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 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 $V_i$ (the latest version of source code) and $V_{i-1}$
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.

<table>
<thead>
<tr>
<th>function</th>
<th>definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P(X_c) )</td>
<td>Parent clone of ( X_c ) exists in ( V_{t-1} )</td>
</tr>
<tr>
<td>( C(X_p) )</td>
<td>Child clone of ( X_p ) exists in ( V_t )</td>
</tr>
<tr>
<td>( M(X_c) )</td>
<td>( X_c ) was modified between two versions</td>
</tr>
<tr>
<td>( CP(X_c) )</td>
<td>A pair of ( X_c ) and its parent clone is a clone pair</td>
</tr>
</tbody>
</table>

- **Stable clone**: \( X_c \) satisfies \( P(X_c) \) \( \land \) \( \neg M(X_c) \)
- **Modified clone**: \( X_c \) satisfies \( P(X_c) \) \( \land \) \( M(X_c) \) \( \land \) \( CP(X_c) \)
- **Moved clone**: \( X_c \) satisfies \( P(X_c) \) \( \land \) \( M(X_c) \) \( \land \) \( \neg CP(X_c) \)
- **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.
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 comprised of following steps:

Step 1: Get the current version of source code from version control system as the latest version \(V_t\). \(^2\)

Step 2: Categorize code clones and clone sets between \(V_t\) and \(V_{t-1}\) which is described in section II.

Step 3: Generate html files for web-based user interface (UI) and a text file for e-mail notification \(^3\).

Step 4: 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.

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

---

\(^1\)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.
for (int i = 0; i < contents.size(); i++) {
try {
Content content1 = contents.get(i);
Content content2 = (Content) content1.clone();
}
} 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) {
    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;
}
}

(b) Code fragment for move contents

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

Table III
CLONE SETS CATEGORIZATION OF CASE STUDY 2

<table>
<thead>
<tr>
<th>Analysis day</th>
<th>#Stable</th>
<th>#Changed</th>
<th>#New</th>
<th>#Deleted</th>
</tr>
</thead>
<tbody>
<tr>
<td>December 19, 2011</td>
<td>873</td>
<td>4</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>December 28, 2011</td>
<td>833</td>
<td>34</td>
<td>37</td>
<td>14</td>
</tr>
<tr>
<td>December 29, 2011</td>
<td>895</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>January 6, 2012</td>
<td>895</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

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 are 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 of 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 clone codes, 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 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


