社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

Finding Code Clones for Refactoring with Clone Metrics : A Case Study of Open Source Software

Eunjong CHOI†, Norihiro YOSHIDA††, Takashi ISHIO†, Katsuro INOUE†, and Tateki SANO†††

† Graduate School of Information Science and Technology, Osaka University
1–5 Yamadaoka, Suita-shi, Osaka, 565–0871 Japan
†† Graduate School of Information Science, Nara Institute of Science and Technology
8916–5 Takayama-cho, Ikoma-shi, Nara, 630–0192 Japan
††† Software Process Innovation and Standardization Division, NEC Corporation
5–7–1 Shiba, Minato-ku, Tokyo, 108–8001 Japan
E-mail: †{ejchoi,ishio,inoue}@ist.osaka-u.ac.jp, ††yoshida@is.naist.jp, †††t-sano@cp.jp.nec.com

Abstract A code clone is a code fragment that has identical or similar code fragments to it in the source code. Code clone has been regarded as one of the factors that makes software maintenance more difficult. Therefore, to refactor code clones into one method is promising way to reduce maintenance cost in the future. In our previous study, we proposed a method to extract code clones for refactoring using clone metrics. We had conducted an empirical study on Java application developed by NEC Corporation. It turned out that our proposed method is effective to extract refactoring candidate code clones. In this paper, we show the result of applying our proposed method into open source software systems.

Key words code clone, refactoring, open source software

1. Introduction

Code clone is similar or identical code fragments in source code. The presence of code clones has been regarded as indication of low maintainability of software because if a bug is found in a code clone, the other code clone have to be checked for defect detection.

Refactoring [5] is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure. Refactoring code clones (e.g., merge code clones into a single method in Java program) is an effective way to reduce code clones in a software system.

However, all code clones detected by a code clone detection tool are not appropriate for refactoring. For example, language-dependent code clones [6] (i.e. code clones that indispensably exist in a source code due to the specifications of used program language) are clearly inappropriate for refactoring. Although, numerous techniques and tools have been proposed for code clone detection [8], [11], only little has been known about which detected code clones are appropriate for refactoring and how to extract code clones for refactoring.

Our previous study [4] proposed method to extract code clones for refactoring using clone metrics. We showed the usefulness of our proposed method with a survey to a developer in NEC Corporation. According to the feedback, it turned out that our proposed method using combined clone metrics is effective method to extract code clones for refactoring.

However, due to the time limitation, we conducted on a single system in previous study. Therefore, although we could validate our method in previous study, our method may not generalize to other software systems.

In this study, we apply our proposed method to open source software and discuss its results. The rest of this paper is organized as follows: We first expalin the background of our study in Section 2 and then describe a case study and its results in Section 3. Section 4 discusses threats to validity. Section 5 surveys related work, and Section 6 concludes our paper.

2. Background

In this section, we explain about clone metrics and our previous method based on them.

2. 1 Clone Metrics

Our research group have developed and proposed a token-

based code clone detection tool CCFinder [11] and a code clone analysis environment Gemini [13] which visualizes the code clone information from CCFinder. Gemini supports clone metrics LEN(S), POP(S) and RNR(S) [6], [13]. Each of them characterize a clone set (i.e. an equivalent of code clones) S:

- LEN(S) − The average number of token sequence of code clones in a clone set S.
- POP(S) − The number of code clones in a clone set S.
- RNR(S) − The ratio of non-repeated token sequence of code clone in a clone set S.

The definition of RNR(S) metric is described in Equation (1). If clone set S includes n code clones, $c_1, c_2 \ldots, c_n$, $LOS_{whole}(f_i)$ is the Length Of the whole token Sequence of code clone c_i. $LOS_{non-repeated}(f_i)$ is the Length Of non-repeated token Sequence of code clone c_i, then,

$$RNR(S) = \frac{\sum_{i=1}^{n} LOS_{non-repeated}(c_i)}{\sum_{i=1}^{n} LOS_{whole}(c_i)} \times 100 \quad (1)$$

2.2 Features of Clone Sets Whose Each Clone Metric is Higher

Gemini characterizes code clones using clone metrics. Our research group has analyzed clone sets in numerous software systems for several proposes (e.g., refactoring, defect-check) using Gemini. The followings are the characteristics of clone sets extracted by each clone metric [6].

- Clone sets whose LEN(S) is higher than others − Clone sets whose LEN(S) is higher than other clone sets means that each code clone in a clone set S consists of longer token sequences than others clone sets. According to our observation, many of them include many consecutive if (or if-else) blocks that involve similar but different conditional expressions.
- Clone sets whose RNR(S) is higher than others − Clone sets whose RNR(S) is higher than other clone sets means that each code clone in a clone set S consists of more non-repeated code. According to our observation, because clone metric RNR(S) distinguishes only "non-repeated" token sequences from "repeated" token sequences of code clones, many of them do not organize a single semantic unit (i.e., many instructions forming a single functionality)
- Clone sets whose POP(S) is higher than others − Clone sets whose POP(S) is higher than other clone sets means that code clones in a clone set S appear more frequently in the system. According to our observation, those clone sets include many small size code clones. Moreover, these clone sets include a lot of language-dependent code clones.

2.3 Combinations of Clone Metrics

Our research group have suggested code clones for refactoring to industrial software developers.

However, according to their opinion, many of code clones that are extracted using just high single clone metric are inappropriate for refactoring due to the weakness described in Section 2.2. In order to improve the weakness of single-metric-based extraction, we proposed a method based on "combination of clone metrics". The followings are the details about the conducted case study in previous study for validating the our proposed method that using combined clone metrics.

2.3.1 Target Clone Sets

The target of the case study was a Java software developed by NEC corporation. It is 110KLOC across 296 files. From 736 clone sets that are detected by CCFinder, we selected 62 clone sets using a single clone metric value, and combined clone metrics values. We used 30 tokens as the minimum length of token sequence of a code clone to CCFinder.

The following are the details of subject clone sets:

- S_{LEN} − Clone sets whose LEN(S) value are top 10 high.
- S_{RNR} − Clone sets whose RNR(S) value are top 10 high.
- S_{POP} − Clone sets whose POP(S) value are top 10 high.
- $S_{LEN \cdot RNR}$ − 15 clone sets whose LEN(S) and RNR(S) values are high rank in the top 15.
- $S_{LEN \cdot POP}$ − 7 clone sets whose LEN(S) and POP(S) values are high rank in the top 15.
- $S_{RNR \cdot POP}$ − 18 clone sets whose RNR(S) and POP(S) values are high rank in the top 15.
- $S_{LEN \cdot RNR \cdot POP}$ − 1 clone set whose LEN(S), RNR(S) and POP(S) values are high rank in the top 15.

2.3.2 Results

We conducted a case study according to the following steps:

(1) Selected clone sets (details about these clone sets are described in Section 2.3.1) from CCFinder's output using clone metrics in Gemini.

(2) Conduct a survey about these clone sets and got feedback from a developer. The developer is a project manager with 10 years of development experiences with Java.

(3) Analyze the result of conducted survey.

We use *Precision* to analyses results of the survey. It can be used to investigate the question "How many clone sets were accepted as refactorable clone sets by a developer?" Equation (2) describes Precision. Let S_{All} represent all clone

Table 1 Precisions in the survey conducted in previous study

Clone Sets	#Selected Clone Sets	#Refactoring	Precision
S_{LEN}	10	7	**0.70**
S_{RNR}	10	4	0.40
S_{POP}	10	3	0.30
$S_{LEN \cdot RNR}$	15	13	**0.87**
$S_{LEN \cdot POP}$	7	6	**0.86**
$S_{RNR \cdot POP}$	18	14	**0.78**
$S_{LEN \cdot RNR \cdot POP}$	1	1	**1.00**

Table 2 Results of clone sets detected by a single clone metric, and combined metrics in the survey

Filtering Method	#Selected Clone Sets	#Refactoring	Precision
Single Clone Metric	30	14	0.47
Combined Clone Metrics	41	34	**0.83**

sets that are selected by each method, S_{ARC} represents clone sets accepted as refactorable clone sets by a developer.

$$Precision = \frac{|S_{ARC}|}{|S_{All}|} \quad (2)$$

Tables 1 and 2 describe the result of the survey. In Tables 1 and 2, column `Clone Sets` shows name of target clone sets. `#Selected Clone Sets` and `#Refactoring` show the number of selected clones from CCFinder's output and the number of clone sets that selected as refactorable clone sets in a survey respectively.

As shown in the column "Precision" in Table 1, precisions of S_{RNR}, S_{POP} are smaller than 0.50. This means that those clone sets were accepted by a developer as inappropriate clone sets for refactoring.

However, precisions of S_{LEN}, $S_{LEN \cdot RNR}$, $S_{LEN \cdot POP}$, $S_{RNR \cdot POP}$, and $S_{LEN \cdot RNR \cdot POP}$ are more than 0.70. This means that they were accepted by a developer as appropriate clone sets for refactoring. It is clear that all clone sets that are extracted using combined clone metrics values ($S_{LEN \cdot RNR}$, $S_{LEN \cdot POP}$, $S_{RNR \cdot POP}$, and $S_{LEN \cdot RNR \cdot POP}$) are more than 0.70 (They are shown in bold in column "Precision" in Table 1) are accepted as refactorable clone sets.

As shown in Table 2, Precision of clone sets that are extracted using combined clone metrics values are higher than clone sets than using each single clone metric. Therefore, we found code clones that extracted using combined clone metrics is more frequently accepted as refactorable clone sets than using each single clone metric.

3. Case Study of Open Source Software

3.1 Target Systems

In this study, we use two open source Java projects: Apache Ant [2] and JBoss [1], as our target systems. We use code clone detection tool, CCFinder and set 30 tokens as the minimum token length of a code clone because of its use and setting in previous study. The followings are the details of each subject system.

- Apache Ant − It is 198KLOC across 778 files. we selected 87 clone sets from 998 clone sets detected by CCFinder.
- JBoss − It is 633KLOC across 3,344 files. we selected 299 clone sets from 730 clone sets detected by CCFinder.

3.2 Result

Table 3 shows the precisions of clone sets in each software system. The precisions are little different from the previous study. The precisions of both S_{RNR}, both $S_{LEN \cdot RNR}$, $S_{LEN \cdot POP}$ in Apache Ant, $S_{RNR \cdot POP}$, and $S_{LEN \cdot RNR \cdot POP}$ in JBoss are more than 0.50 (They are shown in bold in column "Precision" in Table 3). The followings are our analysis on the results.

- S_{LEN} (The precisions are 0.00 and 0.20 in Apache Ant and JBoss respectively) − Code clones in those clone sets consist many consecutive if (or if-else) blocks. Consecutive if (or if-else) blocks are difficult to perform refactoring, generally. However, in previous study, just 2 types of parameters are included in consecutive if (or if-else) blocks. Therefore, a developer selected many clone sets whose LEN(S) value are top 10 high as refactorable clone sets. On the contrary, due to various types of parameters are included in consecutive if (or if-else) blocks, it is inappropriate for refactoring of clone sets in both Apache Ant and JBoss.
- S_{RNR} (The precisions are 0.70 and 0.80 in Apache Ant and JBoss respectively) − The size of many code clones in those clone sets are short. Several clone sets in Apache Ant and JBoss include insufficient scale code clones to organize semantic units..
- S_{POP} (The precisions are 0.00 and 0.00 in Apache Ant and JBoss respectively) − Code clones in clone sets whose POP(S) are higher than other clone sets appear more frequently in the system. Many of them in both Apache

Table 3 Results of each target system

Clone Sets	Target System	#Selected Clone Sets	#Refactoring	Precision
S_{LEN}	Apache Ant	10	0	0.00
	JBoss	10	2	0.20
S_{RNR}	Apache Ant	10	7	**0.70**
	JBoss	10	8	**0.80**
S_{POP}	Apache Ant	10	0	0.00
	JBoss	10	0	0.00
$S_{LEN \cdot RNR}$	Apache Ant	8	6	**0.75**
	JBoss	63	37	**0.59**
$S_{LEN \cdot POP}$	Apache Ant	18	10	**0.56**
	JBoss	104	3	0.03
$S_{RNR \cdot POP}$	Apache Ant	34	16	0.47
	JBoss	129	32	0.25
$S_{LEN \cdot RNR \cdot POP}$	Apache Ant	-	-	-
	JBoss	2	2	**1.00**

Ant and JBoss appear frequently in the system because they are language-dependent code clones and language-dependent code clones are not appropriate refactoring.

- $S_{LEN \cdot RNR}$ (The precisions are 0.75 and 0.59 in Apache Ant and JBoss respectively) − Due to various parameter types between code clones in clone sets in both Apache Ant and JBoss, several of them could not be merged into a single program unit.
- $S_{LEN \cdot POP}$ (The precisions are 0.56 and 0.03 in Apache Ant and JBoss respectively) − Several of thoses clone sets include many language-dependent code clones in both Apache Ant and JBoss. Therefore, many of them are inappropriate for refactoring.
- $S_{RNR \cdot POP}$ (The precisions are 0.47 and 0.25 in Apache Ant and JBoss respectively) − Several of thoses clone sets do not consist of a single functionality in both Apache Ant and Jboss. Even though they include many instructions, instructions do not consist of a single functionality.
- $S_{LEN \cdot RNR \cdot POP}$ (The precisions are 0.00 and 1.00 in Apache Ant and JBoss respectively) − Those clone sets are appropriate for refactoring compared to the other combination of clone metrics.

4. Threats to Validity

Due to the limited time and effort, we only evaluate precision of our method. We believe that precision is a good enough criterion for validating our method.

Moreover, because we use only 3 clone metrics in our studies, it may not good enough to reveal the all characteristic of appropriate code clones for refactoring. We are planning to use other clone metrics and source code metrics (e.g., complexity metrics).

Finally, because our case study is conducted on Java software system, our method may not work on other language software(e.g., C/C++, Python). In order to improve the generality of our research, we need to investigate the usefulness of our method for source code written in without Java.

5. Related Work

Jiang *et al.* [9] and Kapser *et al.* [12] pointed out that code clone detection tools using parameterized matching detected a lot of false positives. Jiang *et al.* [9] used textual filtering techniques to remove false positives from CCFinder's output. They removed code clones whose textual similarity falls below a certain threshold. Kapser *et al.* [12] proposed the following techniques to remove false positives from the output of their token-based clone detection tool.

- Identifier names outside functions (e.g., Java methods, C functions) are not parameterized. For example, declarations of field variables in Java programs and external variables in C programs are often duplicated but most of them are false positives.
- Simple method calls are matched only if Levenshtein Disitance of those method names is small.
- Logical structures (e.g., switch statements, `if` (or `if-else`) blocks) are matched if 50% of tokens in these structures are identical.

There is the possibility to make our method more effective by applying the filtering techniques proposed by Jiang *et al.* and Kapser *et al.* as the preprocessor or the postprocessor of our method.

CCFinderX [10] developed by Kamiya provides the metric TKS(S) that means the number of token types in code clones belonging clone set S. The metric TKS is effective to remove clone sets not in need of developer's investigation (e.g., consecutive variable declarations) because those clones tend to have small numbers of token types. This means that there is the possibility of improving the effectiveness of our method

by use of the metric TKS in addition to the use of the metric RNR. However, clone sets with low RNR value include a lot of consecutive parts. They tend to have small number of token types and low TKS value. This correlation means that the use of both TKS and RNR is not significantly effective.

Balazinska [3] *et al.* and Higo [7] *et al.* characterize clone sets according to the ease of refactoring. Balazinska *et al.* characterize clone sets by analyzing the following information:

- Differences among the code clones belonging to a clone set
- Dependencies between the code clone belonging to a clone set and its surrounding code

Higo [7] *et al.* proposed two metrics to represent the average number of the externally defined variables respectively referenced and assigned in the code clones belonging to a clone set. The combination of our method and the techniques focusing on the ease of refactoring has the possibility to improve the effectiveness of clone set filtering.

6. Summary and Future Work

In this paper, we conducted a case study of open source software systems and discuss its result. We found that that reasons why several clone sets are inappropriate for refactoring.

As future work, we are planning to perform case studies of software systems written without using Java. Moreover, we would like to investigate recall and more metrics.

Acknowledgments

We express our great thanks to Ms. Fusako Mitsuhashi and Mr. Shin 'ichi Iwasaki of NEC Corporation for data collection. This work is being conducted as a part of Stage Project. Also, this work is partially supported by JSPS, Grant-in-Aid for Scientic Research (A) (21240002) and Grant-in-Aid for Research Activity start-up(22800040).

References

[1] JBoss Application Server. `http://www.jboss.org`.

[2] The Apache Ant Project. `http://ant.apache.org/`.

[3] M. Balazinska, E. Merlo, M. Dagenais, B. Lagüe, and K. Kontogiannis. Advanced clone-analysis to support object-oriented system refactoring. In *Proc. of WCRE 2000*, pages 98–107, 2000.

[4] E. Choi, N. Yoshida, T. Ishio, K. Inoue, and T. Sano. Extracting code clones for refactoring using combinations of clone metrics. In *Proc. of the IWSC 2011*, pages 7–13, 2011.

[5] M. Fowler. *Refactoring: improving the design of existing code.* Addison Wesley, 1999.

[6] Y. Higo, T. Kamiya, S. Kusumoto, and K. Inoue. Method and Implementation for Investigating Code Clones in a Software System. *Information and Software Technology*, 49(9-10):985–998, 2007.

[7] Y. Higo, S. Kusumoto, and K. Inoue. A metric-based approach to identifying refactoring opportunities for merging code clones in a Java software system. *J. Softw. Maint. Evol.: Res. Pract.*, 20:435–461, 2008.

[8] L. Jiang, G. Misherghi, Z. Su, and S. Glondu. DECKARD: Scalable and accurate tree-based detection of code clones. In *Proc. of ICSE 2007*, pages 96–105, 2007.

[9] Z. M. Jiang and A. E. Hassan. A framework for studying clones in large software systems. In *Proc. of SCAM 2007*, pages 203–212, 2007.

[10] T. Kaimiya. Tutorial of CLI Tool ccfx, 2008. `http://www.ccfinder.net/doc/10.2/en/tutorial-ccfx.html`.

[11] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. *IEEE Trans. Softw. Eng.*, 28(7):654–670, 2002.

[12] C. J. Kapser and M. W. Godfrey. "Cloning considered harmful" considered harmful: patterns of cloning in software. *Empir Software Eng*, 13:645–692, 2008.

[13] Y. Ueda, T. Kamiya, S. Kusumoto, and K. Inoue. Gemini: Maintenance support environment based on code clone analysis. In *Proc. of METRICS 2002*, pages 67–76, 2002.