Extracting Code Clones for Refactoring Using Combinations of Clone Metrics

Eunjong Choi[1], Norihiro Yoshida[2], Takashi Ishio[1], Katsuro Inoue[1], Tateki Sano[3]
[1]Graduate School of Information Science and Technology, Osaka University, Japan
{ejchoi, ishio, inoue}@ist.osaka-u.ac.jp
[2]Graduate School of Information Science, Nara Institute of Science and Technology, Japan
yoshida@is.naist.jp
[3]Software Process Innovation and Standardization Division, NEC Corporation, Japan
t-sano@cp.jp.nec.com

ABSTRACT

Code clone detection tools may report a large number of code clones, while software developers are interested in only a subset of code clones that are relevant to software development tasks such as refactoring. Our research group has supported many software developers with the code clone detection tool CCFinder and its GUI front-end Gemini. Gemini shows clone sets (i.e., a set of code clones identical or similar to each other) with several clone metrics including their length and the number of code clones; however, it is not clear how to use those metrics to extract interesting code clones for developers. In this paper, we propose a method combining clone metrics to extract code clones for refactoring activity. We have conducted an empirical study on a web application developed by a Japanese software company. The result indicates that combinations of simple clone metric is more effective to extract refactoring candidates in detected code clones than individual clone metric.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Restructuring, reverse engineering, and reengineering*; D.2.8 [**Software Engineering**]: Metrics—*Product metrics*

General Terms

Experimentation

Keywords

Code clone, Refactoring, Industrial case study

IWSC'11, May 23, 2011, Waikiki, Honolulu, HI, USA

1. INTRODUCTION

Many code clone detection tools[5, 6, 8] have been proposed to capture various aspects of source code similarity[11]. A code clone detection tool generally finds all source code clones that match its own definition of code clone; therefore, a tool may report a large number of code clones for large scale software. On the other hand, software developers are interested in only the subset of code clones that are relevant to their activities.

Refactoring[2] is one promising activity to improve the maintainability of code clones. Code clone is not always appropriate for refactoring because developers sometimes have to repeatedly write code clones that cannot be merged due to language limitations[4, 9, 10]. However, a *clone set* (i.e., a set of code clones identical or similar to each other) indicates considerable opportunities for developers to merge code clones into one or a few program units(e.g., Java methods) by refactoring[4, 9, 10].

In this research, we focus on code clones that should be checked for refactoring before a software system is released. Although code clones are not always appropriate for refactoring, developers would like to find and modularize common functionalities in an important system because such a system will be maintained in next 10 years as a part of the infrastructure of a company. Due to the limited development time and cost, it is not acceptable for developers to manually check all the code clones detected by a tool. We propose a filtering approach to extracting code clones for refactoring from a large amount of code clones.

Our research group has developed a code clone analysis tool named Gemini[3, 12]. Gemini is a GUI front-end for CCFinder[8]; it takes as input a list of code clones generated by CCFinder and shows developers quantitative information on clone sets by clone metrics. Its clone metrics include *LEN* (the average LENgth of token sequences of code clones in a clone set[3]), *POP* (the POPulation of code clones in a clone set), and *RNR* (the Ratio of Non-Repeated token sequences of code clones in a clone set). Using these metrics, developers can extract a subset of code clones for their purpose. However, little is known about how using these metrics when developers extract interesting code clones for refactoring activity.

We have analyzed many industrial software and received feedback from the developers. From these experiences, we recognized that clone sets extracted by a combination of

multiple clone metrics are more appropriate for refactoring than clone sets extracted by extremely higher values of individual clone metric. However, we have not evaluated what makes them relevant for refactoring. In this study, we evaluate clone sets extracted by a combination of clone metrics. The contributions of this study are as follows.

- Using clone metrics, we propose a method to extract refactoring candidate code clones from CCFinder's output.
- We confirm the effectiveness of our method with a survey in the form of a questionnaire for a software developer in a company.

The rest of paper is organized as follows: Section 2 provides a brief description of Gemini. In Section 3, we describe our proposed method. In Section 4, we describe a case study and its results, and discuss threats to validity. In Section 5, we present some related work. Section 6 concludes our paper with final remarks and indications about our future work.

2. BACKGROUND

2.1 Code Clone Visualization Tool: Gemini

We have developed a GUI front-end for CCFinder[8], Gemini[3, 12]. CCFinder and Gemini are distributed to hundreds of Japanese and international companies and universities. Also, we have often analyzed industrial code clones with Gemini through industry-university collaborations.

CCFinder detects a large number of code clones in large scale systems. To support code clone analysis, Gemini has several functionalities to extract clone sets from CCFinder's output. One major functionality of Gemini is the Metric Graph. The Metric Graph visualizes the clone metrics and enables the user to select a part of the clone sets using clone metrics. For example, POP(S) is a clone metric representing the number of code clones in clone set S. If a user is interested in clone sets that appear in more than 5 locations, he or she can select the clone sets using a filter "POP(S) $\geq$ 5".

Gemini supports clone metrics including LEN(S), RNR(S) and POP(S).

LEN(S)[3, 12] : LEN(S) is the average length of token sequences of code clones in clone set S. Higher LEN(S) values mean that each code clone in a clone set S consists of more token sequences. Contrary to this, lower LEN(S) values mean that each code clone in a clone set S consists of less token sequences, and the size of code fragments are smaller.

RNR(S)[3] : RNR(S) is the ratio of non-repeated token sequences of code clones in clone set S.
We assume that the clone set S includes n code clones, $c_1, c_2 \ldots, c_n$, $LOS_{whole}(f_i)$ represents the Length Of the whole token Sequence of code clone c_i, and $LOS_{repeated}(f_i)$ represents the Length Of repeated token Sequence of code clone c_i, then,

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

Here, we give an example of the compute RNR(S) value. We assume that we detect code clones from three source files(F_1, F_2, F_3). Each source file consists of following five tokens. (Superscript * indicated that the token is in a repeated token sequence.)

$$F_1 : a\ b\ c\ a\ b$$
$$F_2 : c\ c^*\ c^*\ a\ b$$
$$F_3 : c\ c^*\ a\ b\ c$$

Here, we assume that at least two tokens are needed to be identified as a code clone. With this assumption, the following two clone sets are detected from the source files. We use label *C(F_i, j, k)* to represent a fragment. Fragment label *C(F_i, j, k)* starts at the jth token and ends at the kth token in source file F_i (j must be less than k).)

$$S_1 : C(F_1, 1, 2), C(F_1, 4, 5), C(F_2, 4, 5), C(F_3, 3, 4)$$
$$S_2 : C(F_2, 1, 2), C(F_2, 2, 3), C(F_3, 1, 2)$$

In this case, RNR(S) values are computed as following:

$$RNR(S_1) = (1 - \frac{0+0+0+0}{2+2+2+2}) \times 100 = 100$$
$$RNR(S_2) = (1 - \frac{1+2+1}{2+2+2}) \times 100 = 30$$

Here, we exaplain the clone sets whose RNR is higher, and lower with Figure 1. [1]

Higher RNR(S) values mean that each code clone in a clone set S consists of more non-repeated token sequences(Figure 1(a)). Contrary to this, lower RNR(S) values means that each code clone in a clone set S consists of more repeated token sequences(Figure 1(b)). In most cases, repeated code sequences are involved in language-dependent code clones (e.g., code clones that involve consecutive if (or if-else) blocks, case entries of switch statements, consecutive variable declarations).

POP(S)[3, 12] : POP(S) is the number of code clones in a clone set S. Higher POP(S) values mean that code clones in a clone set appear more frequently in the system. Contrary to this, lower POP(S) values mean that code clones in a clone set appear in fewer places is the system.

The Metric Graph in Gemini is designed to enable the users to quantitatively select clone sets. Here, we explain the Metric Graph using Figure 2. In the Metric Graph, each clone metric has a parallel coordinate axis. The clone set is shown by a polygonal line connecting clone metric values. Two lines for clone sets S_1 and S_2 show that LEN(S_1) and RNR(S_1) are higher than LEN(S_2) and RNR(S_2), respectively, while POP(S_1) is lower than POP(S_2).

Users can specify the upper and lower limits of each clone metric. The grey area shows the range bounded by clone metrics upper and lower limits. In Figure 2(a), all the clone

[1]The examples in Figure 1 are picked from Ant 1.7.0 since the the subject source code used in Section 4. The source code clones are similar to these examples.

```
if (assumeJava11()) {
        Path cp = new Path(project);

        Path bp = getBootClassPath();
        if (bp.size() > 0) {
           cp.append(bp);
        }

        if (extdirs != null) {
           cp.addExtdirs(extdirs);
        }
        cp.append(classpath);
        cp.append(sourcepath);
```

(a) Code fragment in a clone set whose RNR(S) is 97, the highest RNR(S) value in Ant 1.7.0.

```
printProperty(out, ProxySetup.HTTP_PROXY_PORT);
printProperty(out, ProxySetup.HTTP_PROXY_USERNAME);
printProperty(out, ProxySetup.HTTP_PROXY_PASSWORD);
printProperty(out, ProxySetup.HTTP_NON_PROXY_HOSTS);
printProperty(out, ProxySetup.HTTPS_PROXY_HOST);
printProperty(out, ProxySetup.HTTPS_PROXY_PORT);
printProperty(out, ProxySetup.HTTPS_NON_PROXY_HOSTS);
printProperty(out, ProxySetup.FTP_PROXY_HOST);
```

(b) Code fragment in a clone set whose RNR(S) is 2, the lowest RNR(S) value in Ant 1.7.0.

Figure 1: Example of code fragments in clone set whose RNR(S) is the highest, and lowest

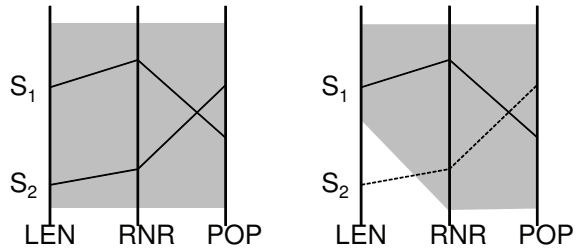
metric values are included in the grey part. As such, all clone sets are selected. In Figure 2(b), the values of LEN(S_2) is smaller than the lower limit of LEN, S_2 is not selected. This means that we can get "long" code clones by changing the lower limit of LEN. Thus the Metric Graph enables users to choose arbitrary clone sets based on clone metric values with AND operators.

The Metric Graph of Gemini provides clone sets which satisfy clone metrics values in the range specified by developers. However, research has not been done on the appropriate combinations of clone metric values range for effective clone sets filtering. Consequently, it is difficult for developers to give Gemini appropriate clone metric-value ranges for effective filtering.

2.2 Filtering Clone Sets Using Individual Clone Metric

Using Gemini, we have suggested to industrial software developers code clones that should be checked for refactoring and received significant feedback. According to their opinion, many clone sets whose individual clone metric value is high are inappropriate for refactoring. Based on the feedback, we have analyzed the clone sets whose individual clone metric value is high. Here is features of clone sets whose individual clone metric value is high that we observed and the obstacles that we recognized to perform refactoring with Figure 3 [2].

[2]The examples in Figure 3 are picked from Ant 1.7.0 since the the subject source code used in Section 4. The source code clones are similar to these examples.



(a) before (b) after

Figure 2: Filtering clone sets using the Metric Graph

Clone sets whose LEN(S) is higher than others
As described in Section 2.1, clone sets whose LEN(S) is higher than other clone sets(Figure 3(a)) means that each code clone in a clone set S consists of more token sequences. However, in our observation, these clone sets include many consecutive if (or if-else) blocks that involve similar but different conditional expressions. Therefore, it takes effort to perform refactoring these code clones.

Clone sets whose RNR(S) is higher than others
As described in Section 2.1, clone sets whose RNR(S) is higher than other clone sets(Figure 3(b)) means that each code clone in a clone set S consists of more non-repeated code sequences. However, in our observation, clone metric RNR(S) distinguishes only "non-repeated" token sequences from "repeated" token sequences of code clones. The clone sets whose RNR(R) is higher have the tendency of including a procedure corresponding to a semantic unit. (e.g., just only a few lines of code clones, if (or if-else) blocks with a few lines)

Clone sets whose POP(S) is higher than others
As described in Section 2.1, clone sets whose POP(S) is higher than other clone sets(Figure 3(c)) means that code clones in a clone set S appear more frequently in the system. However, in our observation, these clone sets include many small size code clones. It takes more effort to perform refactoring the smaller code clones than longer code clones. Moreover, these clone sets include a lot of language-dependent code clones. Therefore, it is inefficient to perform refactoring theses clone sets.

Therefore, we also recognized the following characteristic.

- It was observed that if a clone set whose one clone metric value is high, the other clone metrics values are low. Therefore, we might conjecture that using combined clone metrics can be a feasible method to get refactoring candidate code clones than using individual clone metric .
- Combining clone metrics might cover defects of clone sets whose individual clone metric value is high.

```
if ((p = getProject().getProperty("ant.netrexxc.binary")) != null) {
   this.binary = Project.toBoolean(p);
}
// classpath makes no sense
if ((p = getProject().getProperty("ant.netrexxc.comments")) != null) {
   this.comments = Project.toBoolean(p);
}
if ((p = getProject().getProperty("ant.netrexxc.compact")) != null) {
   this.compact = Project.toBoolean(p);
}
                    The rest is omitted
                            .
                            .
```

(a) Code fragment in a clone set whose LEN(S) is 455, the highest LEN(S) value in Ant 1.7.0.

```
}
    } else {
      // is the zip file in the cache
      ZipFile zipFile = (ZipFile) zipFiles.get(file);
      if (zipFile == null) {
        zipFile = new ZipFile(file);
        zipFiles.put(file, zipFile);
      }
      ZipEntry entry = zipFile.getEntry(resourceName);
      if (entry != null) {
        return zipFile.getInputStream(entry);
```

(b) Code fragment in a clone set whose RNR(S) is 97, the second highest RNR(S) value in Ant 1.7.0.

```
    out.print("<!ELEMENT target (");
    out.print(TASKS);
    out.print(" | ");
    out.print(TYPES);
    out.println(")*>");
    out.println("");

    out.println("<!ATTLIST target");
    out.println("      id       ID   #IMPLIED");
    out.println("      name     CDATA #REQUIRED");
    out.println("      if       CDATA #IMPLIED");
    out.println("      unless   CDATA #IMPLIED");
    out.println("      depends  CDATA #IMPLIED");
```

(c) Code fragment in a clone set whose POP(S) is 21, the highest POP(S) value in Ant 1.7.0.

Figure 3: Examples of code fragments in clone set in Ant 1.7.0

3. PROPOSED METHOD

Based on observation described in Section 2.2, we propose a method for extracting code clones using combinations of clone metrics in this study.

The details regarding the proposed method and its advantages that we expect are the followings:

1. Clone sets whose LEN(S) and RNR(S) are higher than other clone sets
   - Clone metric RNR(S) eliminates types of if (or if-else) blocks detected by higher LEN(S) values.
   - Clone Metric LEN(S) eliminates small size code clones detected by higher RNR(S) values.
2. Clone sets whose LEN(S) and POP(S) are higher than other clone sets
   - Only if code clones in a clone set have similar or same constructs of token sequence, the clone metric POP(S) value increases. Therefore, clone metric POP(S) eliminates consecutive if blocks detected by higher LEN(S) values.
   - Clone metric LEN(S) eliminates small size clone sets detected by higher POP(S) values.
3. Clone sets whose RNR(S) and POP(S) are higher than other clone sets
   - Clone metric RNR(S) eliminates language dependent code clones detected by higher POP(S) values. Therefore, It is highly possible that these clone sets include specific logics, because consecutive if (or if-else) blocks, case entries of switch statements, consecutive variable declarations are not included.
   - higher POP(S) values mean code clones in a clone set appear more frequently in software. Therefore, it improves the maintainability to perform refactoring to these clone sets.
4. Clone sets whose LEN(S), RNR(S), and POP(S) are higher than other clone sets
   - Clone metric RNR(S) eliminates language dependent code clones.
   - Clone sets whose higher POP(S) apply good motivation for refactoring to developers because to perform refactoring to code clones in a clone set appear more frequently in software improves the maintainability.
   - Clone metric LEN(S) eliminates small size clone sets.

4. INDUSTRIAL CASE STUDY

To validate our proposal method, we performed a case study. Our case study subject is an industrial software that was developed by NEC, a Japanese multinational IT company. This section describes the study steps and study subjects. Results of case study are then reported. Finally, threats to validity are discussed.

4.1 Study Steps

The steps of the case study and its explanation are the following:

1. Gemini extracted clone sets from CCFinder's output.

   The input of CCFinder was software developed by NEC. NEC plans to maintain the software for the next 10 years. Developers wanted to perform refactoring of this software for maintenance efficiency. Therefore, it was adequate software to extract clone sets for refactoring.

2. We selected clone sets from Gemini's output using clone metrics.
   - Based on following assumptions, we selected metrics LEN(S), RNR(S), and POP(S).

  - To perform refactoring small size clone sets, and a few code clones is more ineffective than long size clone sets, and numerous number of code clones.
  - It is ineffective to check language-dependent code clones for refactoring.
- We selected clone sets detected by higher combined clone metrics values. To compare a result of these clone sets, clone sets whose individual clone metric value is higher are also selected.

3. We conducted 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.
   - We asked him to fill out surveys considering feasibility of performing refactoring and cost in maintenance.
   - The survey includes a list of selected code clones and a question for each clone set. The question asks whether developers would perform refactoring or not. The developer answered using the following options:
     - Perform refactoring.
     - Write comments about code clones, but don't perform refactoring.
     - Change nothing.
     - Others.
4. We analyzed the result of survey.
   - If clone sets were marked as "Write comments about code clones, but don't perform refactoring" or "Change nothing" or "Others", we regarded those clone sets as inappropriate clone sets for refactoring.
   - If clone sets detected by higher combined clone metrics were more marked as "Perform refactoring" than clone sets detected by higher individual clone metric value, we can say our proposed method successfully to selected the clone sets for refactoring.

4.2 Study Subject

The subject of this case study was a web-application software implemented in Java. It is 110KLOC across 296 files. In this study, we set 30 tokens as the minimum token length of a code clone. As CCFiner's output, 736 clone sets are detected. Due to the limitation of time and effort, we selected each 62 clone sets detected by higher individual clone metric value, and combined clone metrics values. The following are 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.

Table 1: Correlation between clone metrics LEN, RNR, and POP

	LEN	RNR	POP
LEN	-		
RNR	−0.02	-	
POP	−0.20	−0.12	-

$S_{RNR \cdot POP}$ **:** 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) · POP(S) value are high rank in the top 15.

Note that the intersection of the set $S_{LEN} \cup S_{POP} \cup S_{RNR}$ and the set $S_{LEN \cdot RNR} \cup S_{LEN \cdot POP} \cup S_{RNR \cdot POP}$ is only 5 in all of 62 surveyed clone sets.

Here, we determined the correlation between metrics LEN, RNR, and POP To confirm our proposal method. Table 1 describes Spearman correlation coefficient. As shown in Table 1, all correlations were smaller than 0.50 Therefore, there is no significant relationship among the clone metrics LEN, RNR and POP.

4.3 Result of Survey

To analyze the result, we use *precision* and *average precision*. Here, purpose of using precision and average precision are described below:

Precision : To investigate the question "How many refactoring candidates were accepted by developer?"

Average precision : To investigate the question "Are refactoring candidates shown in high rank?". It takes value 0 and 1, with higher values indicating that clone sets of higher metrics values are more accepted as refactoring candidates.

Let S_{All} represent ALL clone sets that are selected by each method, S_{ARC} represents clone sets Accepted as Refactoring Candidates by a developer.

$$Precision = \frac{|S_{ARC}|}{|S_{All}|}$$

Additionally, let $|S_{ARC}| = N$, $Precision(r)$ represents the Precision of a clone set who accepted as a refactoring candidate at rank r.

$$Average\,Precision = \frac{\sum_{i=1}^{N} Precision(r)}{N}$$

Table 2 and Table 3 describe the result of the survey. [3]

In Table 2 and Table 3, column `Clone Sets` shows name of subject clone sets. `#Selected Clone Sets` and `#Refactoring` shows the number of selected clones and the number of clone sets marked as "Perform refactoring" respectively.

[3]In Table 2, LEN(S), and LEN(S) · POP(S) have two marks "Perform refactoring" and "Write comments" since those clone sets comprise two parts: one is a refactoring target and another should be commented. They were counted as "Perform refactoring."

Table 2: Results, precision, and average precision of each clone set in the survey

Clone Sets	#Selected Clone Sets	#Refactoring	Precision	Average Precision
S_{LEN}	10	7	**0.70**	0.57
S_{RNR}	10	4	0.40	0.32
S_{POP}	10	3	0.30	0.26
$S_{LEN \cdot RNR}$	15	13	**0.87**	**0.95**
$S_{LEN \cdot POP}$	7	6	**0.86**	**0.92**
$S_{RNR \cdot POP}$	18	14	**0.78**	**0.97**
$S_{LEN \cdot RNR \cdot POP}$	1	1	**1.00**	**1.00**

Table 3: Results of clone sets detected by higer individual clone metric, and combined metrics in the survey

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

As shown in the column "Precision" in Table 2, a clone set who has the lowest precision(0.30) is S_{POP}, and a clone set who has the highest precision(1.00) is $S_{LEN \cdot RNR \cdot POP}$. This means using metrics LEN(S), RNR(S), and POP(S) are the most acceptable for refactoring, and using metric RNR is unacceptable to extract clone sets for refactoring. However, number of LEN(S) · RNR(S) · POP(S) are only one, we need a further investigation that these clone sets are really the most the most appropriate clone sets for refactoring.

The details of precision are the followings. The precisions of S_{RNR}, S_{POP} are smaller than 0.50. This means that these clone sets are appropriate for refactoring. However, the 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 (They are shown in bold in column "Precision" in table 2), this means that these clone sets are inappropriate for refactoring.

Nevertheless, we had a strong concern that S_{LEN} are really appropriate clone sets for refactoring. Therefore, we introduced *average precision* to confirm that using clone metric LEN(S) is really acceptable to extract clone sets for refactoring. As shown in Table 2, *average precision* of S_{LEN} is 0.57. Contrary to this, the *average precisions* of all clone sets detected by higher combined clone metrics are over 0.90(They are shown in bold in column "Average Precision" in table 2). Clone sets who has higher average precisions mean that clone sets detected by higher clone metric values are more appropriate for refactoring than clone sets detected by lower clone metric values . In fact, top 2 clone sets of S_{LEN} were not marked as "Perform refactoring" in the survey. However, top 2 clone sets of all clone sets detected by high combined clone metrics were marked as "Perform refactoring" by a developer. Hence, to extract refactoring candidate clone sets, using clone metric LEN(S) is unacceptable.

Moreover, as shown in Table 3, Precision of clone sets detected by higher combined clone metrics (They are shown in bold in column "Precision" in table 3) are higher than precision of clone sets detected by higher individual clone metric. In conclusion, using higher combined clone metrics is more appropriate to extract clone sets for refactoring than using each individual clone metric.

4.4 Threats to Validity

Because of the limitation of developer's time and effort, the survey included 10 clone sets for higher individual clone metric, and 41 clone sets selected by the combinations of higher clone metrics. If we selected all clone sets in software for survey, we might get different results. In addition, It is highly possible to lost relevant clone sets for refactoring due to filtering. However, as we mentioned Section 4.3, clone sets whose combined clone metrics values are higher were more accepted as refactoring candidates in the survey. Thus, we believe that the result of survey dose not significantly change if we checked all code clone sets in software.

Moreover, our case study is conducted on a single system and got feedback from one developer. Therefore, although we could successfully validate our method in this study, our method may not generalize to other software. To improve generality, we need to investigate other software. We are planning to assess our method with open source software based on their changed history, investigate recall. However, there is a small chance to analyze industrial software in a refactoring phase prior to its release.

5. RELATED WORK

Jiang *et al.*[6] and Kapser *et al.*[9] pointed out that code clone detection tools using parameterized matching detected a lot of false positives. Jiang *et al.*[6] 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.*[9] 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[7] 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 are not significantly effective.

Balazinska [1] *et al.* and Higo [4] *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 [4] *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.

Here, we summarize the main originality of our work.

- To remove false positives, we propose a method that combines multiple metrics. Those metrics are categorized into two types. LEN(S) and POP(S) metrics focus on the amount of code duplication. On the other hand, RNR(S) metric focuses onwhether or not code clones belonging to a clone set are language-dependent code.
- In the case studies, our method extracted a lot of clone sets that are evaluated as refactoring candidates by a NEC developer.

6. CONCLUSION AND FUTURE WORK

In this paper, we proposed a method to extract appropriate clone sets for refactoring from the output of CCFinder. Also, we showed the usefulness of the proposed method with an industrial case study using the source code developed by NEC. The case study indicates the proposed method is more efficient than using individual clone metric to extract clone sets for refactoring.

As future work, we are planning to perform case studies of open source software and investigate the usefulness of proposed method. Moreover, we would like to improve our proposed method by taking previous filtering and clone metrics described in 5 section to proposed method. Finally, we would like to check the following factors, for the reason that these factors mainly influence the estimated trade off between effort for clone removal and estimated savings due to better future maintainability.

- Number and type of differences between code clones
- Location of the code clones in the software
- Ownership issues
- Test coverage

Acknowledgments

We express our great thanks to Ms. Fusako Mitsuhashi and Mr. Shin'ichi Iwasaki of NEC Corporation for data collection, and Dr. Daniel German of the University of Victoria for helpful comments on earlier revisions of this paper. We also thank Dr. Simone Livieri of Osaka University for proofreading this paper. This work is being conducted as a part of Stage Project. Also, this work is partially supported by JSPS, Grant-in-Aid for Scientific Research (A) (21240002) and Grant-in-Aid for Research Activity start-up(22800040).

7. REFERENCES

[1] 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.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

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

[10] M. Kim, V. Sazawal, D. Notkin, and G. Murphy. An empirical study of code clone genealogies. In *Proc. of ESEC/FSE 2005*, pages 187–196, 2005.

[11] C. Roy and J. Cordy. Scenario-based comparison of clone detection techniques. In *Proc. of ICPC 2008*, pages 153–162, 2008.

[12] 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.