LETTER

Semi-automatically Extracting Features from Source Code of Android Applications

Tetsuya KANDA[†a)], **Yuki MANABE**[†b)], ***Nonmembers***, **Takashi ISHIO**[†c)], ***Member***,
Makoto MATSUSHITA[†d)], ***Nonmember***, ***and*** **Katsuro INOUE**[†e)], ***Fellow***

SUMMARY It is not always easy for an Android user to choose the most suitable application for a particular task from the great number of applications available. In this paper, we propose a semi-automatic approach to extract feature names from Android applications. The case study verifies that we can associate common sequences of Android API calls with feature names.

key words: *Android, feature extraction, software categorization, API*

1. Introduction

Android is one of the most popular platforms for mobile phones and tablets. A user can search and choose from more than 600,000 Android applications in Google Play [1]. Because there are so many choices, however, selecting an appropriate application is not a trivial task. For example, in November 2012, at least 1,000 applications could be found when searching with the keyword "calculator" on Google Play.

A simple but important criterion for selection of an application is the set of features it provides. Investigating the features by trying each application, however, is time consuming. Although documentation is an important source of information, many applications are less than adequate in this area.

MUDABlue [2] and LACT [3] are the solutions that enable users to focus on a set of similar applications. These approaches automatically categorize applications with similar features based on characteristics of the source code. While they can extract a set of similar applications, they cannot show a list of the features provided by the applications in a specific category.

In this study, we propose a semi-automatic approach to extracting features from Android applications. The promise of our proposed solution is that a feature can be associated with a particular sequence of API calls. API calls are used to control GUI components, network connections, and hardware devices such as a camera, GPS, or touch screen. Although software developers can use arbitrary sequences of API calls, we hypothesize that a popular feature of an application is likely to be implemented by the same sequence of API calls, since similar applications use the same set of APIs [4]. In our proposed solution, therefore, we automatically extract common sequences of API calls in two or more applications, and manually associate each of these with a feature name. We use the associations as a knowledge-base. We then automatically extract API calls from other target applications and, using our knowledge-base, output feature names associated with the API calls.

It should be noted that this paper is a revised version of our technical report [5]. The technical report describes the detailed implementation of our method. We compare extracted features from applications in this paper while we compared sequence of API calls of applications directly in the technical report, yet it does not affect the result of the case study.

The next section describes the technical details of our approach. Section 3 shows the results of a case study. In Section 4, we conclude this article with a discussion of future work.

2. Associating API Calls with Feature Names

The objective of our study is to extract a list of features from multiple applications and build a knowledge-base. A user can then more easily compare the features of two or more applications. Our approach has two phases: build a knowledge-base from a set of applications, and, using the knowledge-base, extract and list the features from another set of applications.

Our knowledge-base is defined as a set of associations $\langle S, f \rangle$, where S is a sequence of API calls and f is a feature name. We build a knowledge-base using the following three steps.

Step 1 We translate each application into a set of sequences of API calls. As Android applications are written in Java, we extract a sequence of Android API calls from each method of the application. A method call is identified by its name and the receiver type declared in the source code. We recognize an Android API as any method call whose fully qualified class names start with "android." or "com.google.android.". Figure 1 shows an example of a sequence of API calls extracted from a method in an application. API calls in a sequence are sorted by line number. If

Manuscript received January 1, 2011.
Manuscript revised January 1, 2011.

[†]Graduate School of Information Science and Technology, Osaka University,
1-5, Yamadaoka, Suita, Osaka, 565-0871, Japan
a) E-mail: t-kanda@ist.osaka-u.ac.jp
b) E-mail: y-manabe@ist.osaka-u.ac.jp
c) E-mail: ishio@ist.osaka-u.ac.jp
d) E-mail: matusita@ist.osaka-u.ac.jp
e) E-mail: inoue@ist.osaka-u.ac.jp

DOI: 10.1587/transinf.E0.D.1

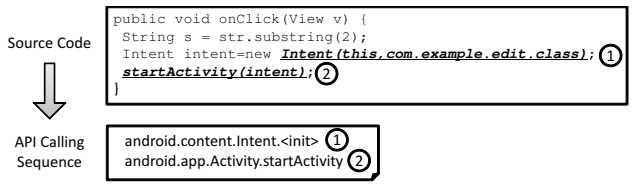

Fig. 1 Extracting sequences of API calls

two or more API calls are involved in a single line, they are sorted in alphabetical order. As a result, N applications are translated into sets $Apps = \{A_1, \cdots, A_N\}$, where A_i is a set of sequences of API calls extracted from the i-th application.

Step 2 We extract common sequences of API calls involved in at least two applications as candidates for features. We compute a set of common sequences as follows:

$$C = \bigcup_{A_i, A_j \in Apps, i \neq j} \{LCS(s,t) | s \in A_i, t \in A_j\}$$

where $LCS(s,t)$ is the longest common subsequence of two sequences s and t. We exclude sequences that consist of only a single API call from C. We denote the resultant set by $CommonAPI$.

Step 3 We manually associate each sequence S in $CommonAPI$ with a feature name, f, and store the association $\langle S, f \rangle$ in a knowledge-base. A feature name can be associated with a sequence if the sequence controls a particular device or system component, because application features are often characterized by devices and components used by the application.

We use the knowledge-base to translate API calls in a target application into feature names. If an application involves a sequence S', including a subsequence S, matching an association $\langle S, f \rangle$ in the knowledge base, we output f as a feature of the application.

3. Case Study

We conducted a case study to evaluate whether our approach could extract the features of applications. We collected 11 applications labeled "Map" in Google Code as shown in Table 1. We built a knowledge-base from six applications (KB1-KB6) and then used it to extract features from the remaining applications (T1-T5).

We extracted 156 common API calling sequences from the six applications. The first author of this paper could associate names with 23 out of the 156 sequences. Table 2 shows an example of the sequences and their feature names. The feature names simply describe what components are controlled by the API sequences. In this example, "Alert dialog," "Submenu," and "Show Toast (pop-up message)" are related to the user interface, while "Get Location" and "Set Location" are related to map features. Using the knowledge-base, we then extracted a list of features for each application (T1-T5). Table 3 shows the features found in the target applications. From these results, without using the applications, we could observe that T1 and T5 can specify a location on a map and that T2 is probably not a map viewer.

It should be noted, as we hypothesized, that 18 of the 23 identified API calling sequences are involved in at least one target application. This result is promising because it indicates that a small knowledge-base could cover the popular features of many applications in the same category.

4. Conclusion

We proposed an approach to extracting features from an Android application using a knowledge-base built from source code of applications. The results of a case study showed that our approach could extract features from an application and list them in terms of devices and components used by the applications. Although our approach is promising, we were unable to represent the usage or purpose of the components. We also could not recognize features implemented by general-purpose GUI components. To resolve this problem, we intend to enhance our approach using information about data names and types used in applications. In addition, we would like to use our approach to understand the variability of software product lines in our future work.

Table 1 Applications used in the case study

ID	Application name	LOC	#API calls
KB1	OpenGPSTracker	8122	1099
KB2	mapsforge	37326	1407
KB3	OSMandroid	3150	175
KB4	TripComputer	14487	825
KB5	shareyourdrive	2761	346
KB6	savage-router	1041	66
T1	MapDroid	6387	1160
T2	cycroid	1278	761
T3	yozi	5348	159
T4	maps-minus	1785	218
T5	BigPlanetTw	4139	432

Table 2 Example of sequence of API calls

Feature name	Sequence of API calls
Alert dialog	android.app.AlertDialog.Builder.<init> android.app.AlertDialog.Builder.setTitle
Get Location	android.location.Location.getLatitude android.location.Location.getLongitude
Show toast (pop-up message)	android.widget.Toast.makeText android.widget.Toast.show
Set Location	android.location.Location.setLatitude android.location.Location.setLongitude
Submenu	android.view.Menu.addSubMenu android.view.SubMenu.setIcon

Table 3 Features identified in five applications

ID	T1	T2	T3	T4	T5
Alert Dialog	✓	✓	✓	✓	✓
Get Location	✓		✓	✓	✓
Show Toast (pop-up message)	✓	✓		✓	✓
Set Location	✓				✓
Submenu					✓

Acknowledgement

This work was supported by JSPS KAKENHI Grant Number 23680001.

References

[1] "Android Apps in Google Play - The year of opportunity." http://commondatastorage.googleapis.com/io2012/presentations/live%20to%20website/123.pdf.

[2] S. Kawaguchi, P.K. Garg, M. Matsushita, and K. Inoue, "MUDABlue: An automatic categorization system for open source repositories," Proc. Asia-Pacific Software Engineering Conference, pp.184–193, Nov. 2004.

[3] K. Tian, M. Revelle, and D. Poshyvanyk, "Using latent Dirichlet allocation for automatic categorization of software," Proc. International Working Conference on Mining Software Repositories, pp.163–166, May 2009.

[4] C. McMillan, M. Linares-Vasquez, D. Poshyvanyk, and M. Grechanik, "Categorizing software applications for maintenance," Proc. International Conference on Software Maintenance, pp.343–352, Sept. 2011.

[5] T. Kanda, Y. Manabe, T. Ishio, M. Matsushita, and K. Inoue, "A prototype of comparison tool for Android applications based on difference of API calling sequences," IEICE Technical Report, SS2010-10, vol.111, no.107, pp.35–40, Jun 2011.