Industrial Application of Clone Change Management System

Yuki Yamanaka *, Eunjong Choi *, Norihiro Yoshida †, Katsuro Inoue *, Tateki Sano ‡
* *Graduate School of Information Science and Technology, Osaka University, Japan*
{*y-yuuki, ejchoi, inoue*}*@ist.osaka-u.ac.jp*
† *Graduate School of Information Science, Nara Institute of Science and Technology, Japan*
yoshida@is.naist.jp
‡ *Software Process Innovation and Standardization Division, NEC Corporation, Japan*
t-sano@cp.jp.nec.com

***Abstract*—Clone change management is one of crucial issues in open source software(OSS) development as well as in industrial software development (e.g., development of social infrastructure, financial system, and medical equipment). When an industrial developer fixes a defect, he/she has to find the code clones corresponding to the code fragment including it. So far, several studies performed on the analysis of clone evolution in OSS. However, to our knowledge, a few researches have been reported on an application of a clone change management system to industrial development process. In this paper, we propose a clone change management system based on the categorization of clone evolution, and then present case study of industrial application. In case study, we confirmed that the proposed system suggested two unintentionally developed clones in a half of the month.**

***Keywords*-Code Clone, Clone Evolution, Software Maintenance, Change Management**

I. Introduction

During software maintenance, change management of code clones is required for consistent change [1], [2], [3] of them, and identification of simultaneously changed clones as refactoring opportunity [4].

So far, several studies are done on change management of code clones in a software system. For example, Duala-Ekoko et al. presented *Clone Region Descriptors* to track code clones moved to other locations in source code [2], [3]. Also, Nguyen et al. have developed a clone management tool *JSync* to notify developers change and its inconsistency of code clones in source files [5].

Change management of code clones is required not only in open source projects but also in industry. In industrial development, the consistency of change has to be rigorously inspected to prevent post-release defects, especially in developments of social infrastructure, financial system, and medical equipment. However, to our knowledge, only a few studies have been reported on an applications of clone change management systems to industrial development process [6], [7].

In this paper, we present a clone change management system based on the categorization of clone evolution.

The system regularly reports change information of code clones and clone sets (i.e., a set of code clones identical or similar to each other) in a software system. For example, modified code clones in a clone set and newly created clone sets are reported daily to developers. The changed information is provided through web-based user interface (see Figure 4) and e-mail notification.

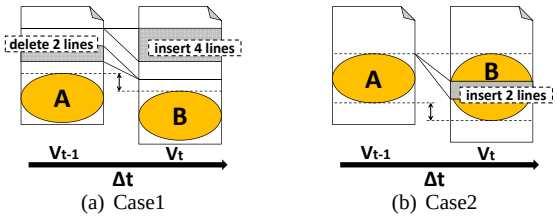
In case study, we applied the clone change management system to the development process of industrial software in NEC. As a result, we confirmed that the clone management system suggested two unintentionally developed clones in a half of the month. The contributions of this study are as follows:

- We developed a clone change management system with the feature of the categorization of clone evolution, which is based on the opinions of industrial developers in NEC.
- We report an industrial application of the clone change management system. The result shows that the system suggested two unintentionally developed clones.

II. Proposed Method

Code clones are often changed (e.g., added, modified) when a software system evolves. The comprehension of the code clones at detection time is insufficient for managing code clones efficiently. A developer needs to understand and apply an appropriate solution correspond to an evolution pattern of code clones. For example, the developer could merge a new added set of code clones into a single method. On the other hand, he/she should consider modifications of all code clones in a clone set, in the case of modification was occurred in one of code clones in the clone set. We categorize code clones and clone sets based on the evolution patterns between two versions of source code, and then implement a clone change manage system based on the categorization. We used CCFinder [8] to detect code clones. CCFinder detects Type-1 and Type-2 code clones [9] and outputs the locations of them in source code.

To describe categorization of code clones and clone sets, we defined V_i as source code at the point of time i, and C_i as a set of code clones detected in V_i. Input of this method is V_t (the latest version of source code) and V_{t-1}



(a) Case1 (b) Case2

Figure 1. Tracing code clones: parent-child relationship of code clones

(the previous version of source code). This method consists of the following steps:

Step1 : Detect C_t and C_{t-1} by analyzing overall V_t and V_{t-1} using CCFinder.

Step2 : Trace code clones between two versions based on a method which is described in section II-A.

Step3 : Categorize code clones in C_t and C_{t-1} based on a definition which is described in section II-B.

Step4 : Categorize clone sets in C_t and C_{t-1} based on a definition which is described in section II-C.

A. Tracing Code Clones

In our previous study, code clones were traced across multiple versions based on correspondence of the start and end line of them in source code. We used the same method in our previous study [10] as a reference to trace code clones between two versions.

We defined the *parent-child relationship* to trace code clones. If code clone $B \in C_t$ corresponds to code clone $A \in C_{t-1}$, we define B as a *child clone* of A, and A as a *parent clone* of B.

For example, code fragment which is located above code clone $A \in C_{t-1}$ was modified between two versions shown in Figure 1(a) (four lines were inserted also two lines were deleted above A). In this case, code fragment B in V_t corresponding to A can be traced by counting inserted and deleted lines. Thus, the start and end line numbers of B are increased by two lines. When C_t contains B, we define B as a child clone of A. On the other hand, code clone $A \in C_{t-1}$ was modified between two versions shown in Figure 1(b) (two lines were inserted to A). In this case, code fragment B in V_t corresponding to A can be traced by counting and inserted lines in A. Thus, the start line number of B is the same as A, and the end line number of B is increased by two lines. When C_t contains B, we define B as a child clone of A.

B. Categorization of Code Clones

In this method, all code clones in V_t and V_{t-1} are categorized based on evolution patterns of them. We defined the following five categories of code clones by using propositional function about code clone $X_c \in C_t$ and $X_p \in C_{t-1}$ shown in Table I.

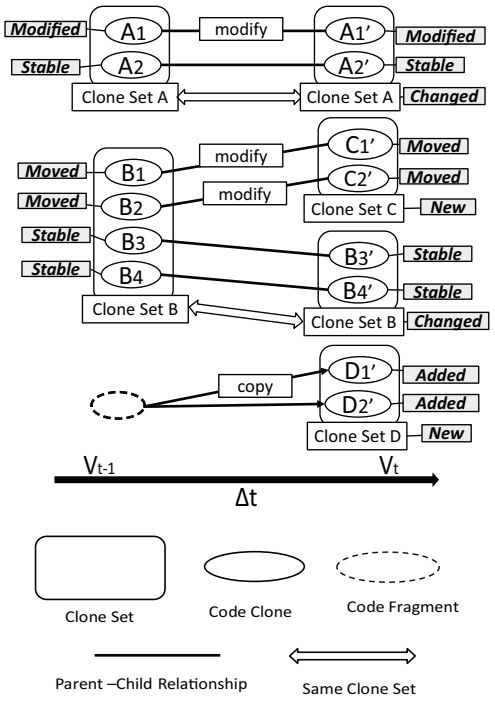


Figure 2. Example of categorzation of code clones and clone sets

Table I
DEFINITION OF PROPOSITIONAL FUNCTION

function	definition
$P(X_c)$	Parent clone of X_c exists in V_{t-1}
$C(X_p)$	Child clone of X_p exists in V_t
$M(X_c)$	X_c was modified between two versions
$CP(X_c)$	A pair of X_c and its parent clone is a clone pair

- *Stable clone*: X_c satisfies $P(X_c) \wedge \neg M(X_c)$
- *Modified clone*: X_c satisfies $P(X_c) \wedge M(X_c) \wedge CP(X_c)$
- *Moved clone*: X_c satisfies $P(X_c) \wedge M(X_c) \wedge \neg CP(X_c)$
- *Added clone*: X_c satisfies $\neg P(X_c)$
- *Deleted clone*: X_p satisfies $\neg C(X_p)$

For example, modified clone means that some code clones of clone set are modified. However, they are contained in the same clone set between two versions. And also, added clone means that the code clone is added newly in V_t.

Additionally, the parent clone of stable, modified or moved clone is categorized into the same category of it.

C. Categorization of Clone Sets

In this method, all clone sets in V_t and V_{t-1} are categorized based on evolution patterns of them. We defined the following four categories of clone sets.

- *Stable clone set:* Only stable clones are shared between two clone sets involved in V_t and V_{t-1} respectively.
- *Changed clone set:* Modified, moved, deleted and added clones are shared between two clone sets involved in V_t and V_{t-1} respectively.

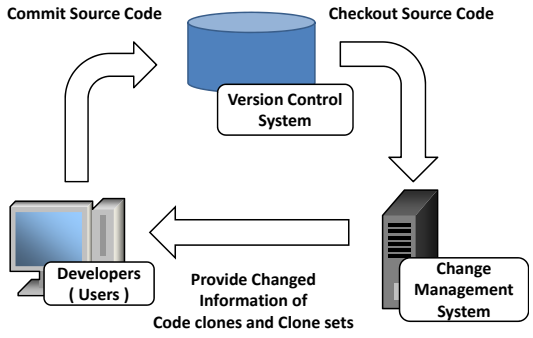


Figure 3. Process of change management system

- *New clone set:* Clone sets involved in only V_t.
- *Deleted clone set:* Clone sets involved in only V_{t-1}.

Changed clone sets contain stable clones, modified clones or moved clones such as clone set A and B in Figure 2 should be modified consistently. Meanwhile, new clone sets that contain added clones such as clone set D in Figure 2 could be merged into a single method.

III. Change Management System

We developed a system for clone change management. It takes two versions of source code as input and categorizes code clones and clone sets comprised of them which is described in section II. Figure 3 shows the process of this system. This system assumes that the developers use version control system such as Subversion [1] in software development. The process of this system is comprises of following steps:

Step1 : Get the current version of source code from version control system as the latest version V_t. [2]

Step2 : Categorize code clones and clone sets between V_t and V_{t-1} which is described in section II.

Step3 : Generate html files for web-based user interface (UI) and a text file for e-mail notification [3].

Step4 : Send an e-mail with generated text file to developers.

As described above, the system provides information of changed code clones and clone sets between the two versions through web-based UI and e-mail notification as following:

- Web-based UI: Figure 4 shows clone set list page and source file page in the web browser. Clone set list page displays the list of clone sets (Figure 4(a)). Users can move to the corresponding source file page by clicking the links of each code clone. Source file page displays code clones that are involved in the selected clone set in clone set list page(Figure 4(b)). Each code clone is highlighted on this page.
- E-mail notification: The list of clone sets categorized into changed, new and deleted clone sets provided in the developers by an e-mail notification. Each information of clone set consists of the code clone list that involved in each clone set and code fragment of each code clone.

[1]http://subversion.tigris.org/.

[2]we use source code that were analyzed in the last time as the previous version V_{t-1}.

[3]Note that the analysis of the system is stored for two months as html files.

Clone Set ID:66

ID	Category	File Name	Location
79	DELETED	¥src¥main¥org¥apache¥tools¥ant¥filters¥ExpandProperties.java	73-87
117	STABLE	¥src¥main¥org¥apache¥tools¥ant¥filters¥PrefixLines.java	86-100
138	STABLE	¥src¥main¥org¥apache¥tools¥ant¥filters¥SuffixLines.java	87-101

Clone Set ID:44

ID	Category	File Name	Location
232	MODIFIED	¥src¥main¥org¥apache¥tools¥ant¥listener¥MailLogger.java	375-380
82	STABLE	¥src¥main¥org¥apache¥tools¥ant¥filters¥FixCrLfFilter.java	143-148
87	STABLE	¥src¥main¥org¥apache¥tools¥ant¥filters¥FixCrLfFilter.java	144-149
823	STABLE	¥src¥main¥org¥apache¥tools¥ant¥taskdefs¥MacroInstance.java	248-253

(a) Clone set list page

```
92      */
        [START ID:78(Deleted Clone) CLONESET:39(Deleted Clone Set)]
93      public int read() throws IOException {
94  +       if (index > EOF) {
95  +           if (buffer == null) {
96  +               String data = readFully();
    -
        [START ID:79(Deleted Clone) CLONESET:66(Deleted Clone Set)]
    -       int ch = -1;
    -
    -       if (queuedData != null && queuedData.length() == 0) {
    -           queuedData = null;
    -       }
    -
    -       if (queuedData != null) {
    -           ch = queuedData.charAt(0);
    -           queuedData = queuedData.substring(1);
    -           if (queuedData.length() == 0) {
    -               queuedData = null;
    -           }
    -       } else {
        [END ID:78]
    -           queuedData = readFully();
    -           if (queuedData == null || queuedData.length() == 0) {
        [END ID:79]
    -               ch = -1;
    -           } else {
```

(b) Source file page

Figure 4. Example of web-based UI

Table II
THE ANSWER FOR Q2

	Category	Q2(a)	Q2(b)	Q2(c)
Clone Set A	New	No	Yes	Merge clone set
Clone Set B	New	No	Yes	Merge clone set

IV. Case Study

To confirm the usefulness of our developed system, we applied it to an industrial software development in NEC, a Japanese multinational IT. The target of software development is a web-application software implemented in Java. It is 120KLOC in 350 files. We set 30 tokens as the

Table III
CLONE SETS CATEGORIZATION OF CASE STUDY2

Analysis day	#Stable	#Changed	#New	#Deleted
December 19, 2011	873	4	13	1
December 28, 2011	833	34	37	14
December 29, 2011	895	1	0	2
January 6, 2012	895	0	0	3

```
for (int i = 0; i < contents.size(); i++) {
    try {
        Content content1 = contents.get(i);
        Content content2 = (Content) content1.clone();
        content2.setTitle(StringUtil.concatPath(toPath, content1.getName()),
    true);
        if (content1 instanceof Page) {
            copyPage((Page) content1,(Page) content2);
        } else if (content1 instanceof File) {
            copyFile((File) content1,(File) content2);
        } else if (content1 instanceof Category) {
            copyCategory((Category) content1,(Category) content2);
        }
    } catch (Exception e) {
        if (!(e instanceof AccessDeniedException))
            throw e;
    }
}
```

(a) Code fragment for copy contents

```
for (int i = 0; i < contents.size(); i++) {
    try {
        Content content1 = contents.get(i);
        Content content2 = (Content) content1.clone();
        content2.setTitle(StringUtil.concatPath(toPath, content1.getName()),
    true);
        if (content1 instanceof Page) {
            move Page((Page) content1,(Page) content2);
        } else if (content1 instanceof File) {
            move File((File) content1,(File) content2);
        } else if (content1 instanceof Category) {
            move Category((Category) content1,(Category) content2);
        }
    } catch (Exception e) {
        if (!(e instanceof AccessDeniedException))
            throw e;
    }
}
```

(b) Code fragment for move contents

Figure 5. Example of a clone set that needs additional maintenance

minimum token length of a code clone to CCFinder. The developer applied the developed system from December 18, 2011 to January 31, 2012, and it analyzed four times during this period. Moreover, we conducted a survey of the our system to a developer, a project manager with 10 years of development experiences with Java. This survey consists of the following questions:

- Q1: By using this system, were you able to find any harmful clone sets that need additional maintenance (e.g., refactoring, consistently change)?
- Q2: If the answer of Q1 is "Yes", please answer following questions.
  - (a): Have you already noticed the existence of it before applying our system?
  - (b): Does this clone set need additional maintenance?
  - (c): Please select appropriate solutions to each clone set:
    - Merge the clone set into a single method.
    - Modify the clone set consistently.
    - Others.

Q1 is the question to confirm that the developer could find a clone set that needs additional maintenance. Moreover, if the answer for Q2(a) is "No", we regarded that the results of this system is useful for software development because a developer could find clone sets that are necessary to be performed additional maintenance by using the developed system. That are necessary to be performed additional maintenance by using the developed system. Q2(b) and Q2(c) are the auxiliary questions.

As a result the survey, we are able to evaluate the developed system because the answer for Q1 was "Yes". The developer could find two clone sets that needed additional maintenance according to this survey. Table II shows the answers for Q2 in the survey. These clone sets appeared in the December 28, 2011 and categorized into new clone set. Figure 5 shows an example of one of them. This clone set consisted of two code clones. One performed copy contents(see Figure 5(a)) and the other performed move contents(see Figure 5(b)). They performed a similar role. Therefore, the developer extracted common code fragment from them and merged into a single method immediately. Another clone set also consisted of two code clones that performed a similar role. Therefore, they were also merged into a single method immediately. As you can see this result, the developer could find clone sets that needed to be merged into a single method immediately by using the developed system.

Table III shows the result of categorization, the number of clone sets of with category at analyzed data. In addition, the most of clone sets were categorized into stable at each day as shown in Table III. Therefore, it is difficult to find changed, new, deleted clone sets from overall clone sets. Therefore, developers could find these clone sets easily by using our system and the cost of clone changed management were reduced.

V. RELATED WORK

A lot of studies have been done for investigating and supporting clone evolution [6], [7], [11], [12], [13], [14], [15], [16], [17], [18].

Kim et al. studied genealogies of code clone [11]. They defined a model of clone genealogy in order to study evolution of code clones across multiple versions of source code. We presented a clone change management system with the feature of the categorization of clone evolution, which is based on the opinions of industrial developers in NEC, and then reported an industrial application of the clone change

management system. Also, other models of clone evolution have been proposed, and discussed [12], [13], [14], [15]. We should improve our model of clone evolution based on those existing models.

Duala-Ekoko et al. presented *Clone Region Descriptors* to track code clones moved to other locations in source code [2], [3]. The tracking code clones in proposed system is based on text difference and similarity. For more accurate tracking code clones, we should integrate *Clone Region Descriptors* into proposed system.

Nguyen et al. have developed a clone management tool *JSync* to notify developers change and its inconsistency of code clones in source files [5]. Also, Jiang et al. [19] and Li et al. [20] have been proposed on the inconsistency detection of code clones from a single version of source code. Proposed system is promising to support inconsistency detection of code clones but unable to show inconsistency between code clones explicitly. We should add the feature of inconsistency detection by them to proposed system.

VI. Conclusion and Future Work

In this paper, we proposed a clone change management system based on the categorization of clone evolution. Our developed system categorizes code clones and clone sets based on their evolution patterns and reports changed information through a web-based UI and e-mail notification. Additionally, we showed the usefulness of proposed system by applying it to the development of an industrial software in NEC.

As future work, we would like to investigate the usefulness of proposed system by applying other software developments for long term. Also, we plan to extend our system based on *Clone Region Descriptors* [3].

Acknowledgment

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 Scientific Research (A) (21240002).

References

[1] P. Jablonski and D. Hou, "CReN: a tool for tracking copy-and-paste code clones and renaming identifiers consistently in the IDE," in *Proc. of OOPSLA eclipse '07*, 2007, pp. 20–31.

[2] E. Duala-Ekoko and M. P. Robillard, "Tracking code clones in evolving software," in *Proc. of ACM/IEEE ICSE '07*, 2007, pp. 158–167.

[3] ——, "Clone Region Descriptors: Representing and Tracking Duplication in Source Code," in *Proc. of ACM TOSEM '10*, 2010, pp. 20–31.

[4] R. Geiger, B. Fluri, H. Gall, and M. Pinzger, "Relation of code clones and change couplings," in *Proc. of FASE '06*, 2006, pp. 411–425.

[5] H. A. Nguyen, T. T. Nguyen, N. H. Pham, J. Al-Kofahi, and T. N. Nguyen, "Clone management for evolving software," *IEEE Transactions on Software Engineering*, vol. 99, no. PrePrints, 2011.

[6] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner, "Do code clones matter?" in *Proc. of ICSE '07*, 2009, pp. 485–495.

[7] J. O. Bailey and E. L. Burd, "Evaluating clone detection tools for use during preventative maintenance." in *Proc. of SCAM '02*, 2002, pp. 36–43.

[8] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A multilinguistic token-based code clone detection system for large scale source code," *IEEE Transactions on Software Engineering*, vol. 28, no. 1, pp. 654–670, 2002.

[9] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, "Comparison and evaluation of clone detection tools," *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 804–818, 2007.

[10] S. Kawaguchi, M. Matsushita, and K. Inoue, "Clone history analysis using configuration management system," *IEICE Transactions (in Japanese)*, vol. J89-D, no. 10, pp. 2279–2287, 2006.

[11] M. Kim, V. Sazawal, D. Notkin, and G. C. Murphy, "An empirical study of code clone genealogies," in *Proc. of ESEC/SIGSOFT FSE '05*, 2005, pp. 187–196.

[12] L. Aversano, L. Cerulo, and M. Di Penta, "How clones are maintained: An empirical study," in *Proc. of CSMR '07*, 2007, pp. 81 –90.

[13] T. Bakota, R. Ferenc, and T. Gyimothy, "Clone smells in software evolution," in *Proc. of ICSM '07*, 2007, pp. 24–33.

[14] J. Krinke, "A study of consistent and inconsistent changes to code clones," in *Proc. of WCRE '07*, 2007, pp. 170 –178.

[15] J. Harder and N. Gode, "Modeling clone evolution," in *Proc. of IWSC '09*, 2009, pp. 17–21.

[16] N. Gode, "Evolution of type-1 clones," in *Proc. of SCAM '09*, 2009, pp. 77–86.

[17] N. Gode and J. Harder, "Clone stability," in *Proc. of CSMR '11*, 2011, pp. 65–74.

[18] T. T. Nguyen, H. A. Nguyen, J. M. Al-Kofahi, N. H. Pham, and T. N. Nguyen, "Scalable and incremental clone detection for evolving software," in *Proc. of ICSM '09*, 2009, pp. 491–494.

[19] L. Jiang, Z. Su, and E. Chiu, "Context-based detection of clone-related bugs," in *Proc. of ESEC/SIGSOFT FSE '07*, 2007, pp. 55–64.

[20] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, "Cp-miner: Finding copy-paste and related bugs in large-scale software code," *IEEE Transactions on Software Engineering*, pp. 176–192, 2006.