Reordering Results of Keyword-based Code Search for Supporting Simultaneous Code Changes

Yusuke Sabi*, Hiroaki Murakami*, Yoshiki Higo* and Shinji Kusumoto*
*Graduate School of Information Science and Technology, Osaka University, Japan
Email: {y-sabi, h-murakm, higo, kusumoto}@ist.osaka-u.ac.jp

***Abstract*—Many research studies have been conducted to help simultaneous code changes on multiple code fragments. Code clones and logical couplings are often utilized in such research studies. However, most of them have been evaluated on only open source projects or students' software. In this paper, we report our academic-industrial collaboration with a software company. The collaboration is intended to suggest multiple code fragments to be changed simultaneously when a developer specifies a keyword such as variable names on source code. In the collaboration, we propose to use code clones and logical couplings information to reorder the code fragments. We confirmed that code clones and logical couplings worked well on helping simultaneous code changes on three projects that have being developed in the company.**

I. Introduction

Software developers often use keyword-based code search tools like grep to detect code fragments to be changed simultaneously. Their intentions to use such tools are collecting all the code fragments to be changed without missing any of them. In their situation, they prioritize recall over precision.

In a Japanese software company that collaborates with us, software developers use a keyword-based code search tool. However, developers tend to make mistakes in cases where many code fragments are detected by the search tool because they checks if each of detected code fragments has to be changed or not one by one.

Our academic-industrial collaboration is intended to help developers in such a situation. In the collaboration, we proposed to use code clones and logical couplings information to reorder code fragments detected by the search tool. We also extended the search tool based on the proposal. We have applied the extended version to three software projects in the company. As a result, we confirmed that reordering with code clones and logical couplings are useful.

II. Preliminaries

In this section, we explain code clones and logical couplings that are used in this research.

A. Code Clone

A code clone (in short, CC) is a code fragment that has similar/identical code fragments in source code. CCs are usually classified into three categories, Type-1, Type-2, and Type-3 [1]. In this research, we use our method-based clone detection technique [3], which detects Type-1 and Type-2 in a short time.

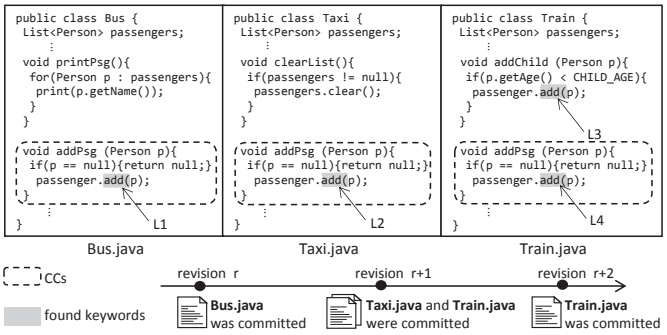


Fig. 1. Sample Source Code

B. Logical Coupling

Logical coupling (in short, LC) is a kind of coupling between program elements (usually, files or methods) that have been changed simultaneously in the past [2]. In this research, the unit of LC is a file due to ease of tool implementation. LCs between files can be easily identified by mining code repositories such as SVN.

III. Proposed Technique

We propose a technique to reorder results of keyword-based code search. The proposed technique uses two kinds of information, CC and LC, to reorder code fragments including a keyword specified by a developer.

Figure 1 shows sample source code to explain the proposed technique. Assume that we detect a bug in the argument of the *add* invocation in Taxi.java. If we want to check all the *add* invocations, the keyword-based code search would take "`add(`" as an input. Then, four "`add(`" in the source code are found. Herein, we label the four "`add(`" as `L1`, `L2`, `L3` and `L4`.

Many keyword-based code search tools including grep show the results in the dictionary order of names of files that contain the found keywords, such as `L1`→`L2`→`L3`→`L4`. In the case of using CCs, the proposed technique reorders the results as `L1`→`L2`→`L4`→`L3` because the three methods containing `L1`, `L2` and `L4` are CCs. In Fig. 1, Taxi.java and Train.java have been changed once simultaneously. Furthermore, Taxi.java and Bus.java have not been changed simultaneously. Taxi.java contains `L2`. Also, Train.java contains `L3` and `L4`. Thus, in the case of using LCs, the proposed technique reorders the results as `L2`→`L3`→`L4`→`L1`.

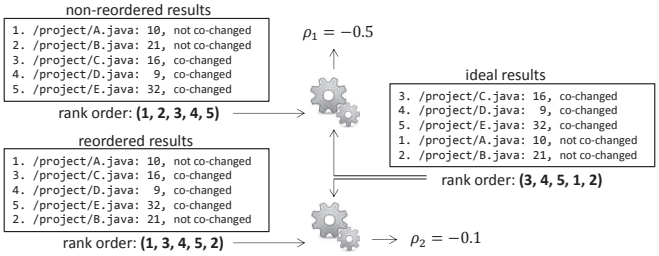


Fig. 2. Examples of Calculating Correlation Coefficients

IV. Experiment

In order to evaluate the usefulness of the proposed technique, we applied the extended version of the tool to three projects in the company. In this application, the company provided us with the source code repositories of the projects and the past results of keyword-based code searches (we call this "*non-reordered results*"). The results included keywords specified for code search, dates of code search, locations (filenames and line numbers) of the specified keywords, and "co-changed" or "not co-changed" for each of the locations. The projects were developed with Java, Javascript, XML and HTML. Table I shows details of the target projects. The reason why the numbers of keyword searches are different between LC and CC is that CC is used for only Java files although LC is used for all the files. In this experiment, we set *minimum token length* as 30 for the configuration of the CC detection.

We conducted the experiment with following steps.

Step-1: We obtained *reordered results* of keyword-based code search by replicating code search with the proposed technique and snapshots of the target projects for each of the past search results.

Step-2: We made *ideal results* by shifting all the "co-changed" keywords to the tops.

Step-3: We calculated correlation coefficients between *non-reordered results* and *ideal results*. We also calculated correlation coefficients between *reordered results* and *ideal results*.

We used Spearman's rank-correlation coefficient ρ as correlation coefficient. Figure 2 shows examples of calculating correlation coefficients. In Fig. 2, the rank order of *non-reordered results* is (1, 2, 3, 4, 5), that of *ideal results* is (3, 4, 5, 1, 2) and that of *reordered results* is (1, 3, 4, 5, 2). In this case, ρ_1 between *non-reordered results* and *ideal results* and ρ_2 between *reordered results* and *ideal results* are calculated as follows.

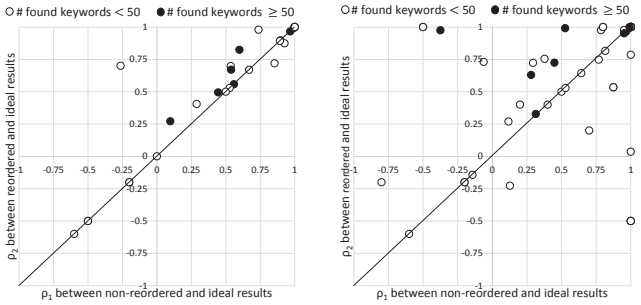
$$\rho_1 = 1 - \frac{6*((1-3)^2+(2-4)^2+(3-5)^2+(4-1)^2+(5-2)^2)}{5^3-5} = -0.5 \quad (1)$$

$$\rho_2 = 1 - \frac{6*((1-3)^2+(3-4)^2+(4-5)^2+(5-1)^2+(2-2)^2)}{5^3-5} = -0.1 \quad (2)$$

If ρ_2 is higher than ρ_1, we can say that reordering get close to the *ideal results*.

TABLE I
Target Projects

Name	# commits	# developers	# keyword search	
			CC	LC
Software A	26,000	45	11	18
Software B	700	13	11	28
Software C	250	9	6	11



(a) Results for CC (b) Results for LC

Fig. 3. Results of Correlation Coefficients

Figure 3(a) and 3(b) shows results of correlation coefficients. In each figure, the horizontal axis represents ρ_1 and the vertical axis represents ρ_2. In Fig. 3(a), 8 out of 28 improved, 2 out of 28 got worse and 18 out of 28 were unchanged in the case of using CCs. In Fig. 3(b), 19 out of 57 improved, 11 out of 57 got worse and 27 out of 57 were unchanged in the case of using LCs. For the cases where the number of found keywords were 50 or more, in Fig. 3(a), 4 out of 7 improved and 3 out of 7 were unchanged. Moreover, in Fig. 3(b), 7 out of 9 improved and 2 out of 9 were unchanged. Those facts indicated that reorderings by the proposed technique did not make the results of keyword-based code searches worse. Thus, we consider that reorderings by the proposed technique worked well especially when many keywords are found by the keyword-based code search.

V. Conclusion

In this paper, we reported our academic-industrial collaboration with a Japanese software company. In this collaboration, we intended to support developers to get code fragments to be changed simultaneously. We extended the search tool that has been used in the company with code clones and logical couplings information to suggest code fragments in a more useful order. We applied the extended version to three software projects in the company. As a result, the extended version was able to improve the order of code fragments in many cases. In the future, the developers in the company will use the extended version of the search tool in practice and we will receive feedbacks from industrial users.

Acknowledgment

We would like to thank Hirokazu Matsuoka and Hiromi Toyokawa for the support to this work. This work was supported by JSPS KAKENHI Grant Numbers 25220003, 24650011, and 24680002.

References

[1] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, "Comparison and Evaluation of Clone Detection Tools," *IEEE TSE*, vol. 33, no. 9, pp. 577–591, 2007.

[2] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," in *ICSM*, 2008, pp. 190–198.

[3] T. Ishihara, K. Hotta, Y. Higo, H. Igaki, and S. Kusumoto, "Inter-Project Functional Clone Detection Toward Building Libraries — An Empirical Study on 13,000 Projects —," in *WCRE*, 2012, pp. 387–391.