics	5,255	123,669	23.53	11	41.73%	38.16%	20.11%
Communication	4,630	537,257	116.04	23	46.67%	21.69%	31.65%
Education	8,100	1,090,928	134.68	34	42.93%	18.50%	38.57%
Entertainment	22,674	2,262,595	99.79	27	44.77%	21.89%	33.34%
Finance	3,958	438,699	110.84	31	40.27%	19.20%	40.53%
Games-Arcade & Action	10,366	1,078,008	103.99	34	47.84%	15.91%	36.25%
Games-Brain & Puzzle	10,355	1,262,509	121.92	48	40.36%	15.96%	43.68%
Games-Cards & Casino	2,052	129,719	63.22	25	48.85%	21.75%	29.41%
Games-Casual	7,486	906,228	121.06	35	45.08%	15.37%	39.55%
Games-Racing	1,679	138,806	82.67	72	54.96%	21.07%	23.97%
Games-Sports	2,956	242,722	82.11	34	53.26%	21.00%	25.74%
Health & Fitness	3,899	397,618	101.98	30	42.87%	20.80%	36.33%
Libraries & Demo	2,706	150,123	55.48	9	44.73%	24.90%	30.36%
Lifestyle	11,314	1,480,139	130.82	33	43.74%	18.51%	37.74%
Media & Video	6,361	394,354	62.00	25	42.84%	27.37%	29.79%
Medical	2,173	184,059	84.70	13	39.86%	18.89%	41.26%
Music & Audio	14,232	5,681,171	399.18	232	49.20%	14.09%	36.72%
News & Magazines	6,732	991,153	147.23	56	48.53%	21.59%	29.88%
Personalization	10,566	361,653	34.23	12	43.47%	30.14%	26.39%
Photography	5,738	264,753	46.14	9	48.14%	29.77%	22.09%
Productivity	4,486	509,583	113.59	27	42.86%	19.62%	37.51%
Shopping	3,642	393,314	107.99	34	42.11%	19.49%	38.41%
Social	6,408	1,013,686	158.19	38	45.82%	20.22%	33.95%
Sports	7,772	1,566,915	201.61	42	45.12%	19.22%	35.66%
Tools	12,129	848,311	69.94	15	40.60%	20.72%	38.68%
Transportation	1,966	182,911	93.04	25	41.67%	18.22%	40.11%
Travel & Local	6,629	1,089,421	164.34	51	40.94%	16.93%	42.13%
Weather	1,078	112,550	104.41	22	44.94%	21.85%	33.21%
All-Categories	208,601	26,453,597	126.81	29	45.47%	18.75%	35.78%

Table II: Top 10 base classes (other than java.lang.Object) across all categories of mobile apps with the highest percentage of inheritance.

class name	%classes inheriting from the class
`android.app.Activity`	8.40%
`android.content.BroadcastReceiver`	1.66%
`java.lang.Enum`	1.57%
`java.lang.Exception`	1.40%
`android.widget.RelativeLayout`	1.35%
`android.os.AsyncTask`	1.13%
`twitter4j.TwitterResponseImpl`	1.03%
`java.lang.RuntimeException`	0.84%
`android.app.ListActivity`	0.76%
`org.codehaus.jackson.map.ser.SerializerBase`	0.74%