LETTER

Extraction of Library Update History Using Source Code Reuse Detection

Kanyakorn JEWMAIDANG[†], ***Nonmember***, **Takashi ISHIO**[††a)], **Akinori IHARA**[††], **Kenichi MATSUMOTO**[††], ***and*** **Pattara LEELAPRUTE**[†], ***Members***

SUMMARY This paper proposes a method to extract and visualize a library update history in a project. The method identifies reused library versions by comparing source code in a product with existing versions of the library so that developers can understand when their own copy of a library has been copied, modified, and updated.

key words: *visualization, software reuse, source code similarity, repository mining*

1. Introduction

Software reuse is important to improve software reliability [1]. Software systems in the industry increasingly use open source software components due to their reliability and cost benefits [2]. While some libraries are available in binary forms, software developers often clone-and-own existing software components for their products [3], [4].

Application developers using a library should replace their library copy with a new version, if the copied version of the library included severe problems such as security vulnerabilities. Adopting a new version of a library may seem a simple task, but have many difficulties [5]. Knowledge about a system's past upgrade activity with respect to a library can help maintainers [6].

To analyze library update activity, Kula et al. [6] proposed a visualization technique using configuration files for a dependency management system. On the other hand, there exists no appropriate tool to understand library update activity of a project using clone-and-own approach. Xia et al. [7] reported that source code reuse activity is often unrecorded, based on their manual analysis of source code repositories. Hence, an automatic analysis tool is necessary to track copied libraries in a project. To identify the original version of a copied library, Ishio et al. [8] proposed a source file comparison method. The method uses aggregated file similarity to effectively identify an original version.

In this paper, we propose a method to extract and visualize a library update history to investigate when developers adopt, update or change their copied library in the version history of a product. To identify the original version of a library copy, we employed a modified version of the source file comparison method [8]; we compare source files between repositories. The identified library versions in each of product versions are visualized in a Sankey diagram [9].

Section 2 explains our extraction method. Section 3 shows a case study that analyzes a library reused in Android project. Section 4 summarizes the current state and future directions of the research.

Manuscript received September 20, 2017.
Manuscript revised December 1, 2017.
Manuscript publicized December 20, 2017.

[†]The authors are with the Department of Computer Engineering, Kasetsart University, Bangkok, 10900 Thailand.
[††]The authors are with the Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma-shi, 630–0192 Japan.
a) E-mail: ishio@is.naist.jp

DOI: 10.1587/transinf.2017EDL8205

2. Extraction of Library Update History

Our method visualizes how software projects reuse and update their own copy of a library. The method compares a set of official versions of a library and a set of release versions of a software product including the library. It identifies reuse pairs between product versions and library versions, and then visualizes the pairs using a Sankey diagram.

The extraction step identifies a set of source code reuse pairs $Reuse(P, L)$ for a set of product versions P and a set of library versions L based on source code similarity as follows.

$$Reuse(P, L) = \{\, (p \in P, l \in L) \mid S(p, l) = \max_{l' \in L} S(p, l') \,\}$$

where $S(p, l)$ is an aggregated file similarity value of a product version p and a library version l [8]. In $Reuse(P, L)$, each product version $p \in P$ is corresponding to only one version of library l whose source files are the most similar to the product source code. We assume that a product p includes at most one version of a library. A library version may be included in a number of product versions.

The aggregated file similarity approximates the amount of source code that can be reused from l to create files in p. It is defined by the sum of file similarity values:

$$S(p, l) = \sum_{f_p \in F_p} \max\{sim(f_p, f_l \in F_l) | sim(f_p, f_l) \geq th\}$$

where F_p represents a set of files in the product p, F_l represents a set of files in the library l, respectively. The similarity function $sim(f_p, f_l)$ is defined as Jaccard index of token trigrams.

$$sim(f_p, f_l) = \frac{|trigrams(f_p) \cap trigrams(f_l)|}{|trigrams(f_p) \cup trigrams(f_l)|}$$

The trigrams ignore whitespace and comments. We use

$th = 0.9$ to ignore lower similarity values. The threshold accurately identifies source files in a product that are reused from a library [8].

The definition of $Reuse(P, L)$ selects a library version l in L that has the highest value of $S(p, l)$ for each version of p. For a particular version of p, if a library version l_1 has more similar files than another version l_2 (i.e., $S(p, l_1) > S(p, l_2)$), l_1 is more likely an original version. We also use $S(p, l)$ to analyze different versions of a product against a particular version of a library. If the value of $S(p_1, l)$ equals to $S(p_2, l)$, we consider two versions of a product p_1 and p_2 have the same library files reused from a version of a library l. It should be noted that we do not directly compare $S(p_1, l_1)$ with $S(p_2, l_2)$ ($p_1 \neq p_2, l_1 \neq l_2$), because it is less meaningful.

The extracted pairs $Reuse(P, L)$ are visualized as a Sankey diagram, that is a visualization used to depict a flow from one set of values to another. In our visualization method, the diagram lists product versions on the left side, library versions on the right side, and connects their versions. Since a large number of links may result in a complicated figure, we filter out versions whose revision numbers are different but whose aggregated file similarity values are the same as the adjacent product versions.

To investigate library reuse and update activity according to the time axis, both product and library versions are sorted by their release date. The oldest versions are placed at the top of a diagram, the latest versions are placed at the bottom, respectively. The diagram links versions with colors indicating different aggregated similarity values. The size of a link between a library and a product indicates the similarity values, i.e., the amount of reused source code of the library and the product.

3. Case Study

To investigate the effectiveness of our extraction approach, we have applied our visualization to libraries reused in Android project. The Android project keeps libraries customized for Android in Git source code repositories [10]. The libraries are stored in individual repositories, separately from one another. Source code in the repositories have Android's version numbers, instead of library version numbers.

We choose three popular libraries written in C/C++ and Java: `junit` [11], `slf4j` [12], and `easymock` [13]. Their copies are located in a library directory named "`external`" in Android source code.

Since the libraries are managed in Git, we collect modified library versions for Android and official release versions of the original library project using tags in the repositories. In case of `junit`, 458 versions for Android releases and 20 official versions are extracted from the repositories. The files for Android releases include source files reused from official library versions and additional files to extend features. The analyzed files exclude the Android kernel and other libraries.

Our method extracted reuse pairs by comparing the JUnit versions for Android and the official versions of JUnit. Table 1 shows example values of $S(p, l)$ for a particular version of JUnit for an Android release. The aggregated file similarity values clearly show the differences between the JUnit version in Android with official versions. Using those values, we identify a reuse pair of Android 4.1.1 and JUnit 4.10.

Table 1 The aggregated file similarity between p = JUnit for `Android 4.1.1` and some official versions of JUnit.

JUnit version (l)	S(JUnit for `Android 4.1.1`, l)
`JUnit4-r3.8.2`	15.45
`JUnit4-r4.10`	160.87
`JUnit4-r4.11`	114.56
`JUnit4-r4.12`	69.65

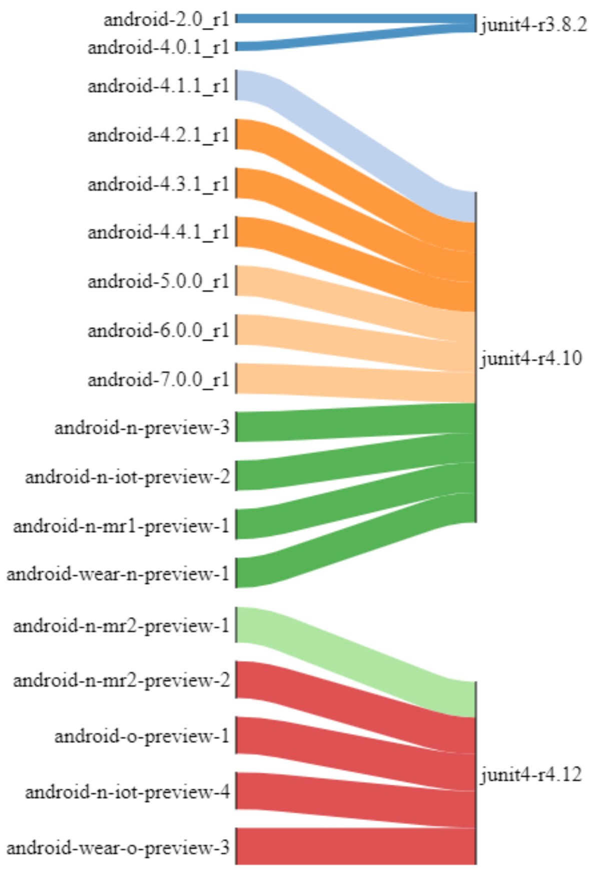

Fig. 1 A visualization result that shows how Android project uses JUnit versions

The analysis compares all version pairs in the repositories. Our single-threaded implementation takes 15 seconds using Intel Xeon E5-2690v3 CPU (2.60GHz) to analyze repositories on a SSD.

Figure 1 shows a result of how Android project uses JUnit versions. Due to the limited space, some minor versions without source code changes are omitted. The left side of the figure shows a list of Android versions sorted by timestamps in the repository. They are connected to JUnit versions on the right side. From the visualization, we could know which version of Android have modified or up-

Table 2 The aggregated file similarity values of version pairs in Fig. 1

Version of JUnit for Android (p)	JUnit version(l)	$S(p,l)$
`android-2.0_r1`	`JUnit4-r3.8.2`	48.00
`android-4.1.1_r1`	`JUnit4-r4.10`	160.87
`android-4.2.1_r1`	`JUnit4-r4.10`	159.84
`android-5.0.0_r1`	`JUnit4-r4.10`	159.87
`android-n-preview-3`	`JUnit4-r4.10`	159.83
`android-n-mr2-preview-1`	`JUnit4-r4.12`	185.80
`android-n-mr2-preview-2`	`JUnit4-r4.12`	191.96

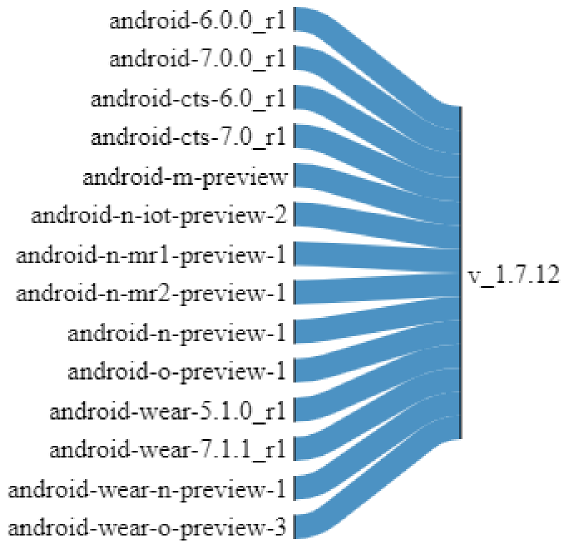

Fig. 2 A visualization result that shows how Android project uses slf4j versions

dated their own copy library. The visualization result shows that the first period of Android project used JUnit version 3.8.2. After the copy was updated to JUnit 4.10 and JUnit 4.12, the developers modified their own copy, as indicated by several colors of the links. Table 2 shows the actual values of aggregated file similarity between the visualized links. Android developers modified their copy of JUnit 4.10 for Android 4.2.1. The commit message says "Allow subclasses of `JUnit38ClassRunner` to create specialized filtered test suites." The developers introduced a new feature to their own copy and the change decreased the aggregated file similarity. Another change has been performed for Android 5.0.0. Developers introduced their own ("android test runner") package and reverted a modified file to the original version. The change increased the aggregated file similarity. The version has been updated again for Android-n-preview-3. Developers added generic type annotations to aid certain Java compilers. Android developers also used modified versions of JUnit 4.12. The change is related to Hamcrest library [14] that provides rich expressions for JUnit test cases. The first modified version was compiled against Hamcrest 1.1, and the second version was compiled against Hamcrest 2.0.

In case of `slf4j`, our implementation takes 38 seconds to compare 253 versions for Android releases with 58 official versions. Figure 2 shows that Android project uses slf4j version 1.7.12 without modification of source files. Since no vulnerabilities have been reported for slf4j 1.7.12, Android developers can keep the version as is. While slf4j project continues to enhance the library, Android developers use a fixed set of features. Similarly, in case of `easymock`, our visualization shows that Android project uses easymock 2.5.2 without modification. Although it is an old version released in 2009, it is a functional version. Our implementation takes 3 minutes and 8 seconds to compare 439 versions for Android releases and 13 official versions.

4. Summary and Future Work

In the case study, our method successfully extracted and visualized library reuse and update activity in a repository. The case study illustrates that the extracted diargams show us when developers adopt, update or change their cloned library in the version history of a product.

In the future work, we would like to optimize source code comparison to visualize library update activity of a large software project in a practical time. We are also interested in a visualization method to understand how reused source files are customized in a project.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Numbers JP25220003, JP26280021, JP16K16037, and JP17H00731.

References

[1] P. Mohagheghi, R. Conradi, O.M. Killi, and H. Schwarz, "An empirical study of software reuse vs. defect-density and stability," Proceedings of the 26th International Conference on Software Engineering, pp.282–291, 2004.
[2] C. Ebert, "Open source software in industry," IEEE Softw., vol.25, no.3, pp.52–53, 2008.
[3] J. Rubin, A. Kirshin, G. Botterweck, and M. Chechik, "Managing forked product variants," Proceedings of the 16th International Software Product Line Conference, pp.156–160, 2012.
[4] T. Mende, R. Koschke, and F. Beckwermert, "An evaluation of code similarity identification for the grow-and-prune model," Journal of Software Maintenance and Evolution, vol.21, no.2, pp.143–169, 2009.
[5] A. Ihara, D. Fujibayashi, i. Suwa, R.G. Kula, and K. Matsumoto, "Understanding when to adopt a library: A case study on asf projects," Proceedings of the International Conference on Open Source Systems, vol.496, pp.128–138, 2017.
[6] R.G. Kula, C. De Roover, D. German, T. Ishio, and K. Inoue, "Visualizing the evolution of systems and their library dependencies," Proceedings of the 2nd IEEE Working Conference on Software Visualization, pp.127–136, 2014.
[7] P. Xia, M. Matsushita, N. Yoshida, and K. Inoue, "Studying reuse of out-dated third-party code in open source projects," JSSST Computer Software, vol.30, no.4, pp.98–104, 2013.
[8] T. Ishio, Y. Sakaguchi, K. Ito, and K. Inoue, "Source file set search for clone-and-own reuse analysis," Proceedings of the 14th International Conference on Mining Software Repositories, MSR '17, Piscataway, NJ, USA, pp.257–268, IEEE Press, 2017.
[9] Sankey Diagrams, http://www.sankey-diagrams.com/ (accessed Sept. 13, 2017).

[10] Git Repositories on Android, https://android.googlesource.com (accessed Sept. 19th, 2017).
[11] JUnit, http://junit.org/junit4/ (accessed Oct. 24th, 2017).
[12] SLF4J, https://www.slf4j.org/ (accessed Oct. 24th, 2017).
[13] EasyMock, http://easymock.org/ (accessed Oct. 24th, 2017).
[14] Hamcrest, http://hamcrest.org/JavaHamcrest/ (accessed Oct. 24th, 2017).