

A Large-Scale Empirical Study on Software Reuse in Mobile Apps

Israel J. Mojica, McAfee

Bram Adams, École Polytechnique de Montréal

Meiyappan Nagappan, Queen's University

Steffen Dienst, University of Leipzig

Thorsten Berger, University of Waterloo

Ahmed E. Hassan, Queen's University

// A study of hundreds of thousands of Android apps found substantial software reuse, indicating that while these apps benefit from increased productivity, they're also more dependent on the quality of the apps and libraries that they reuse. //

MOBILE APPS ARE applications developed to run on devices such as smartphones or tablets. Users typically access them through online app stores, such as Google Play, BlackBerry World, the Apple App Store, and the Windows Phone marketplace. The number of available products is staggering, with Google Play alone offering 700,000 apps at the end of 2012.¹

The huge market (billions of downloaded apps, with projected revenue of more than US\$22 billion by 2016), combined with standardized app stores and feature-packed mobile platforms such as Android or iOS, has attracted thousands of developers.² A survey of 352 app developers showed that

- 40 percent develop apps outside of their main job,
- 21 percent work on apps part time, and
- 39 percent made their living through app development.³

Many of these developers are highly educated (for instance, 33 percent had some graduate school experience), but their schooling isn't necessarily in software engineering. Thus, the survey concluded that app developers "often lack expertise in all the aspects of app development needed to be successful." Although half of those surveyed earned less than \$15,000 per year, the average income was still \$45,000.

To understand how, despite a lack of formal training, app developers continue to succeed, we had previously studied the use of proven software engineering practices in 4,323 free (as in "no cost") Android apps across five categories.⁴ In particular,





we found that the practice of software reuse (“the use of existing software or software knowledge to construct new software”^{5,6}) is high among app developers (compared to “regular” open source projects), with 61 percent of the classes on average appearing in two or more apps through inheritance, libraries, or frameworks. Surprisingly, we found 48 clusters of 217 apps that were identical, often consisting of fake apps cobbled together by unscrupulous app developers from existing apps (<http://about-threats.trendmicro.com/us/webattack/72/Fake+Apps+Affect+ANDROID+OS+Users>). We also found that some of these developers

- replace advertisement libraries to steal revenue from the original app developers,⁷
- add malicious functions to steal information from the app users (for example, contact list numbers), and
- automatically charge app users without their knowledge (for example, by sending text messages to premium numbers).⁸

To analyze whether these phenomena are just specific to the apps in our previous study or to mobile apps in general, we extended our study to more than 200,000 free Android apps across all 30 app categories from Google Play.

Study Design

Software reuse^{5,6} has been analyzed since 1968, when Douglas McIlroy proposed to mass produce software with the help of reusable components.⁹ Various types of software reuse exist, such as inheritance, code, and framework reuse, each with its own advantages and disadvantages.¹⁰ Here, we study inheritance and code reuse, as well as the special case of framework reuse where one app reuses another entire app. To analyze these different forms of reuse, we generated a signature for each Java class of a mobile app, then tracked the signature across all apps.

Crawling Google Play

In 2011, two of the authors crawled the official Google Play app store to obtain two datasets used in this study:¹¹

the binary Android apps in the official Android Package (APK) format and the apps’ metadata (that is, specific information for each app, such as the app name, app package name, app category, version number, developer name, and developer email address).

We studied only the free (to download) apps because we required access to the source code or bytecode, which isn’t legally possible for paid apps without actu-

We generated a signature for each Java class of a mobile app, then tracked the signature across all apps.

ally paying. However, because the Android platform has the largest user base,¹² free apps represent 75 percent of all Google Play apps (www.appbrain.com/stats/free-and-paid-android-applications), and free apps are downloaded more often (www.gartner.com/newsroom/id/2153215), we believe that our case study space is broad and representative enough to draw valid conclusions.

Google Play considers 27 different app categories and eight game subcategories. The crawling didn’t fetch two categories (Live Wallpapers and Widgets), and we found two identically named subcategories under Games. This left us with 208,601 apps in 30 categories, as the second column of Table 1 shows.

Class Signature Extraction

We used the Software Bertillonage technique, which compares the interface of Java classes, to analyze the mobile apps for software reuse.¹³ We chose this technique instead of more traditional code clone detection because the Google Play app store (like any app store) doesn’t provide access to the source code, only the bytecode. Similarly, hashing-based techniques on bytecode wouldn’t work because developers could change whitespace, comments, the order of methods, or even parts of the implementation of methods. Instead, Bertillonage only analyzes the interface of classes (that is, the class name and method prototypes), which can readily be extracted from bytecode.

We used five steps to obtain our analysis data.

TABLE 1

Characteristics and inheritance results of the Android apps under analysis.

Category	Total no. of distinct apps	Total no. of classes	Mean no of. classes	Median no. of classes	Percent of base classes java.lang. Object	Percent of base classes platform	Percent of base classes domain-specific
Books and 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 and action	10,366	1,078,008	103.99	34	47.84	15.91	36.25
Games—Brain and puzzle	10,355	1,262,509	121.92	48	40.36	15.96	43.68
Games—Cards and 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 and fitness	3,899	397,618	101.98	30	42.87	20.80	36.33
Libraries and 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 and video	6,361	394,354	62.00	25	42.84	27.37	29.79
Medical	2,173	184,059	84.7	13	39.86	18.89	41.26
Music and audio	14,232	5,681,171	399.18	232	49.20	14.09	36.72
News and 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 and 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

1. *Extract bytecode from the APKs.* We used the dex2jar tool (<http://code.google.com/p/dex2jar>) to extract the JAR (Java Archive) files from the APKs.
2. *Extract classes and methods.* For each class in the JAR files, we extracted its fully qualified name (name of the class along with the package that it's in), its base class (if applicable), and the names and parameter types of its methods using the Apache BCEL (Byte Code Engineering Library; <http://commons.apache.org/bcel>). We stored this data in a database.
3. *Remove obfuscation.* Different tools can obfuscate Android app source code (for example, ProGuard). Obfuscated classes have names consisting of one or two letters (for example, "ab") or three to five identical initial letters (for example, "aaaa"), so we decided to omit these particular classes. Furthermore, we also omitted the class called R (Resource) because it's compiled automatically from an XML file describing the GUI.
4. *Generate class signatures.* A 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 because existing tools can't process such interfaces correctly. Originally, Julius Davies and his colleagues didn't sort the methods.¹³ However, we decided to change this to deal with accidental method reordering. We stored the set of generated class signatures in a database for further analysis.
5. *Compare signatures across JAR archives.* Textual comparison of the signatures lets us match classes across JAR archives, and hence, apps. Because Android apps include an app's bytecode along with all of the libraries it uses in the APK, our analyses automatically consider both source and library code.

The third, fourth, and fifth columns in Table 1 show the total, mean, and median number of class signatures for each category, minus obfuscated classes.

Case Study

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

Inheritance Reuse

In Java, inheritance is indicated by the *extends* keyword—for example, *public class com.google.ads.AdView extends android.widget.RelativeLayout*. Here, the

AdView class reuses via inheritance the RelativeLayout base class that's part of the android.widget package. Using the package name of the base class, we can identify if the class is part of the Android API platform (<http://developer.android.com/reference/packages.html>) or is a domain-specific class. Platform base classes also comprise standard Java classes such as java.lang.String or java.net.URL.

The last three columns of Table 1 show the percentage of classes in each category that inherit from the platform and domain-specific base classes. Because all classes in Java inherit at least from the java.lang.Object class, we separated out all those that inherit only from the Object class.

On average, 54.53 percent of the classes in each category inherit from a base class, while only 18.75 percent 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 (nonplatform) base classes.

The Activity class in the Android API is the most popular, with approximately 8.4 percent of the (non-java.lang.Object) class signatures inheriting from it. Table 2 shows how eight out of the top 10 base classes are part of the Android API. The TwitterResponseImpl class in the twitter4j package is from the twitter4j-core library (<http://twitter4j.org>), and the SerializerBase class in the org.codehaus.jackson.map.ser package is from the Jackson mapper library (<http://jackson.codehaus.org>).

Because it's no surprise that mobile apps inherit from platform classes (which developers therefore reuse) such as Activity, we also examined the top 10 inherited domain-specific base classes. The ranks of this group, in the global ranking of base classes, ranged from 7 to 26. Among these top 10, we found classes to

- interface with Twitter;
- handle JSON data;
- interpret the Kawa scheme implementation (www.gnu.org/software/kawa/index.html) used by Android App Inventor—an application to create apps by dragging and dropping visual components (<http://appinventor.mit.edu>);
- interpret JavaScript, HTML5, and CSS3 (used by the PhoneGap framework to create Android apps using Web programming languages [<http://phonegap.com/>] and by the Rhino project to integrate JavaScript in Android apps [<https://developer.mozilla.org/en-US/docs/Rhino>]); and

TABLE 2

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

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

- display advertisements (provided by the Adwhirl library; <http://code.google.com/p/adwhirl/>).

If we compare these findings to prior research on reuse in four open source Java projects,¹⁴ we find that roughly 54.53 percent of the classes in a mobile app inherit from a base class (other than `java.lang.Object`), compared to between 29 and 36 percent 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.

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:

$$\text{PCSR} = 1 - \frac{\text{total number of unique class signatures}}{\text{total number of class signatures}}.$$

A high PCSR value indicates that the reuse of class

signatures is high in that category. Figure 1 shows the PCSR for each category.

Overall, 84.23 percent of class signatures are reused across all categories. Six out of the 30 categories are above the overall PCSR: Games—Brain and puzzles, 89.84 percent; Sports, 88.65 percent; Games—Casual, 87.78 percent; Games—Arcade and action, 86.72 percent; Games—Racing, 84.62 percent; and Music and audio, 84.46 percent. The Comics category has the lowest PCSR, with only 62.72 percent, which is similar to Games—Cards and casino, with a PCSR of 63.59 percent. 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.

Framework Reuse of Whole Apps

In some cases of framework reuse, all classes of an app are 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's 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 data persistence or for look-and-feel purposes). Guido Cardino and his colleagues showed that framework reuse can increase general productivity, among other advantages,¹⁵ but the special case that we consider here 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 :

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

A high value of local reuse means that a high number of class signatures are 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.

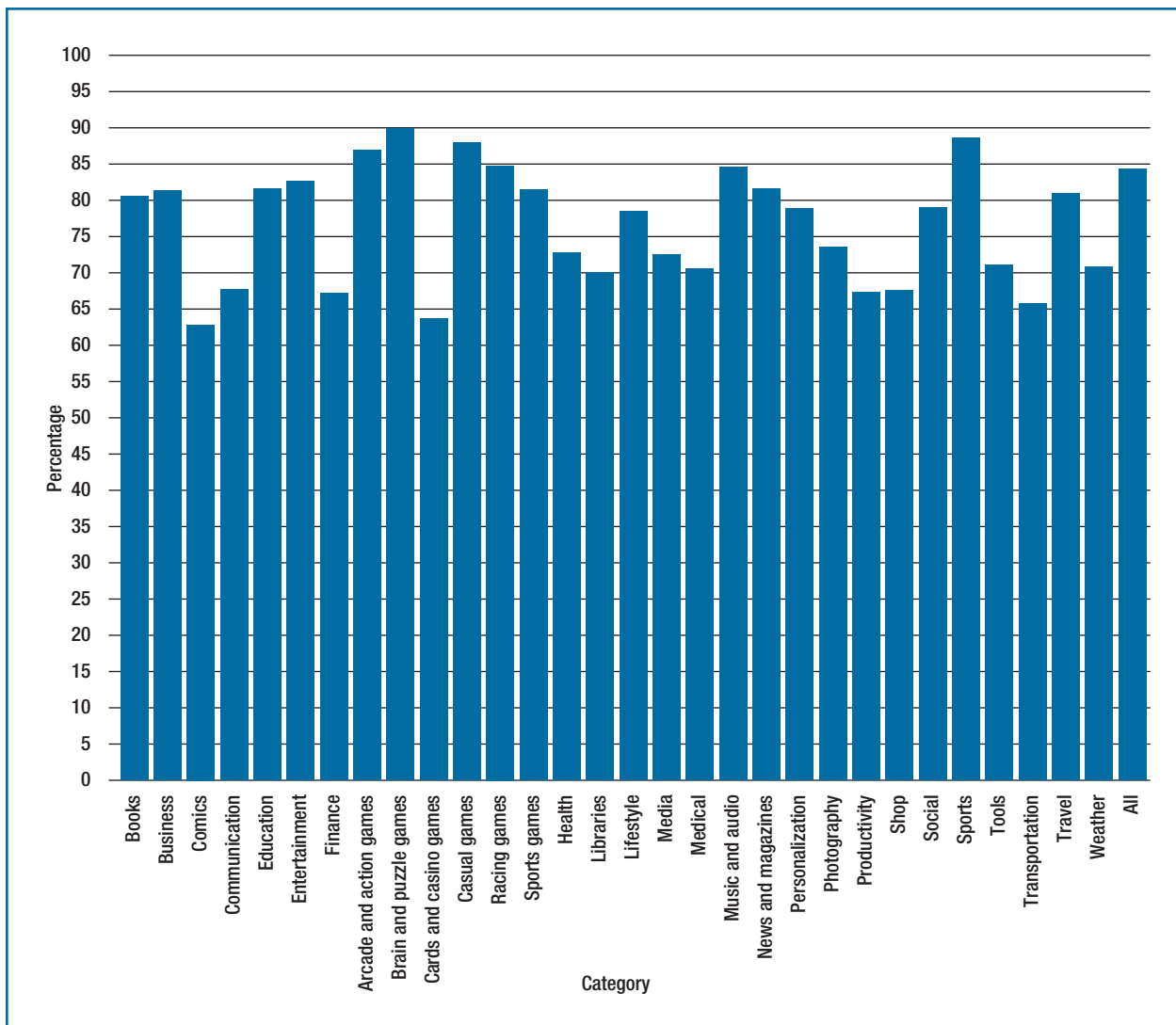


FIGURE 1. Proportion of class signatures reused per category. A high value indicates that very few classes of the apps in the particular category are unique to a single app. When all categories are taken together, only 15.77 percent of the classes are unique.

We found that 1,811 sets of mobile apps had the same set of class signatures, comprising 17,109 individual mobile apps (8.2 percent of all studied mobile apps), and 30 out of the 1,811 sets comprise apps from different categories. The number of classes (app sizes) across the 1,811 different sets of apps where *local reuse=1* is true varies from one to 1,903 classes. Most clusters contain one (81 clusters), eight (63 clusters), or 25 (33 clusters) reused classes, with a median of 61 reused classes. The top three largest clusters consist of 473 apps with one reused class, 339 apps with eight reused classes, and 334 apps with 20

reused classes. The median number of apps across clusters is three apps.

The largest cluster is a set of 473 apps, each of which contains one class. Class *isest.nestmi.IsestApp* is reused by 468 apps in the Games—Arcade and action category, four apps in the Games—Brain and puzzle category, and one app in the Games—Cards and casino category. Three different app developers registered these apps, but their names are similar—for instance, Vital-Game (200 apps), Vital-Game.com (88 apps), and VitalGame (185 apps). Furthermore, the app developers' contact websites

are the same (www.vital-game.com) and state that these apps require Flash to run on Android devices. 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 and magazines category, and are developed by one unique developer (Raycom Media, a TV broadcasting company). Analyzing the classes this set of apps uses shows that they reuse different open source packages, 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 seven clusters (in ascending order by the number of classes):

- 174 apps with 124 classes, distributed across 20 different categories, especially News and magazines (27 apps), Business (24), Communication (22), and Music and audio (22), created by 18 different app developers (contact websites are distinct, which indicates that different app developers created them); AppsBuilder (www.apps-builder.com), a Web app that allows developers to visually build and generate apps without having to program, created 120 apps with package names containing the package `com.appsbuilder`;
- 154 apps with 161 classes, all under the Books and reference category and created by Chinese developer Lexun (these apps have been removed from Google Play, likely by Google);
- 178 apps with 170 classes, all under the Books and reference category and were created by another Chinese developer, 3GQA Dev Team (again, all the apps from this developer were removed from Google Play, likely by Google);
- 106 apps with 178 classes, distributed across 13 different categories, especially Sports (50 apps) and News and magazines (35) and created by 38 different app developers, especially What Ho What Ho (these apps were built using AppYet [www.appyet.com], which helps developers build Android apps without programming);
- 103 apps with 235 classes, distributed across the Games—Casual (99 apps) and News and magazines (4) categories and created by three different app developers who used 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);
- 112 apps with 262 classes, all in the Sports category and 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; and
- 232 apps with 431 classes, distributed across the Lifestyle (169 apps), Entertainment (59), and Education (4) categories. The creator of these apps has three different identifiers, but only one contact website, which again indicates that these apps were built by the same app developer Quipper. (These apps were removed from Google Play, likely by Google.) The apps used different packages such as `com.admob`, `com.facebook`, `org.codehaus.jackson`, and others.

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

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,^{5,6} although many didn't receive a formal training in software engineering.³ It isn't 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 teams, although in recent work, we've found that for these characteristics, mobile apps behave identically to small Unix utility applications.¹⁶ In any case, evidence exists that mobile platforms encourage reuse by making frequently reused apps and libraries a part of the mobile platform itself. For example, this is 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 nonmobile apps.



ISRAEL J. MOJICA is a software engineer at McAfee. His research interests include mobile software and empirical software analysis in general. Mojica received an MS in computer science from Queen's University, Canada. Contact him at Israel_Mojica@McAfee.com.



BRAM ADAMS is an assistant professor at the École Polytechnique de Montréal, where he heads the MCIS (Maintenance, Construction, and Intelligence of Software) lab. His research interests include software release engineering, software integration, software build systems, software modularity, and software maintenance. Adams received a PhD in computer science engineering from Ghent University. He was an organizer of the First International Workshop on Release Engineering (RELENG 13) and is a member of IEEE. Contact him at bram.adams@polymtl.ca.



MEIYAPPAN NAGAPPAN is a postdoctoral fellow in the Software Analysis and Intelligence Lab (SAIL) at Queen's University, Canada. His research interests include deriving solutions that encompass all the various stakeholders of software systems and using large-scale software engineering data to also address the concerns of software operators, build engineers, and project managers. Nagappan received a PhD in computer science from North Carolina State University. He received a best paper award at the International Working Conference on Mining Software Repositories (MSR 12). Contact him at mei@cs.queensu.ca.




STEFFEN DIENST is a PhD student in the Chair of Business Information Systems at the University of Leipzig, Germany. His main research topic is using machine-learning techniques to help monitor the operation of renewable power plants. Other interests range from reverse engineering to functional programming. Dienst received a M.Sc. (Dipl.-Inf.) in computer science from the University of Leipzig. Contact him at sdienst@informatik.uni-leipzig.de



THORSTEN BERGER is a postdoctoral fellow in the Generative Software Development Lab at the University of Waterloo, Canada. His research interests include model-driven development, variability modeling for software product lines and software ecosystems, variability-aware static analyses of source code, and mining software repositories. Berger received a PhD (Dr. rer. nat.) in computer science from the University of Leipzig. Contact him at tberger@gsd.uwaterloo.ca.



AHMED E. HASSAN is the NSERC/BlackBerry Software Engineering Chair at the School of Computing at Queen's University, Canada. His research interests include mining software repositories, empirical software engineering, load testing, and log mining. Hassan received a PhD in computer science from the University of Waterloo. He spearheaded the creation of the Mining Software Repositories (MSR) conference and its research community. Hassan also serves on the editorial boards of *IEEE Transactions on Software Engineering*, *Springer Journal of Empirical Software Engineering*, and *Springer Journal of Computing*. Contact him at ahmed@cs.queensu.ca.

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 its 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. B. Womack, "Google Says 700,000 Applications Available for Android," *Bloomberg News*, 29 Oct. 2012; www.businessweek.com/news/2012-10-29/google-says-700-000-applications-available-for-android-devices.
2. L. Columbus, "Roundup of Mobile Apps & App Store Forecasts, 2013," *Forbes*, 9 June 2013; www.forbes.com/sites/louis columbus/2013/06/09/roundup-of-mobile-apps-app-store-forecasts-2013.
3. A. Craven, *A Demographic and Business Model Analysis of Today's App Developer*, white paper, Gigaom Research, 26 Sept. 2012; <http://appdevelopersalliance.org/files/pages/GigaOMApplicationDevelopers.pdf>.
4. I.J. Mojica-Ruiz et al., "Understanding Reuse in the Android Market," *Proc. IEEE 20th Int'l Conf. Program Comprehension (ICPC 12)*, 2012, pp. 113–122.
5. V.R. Basili, L.C. Briand, and W.L. Melo, "How Reuse Influences Productivity in Object-oriented Systems," *Comm. ACM*, vol. 39, no. 10, 1996, pp. 104–116.
6. W.B. Frakes and K. Kang, "Software Reuse Research: Status and Future," *IEEE Trans. Software Eng.*, vol. 31, no. 7, 2005, pp. 529–536.
7. W. Zhou et al., "Detecting Repackaged Smartphone Applications in Third-party Android Marketplaces," *Proc. 2nd ACM Conf. Data and Application Security and Privacy (CODASPY 12)*, 2012, pp. 317–326.

8. M. Warman, "Fake Android Apps Scam Costs 28,000," *The Telegraph*, 24 May 2012; www.telegraph.co.uk/technology/news/9286538/Fake-Android-apps-scam-costs-28000.html.
9. M.D. McIlroy, "Mass Produced Software Components," *Proc. Software Eng. Concepts and Techniques*, 1969, pp. 138–155.
10. S.W. Ambler, "A Realistic Look at Object-Oriented Reuse," *J. Software Development*, vol. 6, no. 1, 1998, pp. 30–38.
11. S. Dienst and T. Berger, *Static Analysis of App Dependencies in Android Bytecode*, tech. note, Univ. Leipzig, 2012; www.informatik.uni-leipzig.de/~berger/tr/2012-dienst.pdf.
12. D. Kellogg, "40 Percent of U.S. Mobile Users Own Smartphones; 40 Percent are Android," *Nielsen NewsWire*, 2012; www.nielsen.com/us/en/newswire/2011/40-percent-of-u-s-mobile-users-own-smartphones-40-percent-are-android.html.
13. J. Davies et al., "Software Bertillonage: Finding the Provenance of an Entity," *Proc. 8th Working Conf. Mining Software Repositories (MSR 11)*, 2011, pp. 183–192.
14. S. Denier and Y.-G. Gueheneuc, "Mendel: A Model, Metrics, and Rules to Understand Class Hierarchies," *Proc. 16th IEEE Int'l Conf. on Program Comprehension (ICPC 08)*, 2008, pp. 143–152.
15. G. Cardino, F. Baruchelli, and A. Valerio, "The Evaluation of Framework Reusability," *ACM SIGAPP Applied Computing*, vol. 5, no. 2, 1997, pp. 21–27.
16. M.D. Syer et al., "Revisiting Prior Empirical Findings for Mobile Apps: An Empirical Case Study on the 15 Most Popular Open Source Android Apps," *Proc. 2013 Conf. Center for Advanced Studies on Collaborative Research (CASCON 13)*, IBM, 2013, pp. 283–297.
17. M.D. Syer et al., "Exploring the Development of Micro-apps: A Case Study on the BlackBerry and Android Platforms," *Proc. IEEE 11th Intl. Working Conf. Source Code Analysis and Manipulation (SCAM 11)*, 2011, pp. 55–64.



Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.

IEEE SOFTWARE CALL FOR PAPERS

Software Engineering for Internet Computing: Internetwork and Beyond

Submission deadline: 1 June 2014 • Publication: January/February 2015

The Internet, once a network of networks, has become not just the platform of choice for delivering services to increasingly mobile users, but also the connective tissue among people, information, and things. The newest and most popular computing and application paradigms have been born on the Internet, or at least motivated by it, such as Web 2.0, social networking, mobile Internet, cloud computing, the Internet of things, and big data.

This special issue seeks articles that explore state-of-the-art research and industry practices of software engineering for Internet computing. Topics of interest include but are not limited to:

- software and programming models for dominant and emerging Internet-based systems such as cloud computing, service computing, social computing, mobile Internet, Internet-of-things, and cyber-physical systems;
- platforms and application frameworks for Internet-based software, such as Web-based integration (for example, REST and JSON), infrastructure provisioning and deployment (for example, OpenStack and Capistrano), Web-scale data analytics and content handling (for example, MongoDB and Hadoop);
- engineering and quality-assurance approaches for Internet-based software;
- software design models for Internet-based software, such as UML, BPM, and Petri Net;
- software development processes and tools for the Internet (for example, agile development for Internet-based software), or with the Internet (for example, cloud-based development environments);
- technology and human-interaction models and techniques in the development of Internet-based software;
- migration or integration of legacy software to Internet-based software; and
- case studies and experience reports on one or more of the above aspects in industry practices.

Questions?

For more information about the focus, contact the guest editors:

- Tao Xie (taoxie@illinois.edu)
- Antonia Bertolino (antonia.bertolino@isti.cnr.it)
- M. Brian Blake (M.Brian.Blake@miami.edu)
- Pankaj Mehra (dr.pankaj.mehra@gmail.com)
- Hong Mei (meih@pku.edu.cn)

Submission Guidelines

Manuscripts must not exceed 4,700 words including figures and tables, which count for 200 words each. Submissions in excess of these limits may be rejected without refereeing. The articles we deem within the theme and scope will be peer-reviewed and are subject to editing for magazine style, clarity, organization, and space. We reserve the right to edit the title of all submissions. Be sure to include the name of the theme or special issue for which you are submitting.

Articles should have a practical orientation and be written in a style accessible to practitioners. Overly complex, purely research-oriented or theoretical treatments are not appropriate. Articles should be novel. *IEEE Software* does not republish material published previously in other venues, including other periodicals and formal conference/workshop proceedings, whether previous publication was in print or electronic form.

Full author guidelines: www.computer.org/software/author.htm

Submission details: software@computer.org

Submit an article: <https://mc.manuscriptcentral.com/sw-cs>