SAMOA — A Visual Software Analytics Platform for Mobile Applications

Roberto Minelli and Michele Lanza
REVEAL @ Faculty of Informatics — University of Lugano, Switzerland

***Abstract*—Mobile applications, also known as *apps*, are dedicated software systems that run on handheld devices, such as smartphones and tablet computers. The apps business has in a few years turned into a multi-billion dollar market. From a software engineering perspective apps represent a new phenomenon, and there is a need for tools and techniques to analyze apps.**

We present SAMOA, a visual web-based software analytics platform for mobile applications. It mines software repositories of apps and uses a set of visualization techniques to present the mined data. We describe SAMOA, detail the analyses it supports, and describe a methodology to understand apps from a structural and historical perspective.

The website of SAMOA, containing the screencast of the tool demo, is located at `http://samoa.inf.usi.ch/about`

I. INTRODUCTION

Mobile applications, or *apps*, are custom software systems running on handheld devices, *i.e.,* smartphones and tablet PCs. The world of apps is variegated: Each vendor imposes a number of constraints (*e.g.,* the programming language and development environment to be used), provides specific design guidelines, and offers its own distribution channel (*e.g.,* Android's Google Play, Apple's App Store). The market of apps is remarkable: Apps generated a revenue of $4.5 billion USD in 2009 [1], and the business is expected to be worth $25 billion USD [2] a few years from now.

The Apple and Google stores provide ca. one million apps for download. With their increasing popularity, apps are becoming an important software engineering domain. Apps represent a new phenomenon but, as any software system, they will inevitably face evolution, maintenance, and comprehension problems. It is unclear whether existing approaches for program comprehension and maintenance [3], [4], [5] can be ported to apps, since they were devised before apps existed.

We devised a novel approach to analyze apps [6] and implemented SAMOA, a web-based software analytics platform for apps[1]. SAMOA mines software repositories of apps and uses a set of visualization techniques to present the mined data. SAMOA offers a catalogue of custom views to understand the structure and evolution of apps. Both analysts and developers interested in comprehending apps can benefit from these visualizations.

We used SAMOA to investigate part of the F-Droid repository[2]. We discovered, for example, that inheritance is essentially unused in apps, that apps heavily rely on 3^{rd}-party APIs, and that most apps are short-lived single developer projects.

[1]See `http://samoa.inf.usi.ch`
[2]See `http://f-droid.org`

Related work. Since the first apps were developed only a few years ago, there is little directly related work. Ruiz *et al.* focused on software design aspects of apps, namely on reuse by inheritance and class reuse [7]. They divided apps in categories (*e.g.,* casino, personalization, photography), and found that more than 60% of all classes in each category appear in more than two other apps. Hundreds of apps were entirely reused by other apps in the same category. Harman *et al.* introduced "App Store Mining" [8], a novel form of software repository mining. They mined the Blackberry app store and studied a number of correlations between different features of apps.

Differently from Harman *et al.* we want to focus on the source code of apps, rather than on app stores. Our goal is to understand the differences between apps and traditional software systems, and the implications for the maintenance and comprehension of apps. The novelty of apps explains the small amount of related work, but also calls for novel tools and techniques to analyze apps.

We present SAMOA, our visual web-based software analytics platform for mobile applications. SAMOA leverages three factors for the analysis: source code, usage of 3^{rd}-party libraries, and historical data. SAMOA presents the data to the user by means of a catalogue of interactive visualizations. The views are enriched with traditional software metrics complemented by domain-specific ones. SAMOA provides a custom snapshot view to depict a specific revision of one app, a evolution view to present historical aspects of one app, and ecosystem views to depict several apps at once.

II. SAMOA: A VISUAL SOFTWARE ANALYTICS PLATFORM FOR APPS

Figure 1 depicts the main user interface of SAMOA presenting a snapshot view of the ALOGCAT application. The UI is composed of five parts: a (1) *Selection panel* that allows the user to pick the app to be analyzed, and to switch between the different visualizations SAMOA provides; a (2) *Metrics panel* which summarizes a set of metrics in sync with the visualization, being a specific revision of an app (*i.e.,* snapshot) or global measurements about the apps ecosystem; a (3) *Revision info panel* that displays information about a specific revision of an app; an (4) *Entity panel* displaying additional details about the entity in focus; and the (5) *Main view*, the remaining surface dedicated to the interactive views.

Figure 1 illustrates also how we enhance the view using colors and metrics.

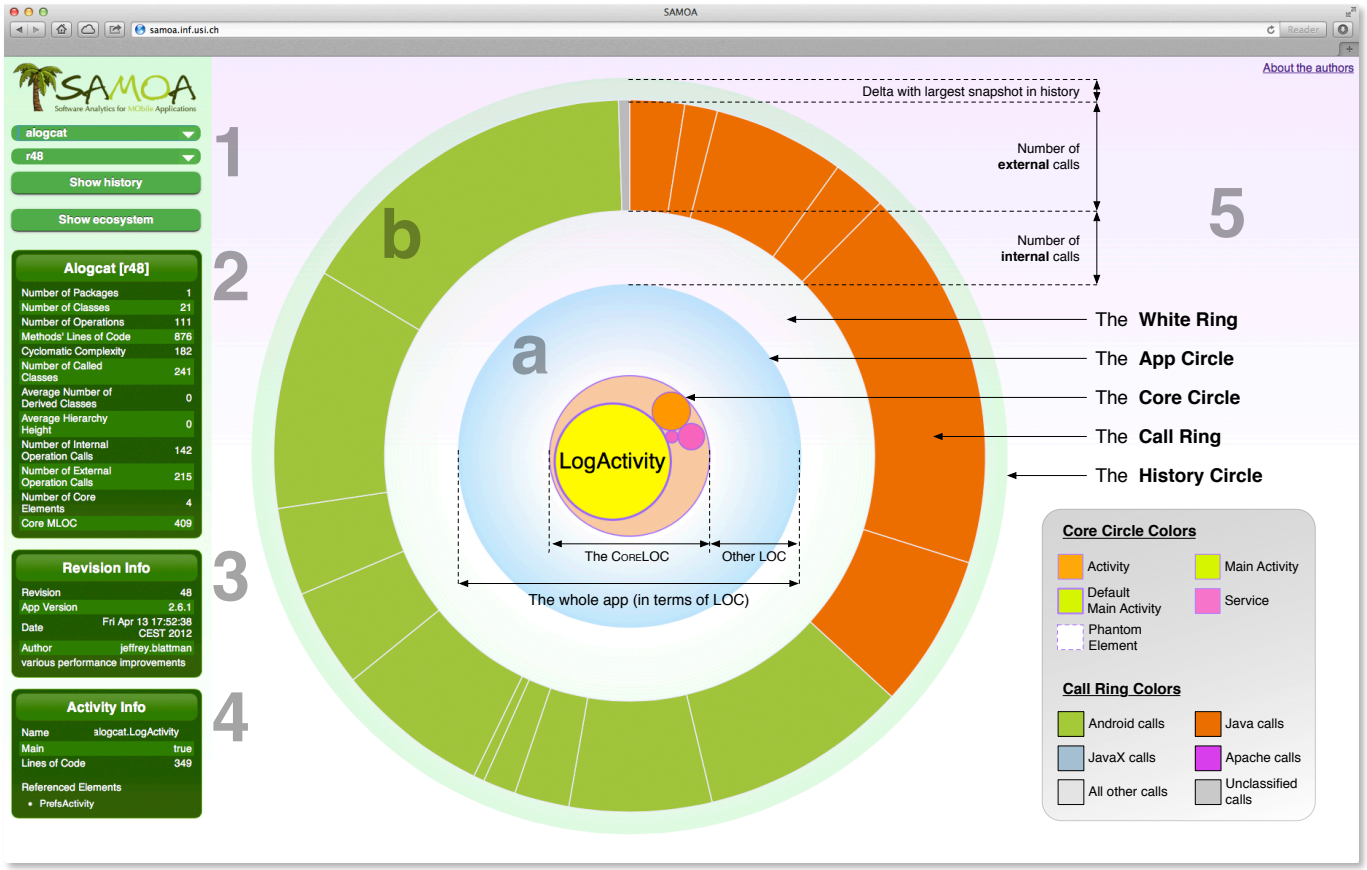

Fig. 1: Samoa: our web-based software analytics platform for mobile applications, depicting a snapshot of Alogcat.

A. Visualizations

To better comprehend mobile applications Samoa provides visualizations at three different granularities: The *"snapshot view"* depicts a specific revision of an app; *"evolution views"* present the evolution of an app over its history; and *"ecosystem views"* depict more than one app at once.

a) Snapshot view: Figure 1 shows the 48^{th} revision of the Alogcat app[3]. This view presents the most important structural properties of an app at hand. Two main parts compose our *"snapshot view"*: the central section (1.a) and a ring (1.b).

The central section of the view presents the entire app in terms of classes and lines of code (LOC). Among all classes, we define as *"Core Elements"* the entities specific to the development of apps (*i.e.,* which inherit from the mobile platform SDK's base classes). This section is composed of an *"App Circle"* (*i.e.,* shaded blue) and a *"Core Circle"* (*i.e.,* light red). The former represents the entire app in terms of LOC, while the latter depicts the core of the app. Circles inside the *"Core Circle"* are *"Core Elements"* (*i.e.,* Java classes), where the radius of each circle is proportional to the value of LOC of the entity represented, and color & stroke provide additional information about the type of the entity (*e.g.,* Activity, Service, Main Activity).

[3]See `http://code.google.com/p/alogcat/`

The radius of the *"App Circle"* is proportional to the number of LOC of the app at the current revision while the radius of the *"Core Circle"* is proportional to the sum of the values of LOC of *"Core Elements"*.

The *"Call Ring"* (Figure 1.b) uses a circular layout to depict the 3^{rd}-party APIs calls the app makes. Its thickness and total span are proportional to the number of external method calls. Each slice of the *"Call Ring"* represents calls to a distinct 3^{rd}-party library (*e.g.,* Apache, JavaX), where colors distinguish calls to different libraries. We use specific colors to depict calls to the four most employed libraries and two tones of gray: one for all calls to other libraries and one for the calls Samoa is not able to identify. The angle spanned by an arc is proportional to the number of method calls it represents. The *"Historical Circle"* (*i.e.,* green shaded) represents the maximum size of an app over its entire history, thus if there is a gap (as in Figure 1) we know that the currently visualized snapshot is not the largest in the history of the app. The number of internal calls (*i.e.,* calls implementing internal behavior of the app) is represented by the thickness of the *"White Ring"*. With this visual cue, we can infer the ratio between internal and external calls. The outer radius of the *"Call Ring"* indicates the size of a snapshot, considering both LOC (*i.e.,* the radius of *"App Circle"*) and method calls (*i.e.,* thickness of the *"Call Ring"* and the *"White Ring"*).

b) Evolution view: SAMOA uses stacked bar charts and line charts to present different types of evolutionary information (*e.g.,* LOC, external calls, or core elements) about an app, considering all the snapshots available to SAMOA.

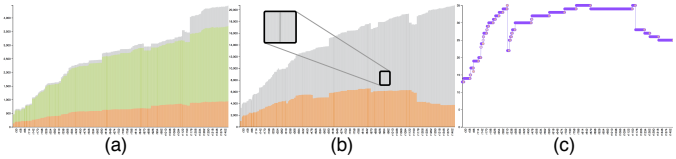

Fig. 2: Evolution views of the CSIPSIMPLE app in terms of (a) 3^{rd}-party calls, (b) LOC, and (c) number of *"Core Elements"*.

Figure 2 depicts the *evolution views* of the CSIPSIMPLE app[4], in terms of (a) 3^{rd}-party calls, (b) LOC, and (c) number of *"Core Elements"*. In the bar chart views, each bar represents a snapshot of the app, divided into layers, according to the type of data presented, *e.g.,* Figure 2.b depicts the evolution LOC, thus layers are CORELOC (*i.e.,* red) and non-CORELOC (*i.e.,* grey). The height of each bar represents the value of a specific software metric, in this example the value of LOC. As highlighted in Figure 2.b, we use opacity to denote an app's release versions: Darker bars are snapshots whose release number changed. SAMOA uses line charts to depict data without a logical layer subdivision, *e.g.,* the evolution of the number of *"Core Elements"* presented in Figure 2.c.

c) Ecosystem view.: *Ecosystem views* depicts several apps at once, using stacked bar charts or a grid layout. Figure 3 depicts an ecosystem view of 12 apps, sorted according to their size (*i.e.,* in terms of LOC), arranged using a grid layout.

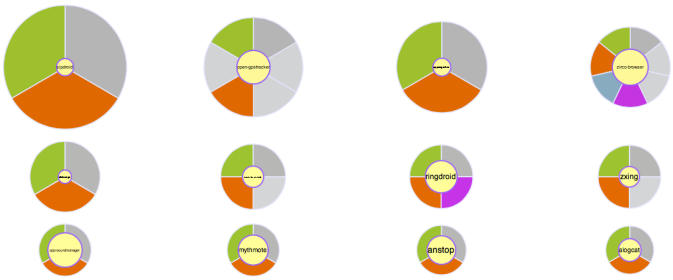

Fig. 3: An ecosystem view of 12 apps, sorted by total LOC.

Each shape is a simplified snapshot view of an app where the radius of the core (*i.e.,* yellow) corresponds to the number of CORELOC. The radius is proportional to the total number of LOC, the span of the *"Call Ring"* shows the proportions of external calls (either with unary or proportional weights).

B. User Interactions

All the visualizations offered by SAMOA are interactive. For example, by hovering on a shape, the entity is highlighted and the "entity panels" of SAMOA provides additional information about the shape in focus (see Figure 1.4). The user can freely zoom and pan the snapshot view. On clicking on a core element, SAMOA displays its source code. In the visualizations based on bar charts, the data can be re-ordered and the user can choose to display layers either grouped or stacked (*i.e.,* default). In both the evolution and ecosystem view clicking on a shape leads to the corresponding snapshot view of the app.

C. A Methodology to Understand Apps

SAMOA provides views at different granularities, with different purposes and applications, which we described in previous work [6]. The typical methodology is to use ecosystem views to get a "big picture" of several apps at once, and then drill down using the evolution view which, in turn, help us to understand where to use the snapshot view.

D. Under the Hood: Architecture and Technologies

SAMOA is composed of a back-end and a front-end, as Figure 4 depicts. The back-end, entirely written in Java, is responsible for a number of tasks: (1) it mines software repositories of apps and extracts apps-specific data from different artifacts. Then (2) it processes the data by, extracting and parsing two different source code representations, namely the AST (Abstract Syntax Tree) and the MSE[5]. From these two representations, plus the Android manifest[6], SAMOA (3) extracts a set of software metrics and (4) generates JSON files that are served to the front-end, implemented using PHP, HTML5, and JavaScript (*i.e.,* views are generated using d3.js).

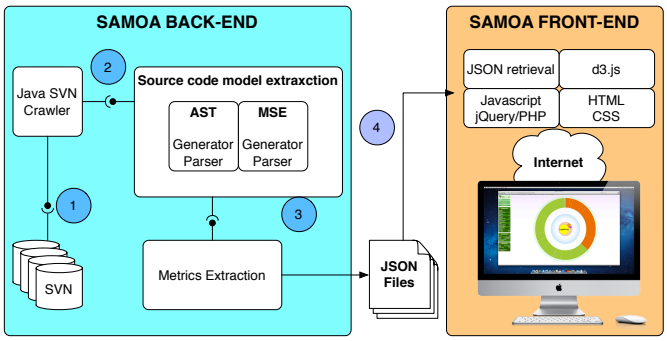

Fig. 4: An architectural overview of SAMOA.

III. A CATALOGUE OF PECULIARITIES OF APPS

Following the methodology described in Section II-C, we devised a catalogue of peculiarities of apps [6]. For example, we can order our ecosystem of apps according to their number of revisions, and investigate on the app with the longest history.

Example I: Figure 5.a shows part the evolution of LOC of ZXING[7], the app with the longest history in our apps ecosystem, with more than 2.2k commits. Android apps have a configuration file (*i.e.,* called manifest) which identifies the *"Core Elements"* of the app. To work properly, an app requires the manifest file to be in sync with the source code.

[4]See `http://code.google.com/p/csipsimple/`

[5]See `http://www.moosetechnology.org/docs/mse`

[6]See `http://goo.gl/Rt6GD`

[7]See `http://code.google.com/p/zxing/`

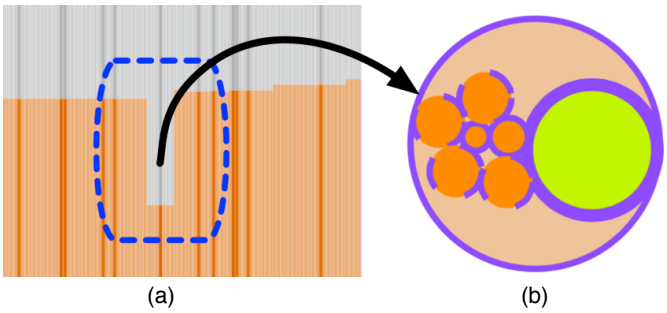

Fig. 5: (a) Part of the evolution of LOC of the ZXING app and (b) one of the snapshot in which the manifest is out of sync.

Android developers should maintain this file in synch manually (*i.e.,* reflecting the changes performed in the source code to the manifest file). During our app analysis we discovered that sometimes developers forgot to keep the manifest updated, introducing bugs in their apps, as in the case of ZXING. In the highlight of Figure 5, there is a time interval in which CORELOC drop and suddenly they increase again. With further investigation, we discovered that the authors have moved some functionalities into sub-packages, but they forgot to update the references in the manifest, preventing both the Android OS and SAMOA to recover links to the *"Core Elements"*. Figure 5.b shows that SAMOA is not able to correctly recover *"Core Elements"*, and depicts some of them as "phantom elements."

Example II: Apps should conform to a set of sound guiding principle. The Android documentation, for example, recommends that apps should have one "Main Activity" and, if not so, they must have a single "Default Main Activity": The real main activity to invoke. In our app analysis, we observed many apps have more than one main activity.

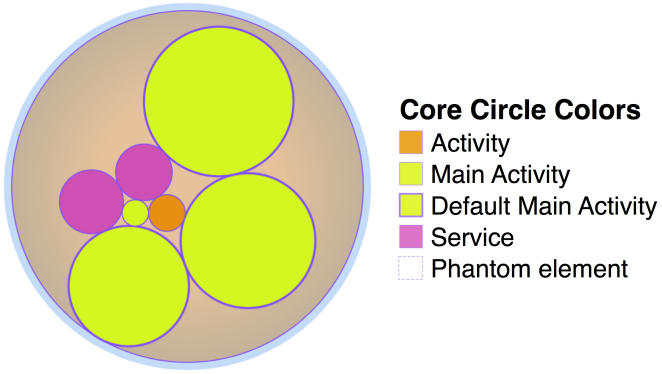

Fig. 6: The central section of the snapshot view of APP-SOUNDMANAGER app at revision 106.

Figure 6 depicts the 106^{th} snapshot of APP-SOUNDMANAGER[8]. This apps lists 4 Main Activities (*i.e.,* yellow), out of which 3 "default" main activities (*i.e.,* thicker stroke in our snapshot view), violating the said Android guideline.

[8]See `http://code.google.com/p/app-soundmanager/`

Example III: In all Java systems, including Android apps, 3^{rd}-party APIs are reused by including JAR files in the build path. Developers of apps have a tendency of directly importing the entire source code of 3^{rd}-party libraries instead of adding the needed JAR files, which is a questionable practice from a legal point of view.

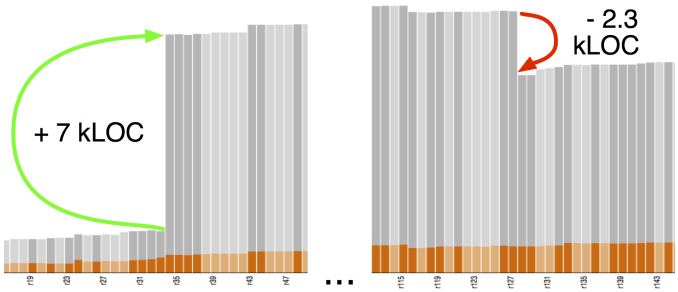

Fig. 7: Part of the evolution of LOC of APPS ORGANIZER.

Figure 7 depicts the evolution of the number of LOC of the APPS ORGANIZER[9] application. At some point the authors added the source code (≈7 kLOC) of the Trove library[10]. Later on, they removed part of the same library (≈2.3 kLOC).

IV. CONCLUSION

We presented SAMOA, a novel web-based software analytics platform, which supports our approach [6] to analyze apps from several points of view, using custom views tailored to the novel domain of mobile applications, offering several means to navigate and inspect information.

Acknowledgements. We gratefully acknowledge the Swiss National Science foundation's support for the project "HI-SEA" (SNF Project No. 146734).

REFERENCES

[1] R. Islam, R. Islam, and T. Mazumder, "Mobile application and its global impact," *IJEST*, 2010.

[2] Markets and Markets, "Global mobile application market (2010–2015)," 2010.

[3] M. Lehman, D. Perry, and J. Ramil, "Implications of evolution metrics on software maintenance," in *Proceedings of ICSM*, 1998, p. 208.

[4] H. Gall, M. Jazayeri, R. Klösch, and G. Trausmuth, "Software evolution observations based on product release history," in *Proceedings of ICSM*, 1997, pp. 160–166.

[5] W. Turski, "The reference model for smooth growth of software systems revisited," *TSE*, pp. 814–815, 2002.

[6] R. Minelli and M. Lanza, "Software Analytics for Mobile Applications - Insights & Lessons Learned," in *Proceedings of CSMR*, 2013, pp. 144–153.

[7] I. Ruiz, M. Nagappan, B. Adams, and A. Hassan, "Understanding reuse in the android market," *ACM-ICPC*, 2012.

[8] M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: MSR for app stores," in *Proceedings of MSR*, 2012.

[9]See `http://code.google.com/p/appsorganizer/`
[10]See `http://trove.starlight-systems.com/`