CCEvovis: A Clone Evolution Visualization System for Software Maintenance

Hirotaka Honda*, Shogo Tokui*, Kazuki Yokoi*, Eunjong Choi‡, Norihiro Yoshida†, and Katsuro Inoue*
*Osaka University, Japan, {h-honda, s-tokui, k-yokoi, inoue}@ist.osaka-u.ac.jp
‡Nara Institute of Science and Technology, Japan, choi@is.naist.jp
†Nagoya University, Japan, yoshida@ertl.jp

***Abstract*—Understanding the evolution of code clones is important in software maintenance. With the information about how code clones evolve, both developers and researchers can understand the impacts of code clones and build a more robust code clone management system. So far, many studies have investigated the evolution of code clones to better understand the effects of code clones. However, only a few systems have been presented to support managing code clones based on the information about how code clone evolves. To mitigate this problem, in this paper, we present CCEvovis, a system that visualizes the evolved code clones across multiple versions of a program. CCEvovis highlights and visualizes the clone change to support software maintenance. CCEvovis is available at: https://github.com/hirotaka0616/CCEvovis.**

***Index Terms*—code clone, software maintenance, visualization**

I. Introduction

Code clones, duplicated code, are created by copying and pasting existing code with/without modifications. Most of the software contains a significant amount of code clones. They cause additional effort in comprehending and maintaining a system.

Several recent systems for the management and visualization of code clones do exist to support maintenance of code clones [1]. By using these systems, developers can effectively apply consistent modifications to a clone set (i.e., a set of code clones that are identical or similar to each other) or find candidates for clone refactoring (i.e, merging code clones into a single unit) [2]. In the past, our research group developed a code clone management system called Clone Notifier [3], which reports creation and change of code clones between two consecutive versions. We also confirmed that a developer successfully identified ten candidates for clone refactoring by using Clone Notifier. However, it has a limitation in that it only provides change information of code clones between two versions. Understanding the evolution of code clones are important for maintenance of code clones. With the information about how code clones evolve, developers can understand the impacts of code clones and manage them accordingly. However, because Clone Notifier provides change information of code clones between two versions, developers might overlook important information for managing code clones in the context of clone evolution.

To mitigate this problem, in this paper, we present CCEvovis, a system for visualizing the evolution of code clones to support code clone maintenance. CCEvovis detects code clones across multiple versions of a program and provides and visualizes and highlights the evolution of code clones based on their evolution patterns.

II. CCEvovis

CCEvovis is a system that visualizes the evolved code clones across multiple versions of a program to support maintenance of code clones. Fig. 1 shows the architecture overview of CCEvovis for two versions of a program. As can be seen in this figure, CCEvovis is comprised of the following four steps: (1) Detect Clones, (2) Map Clones, (3) Categorize Clone Sets, and (4) Visualize the Clone Evolution.

A. Detect Clones

At first, CCEvovis detects code clones C_i and C_{i+i} from two given consecutive versions v_i and v_{i+i} of a program. Note that each version can be also directly extracted from Github repository. To detect code clones, CCEvovis allows a user to choose a code clone detection tool among following representative code clone detection tools.

- **Vector-based clone detection tool [4]:** a lightweight tool for detecting function clones using information retrieval techniques
- **CCVolti [5]:** a code clone detection tool that detects block-level code clones
- **CCFinderX[1]:** a token-based code clone tool that detects syntactically similar code clones
- **SourcererCC [6]:** a token-based code clone detection tool for very large scale source code

B. Map Clones

To map detected code clones C_i and C_{i+i} between two versions, CCEvovis uses the same method with the Clone Notifier. That is, it maps code clones based on correspondence of the start and end line of them in source code between versions. It also uses the *parent-child relationship* to code clones between two versions to map code clones. Let a code clone $c_i \in C_i$ is detected in a version v_i and a code clone $c_{i+1} \in C_{i+1}$ is detected in a version v_{i+1} in the two consecutive versions v_i and v_{i+1}. Moreover, three lines were newly inserted to code clone c_{i+1} between two versions. In

[1]http://www.ccfinder.net/ccfinderx-j.html

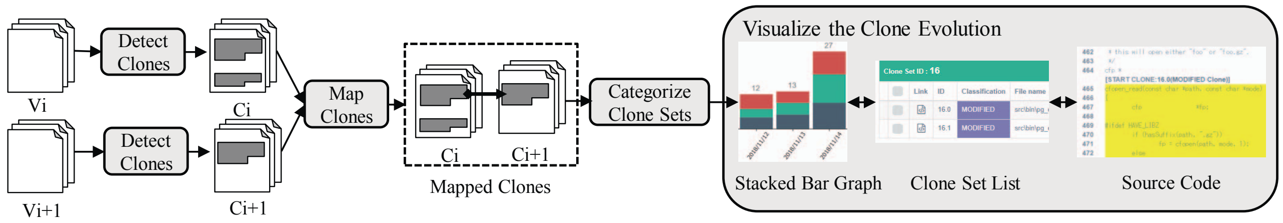


Fig. 1: An architecture overview of CCEvovis

this case, c_i and c_{i+1} are mapped by counting and inserted lines in c_{i+1}. Thus, the start line number of c_{i+1} is the same as c_i, and the end line number of c_{i+1} is increased by three lines. In the the *parent-child relationship*, a *child clone* of c_i is c_{i+1} and a *parent clone* of c_{i+1} is c_i.

C. Categorize Clone Sets

All clone sets are categorized based on change patterns of them between two versions v_i and v_{i+1} of a program. To classify clone sets, CCEvovis also uses the same categories that were used in the Clone Notifier.

- *Stable clone set:* Stable clone set contains code clone c_{i+1} that satisfies the following conditions: (1) parent clone of c_{i+1} exists in v_i and (2) c_{i+1} is unmodified until v_{i+1}. These clone sets contain clones that involved in v_i and v_{i+1} respectively.
- *New clone set:* New clone set contains code clones c_{i+1} whose parent clones do not exist in v_i. These clone sets involved in only v_{i+1}.
- *Deleted clone set:* Deleted clone set contains clones c_i whose child clones do not exist in v_{i+1}. These clone sets involved in only v_i.
- *Changed clone set:* Changed clone set contain a code clone that satisfies one of the following conditions: (1) parent clone of c_{i+1} exists in v_i and c_{i+1} is unmodified until v_{i+1}, (2) code clones c_{i+1} whose parent clones do not exist in v_i, and (3) clones c_i whose child clones do not exist in v_{i+1} These clone sets contain changed, deleted and newly added clones that involved in v_i and v_{i+1} respectively.

D. Visualize the Clone Evolution

Visualization of the clone evolution may be an effective aid to clone management. To support the maintenance of code clones, CCEvovis provides and visualizes different information of the clone evolution with three pages: 1) Stacked Bar Graph page, 2) Clone Set List page, and 3) Source Code page. Fig.2 depicts the visualization of the clone evolution of PostgreSQL[2] from these three pages.

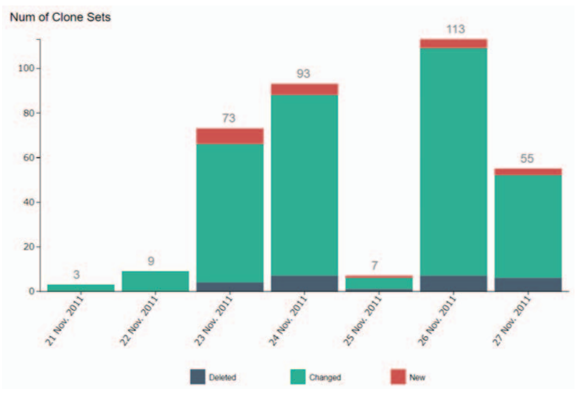
Stacked Bar Graph page: The goal of the Stacked Bar Graph page is to visualize the evolution of the clone sets in different categories, explained in Section II-C. There are two advantages of using the Stacked Bar Graph in CCEvovis. First, a user can intuitively grasp the size, ratio, and tendency of each clone set. Second, he/she is able to easily compare each clone set across multiple versions.

[2]https://www.postgresql.org/

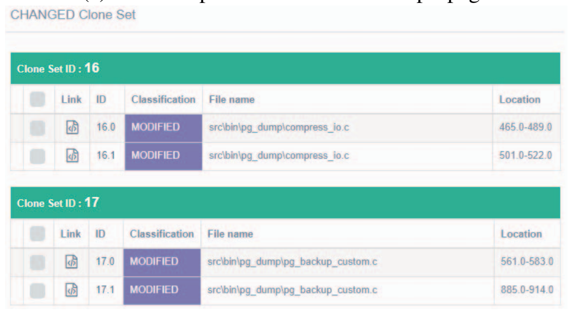
An example of the Stacked Bar Graph page is shown in Fig.2 (a). As can be seen from this figure, the horizontal axis of the Stacked Bar Graph shows each date of the version. The vertical axis of the Stacked Bar Graph presents the number of each clone sets in the different categories. That is, New clone set is shown with a red bar, Changed clone set is shown with a green bar, and Deleted clone sets is shown with a dark blue bar. The height of the Stacked Bar Graph corresponds to the sum of the number of New, Changed, and Deleted clone sets. Meanwhile, CCEvovis does not provide the information on Stable clone sets in the Stacked Bar Graph page. There are two reasons for this: first, the need for providing the information of the Stable clone set is low. Generally, the Stable clone sets are not considered as a target for the management. The second, it avoids unnecessary effort for distinguishing the important information to manage code clones in a short-term analysis. We assume that most of the users adopt CCEvovis to analysis the evolution of code clones within a relatively short time period such as a day or week. In this case, most code clones are not changed [7]. Therefore, most of the clone sets might be classified as Stable clone sets. If Stacked Bar Graph page provides information on Stable clone sets, it is difficult for a user to acquire important information for managing code clones such as New, Changed, and Deleted clone sets. By clicking a Stacked Bar Graph, a user can move to the corresponding Clone Set List page.

Clone Set List page: The goal of the Clone Set List page is to present a list of clone sets in different categories (i.e., New, Changed, Deleted, and Stable clone sets) between two selected versions. Note that this page provides a list of Stable clone sets because a user might want to confirm the list of Stable clone sets for his/her task. These clone sets are displayed with the following order: New, Changed, Deleted, and Stable clone sets. In this page, new clone sets are displayed first, because we assume that these clone sets should be preferentially managed.

An example of the Clone Set List page is shown in Fig.2 (b). As can be seen in this figure, the Clone Set List page displays each clone set with a list of code clones that are belonging to this clone set. This page also shows the assigned ID, evolution pattern, the name file where the code clone is located, and the position of the code clone in the file are displayed for each



(a) An example of Stacked Bar Graph page



CHANGED Clone Set

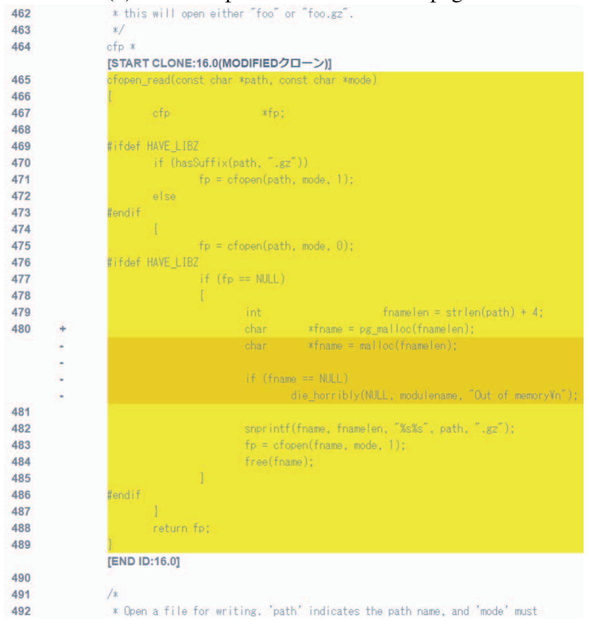
Clone Set ID : 16

	Link	ID	Classification	File name	Location
		16.0	MODIFIED	src\bin\pg_dump\compress_io.c	465.0-489.0
		16.1	MODIFIED	src\bin\pg_dump\compress_io.c	501.0-522.0

Clone Set ID : 17

	Link	ID	Classification	File name	Location
		17.0	MODIFIED	src\bin\pg_dump\pg_backup_custom.c	561.0-583.0
		17.1	MODIFIED	src\bin\pg_dump\pg_backup_custom.c	885.0-914.0

(b) An example of Clone Set List page



```
       * this will open either "foo" or "foo.gz".
       */
      cfp *
      [START CLONE:16.0(MODIFIEDクローン)]
      cfopen_read(const char *path, const char *mode)
      {
              cfp                 *fp;

      #ifdef HAVE_LIBZ
              if (hasSuffix(path, ".gz"))
                      fp = cfopen(path, mode, 1);
              else
      #endif
              {
                      fp = cfopen(path, mode, 0);
      #ifdef HAVE_LIBZ
                      if (fp == NULL)
                      {
                              int                        fnamelen = strlen(path) + 4;
  +                           char        *fname = pg_malloc(fnamelen);
  -                           char        *fname = malloc(fnamelen);
  -
  -                           if (fname == NULL)
  -                                   die_horribly(NULL, modulename, "Out of memory¥n");

                              snprintf(fname, fnamelen, "%s%s", path, ".gz");
                              fp = cfopen(fname, mode, 1);
                              free(fname);
                      }
      #endif
              }
              return fp;
      }
      [END ID:16.0]

      /*
       * Open a file for writing. 'path' indicates the path name, and 'mode' must
```

(c) An example of Source Code page

Fig. 2: Web UI for visualizing the clone evolution

code clone. A user can move the Source Code page by clicking the source code icon.

Source Code page: The goal of the Source code page is to present source code of the code clones that are involved in the selected clone set. An example of the Source Code page is shown in Fig.2 (c). In this page, the source code of each code clone is highlighted with the yellow background color. "+" indicates an added line in the new version, and "-" indicates a deleted line in the new version.

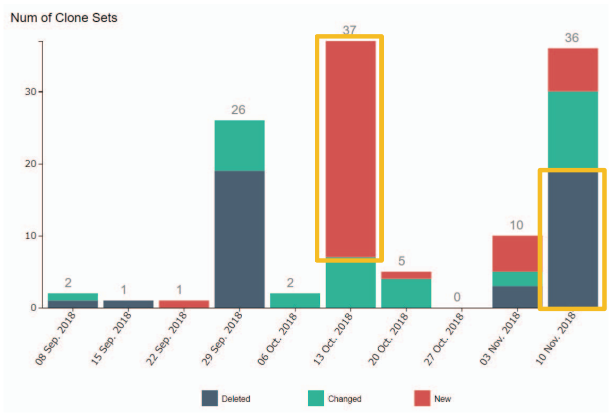


Fig. 3: An overview of the Stacked Bar Graph page

III. Illustrative Usage Scenario

In this section, we demonstrate the usage of CCEvovis for visualizing the evolution of code clones to support the maintenance of code clones. In particular, we provide illustrative of a usage scenario with respect to the following code clone maintenance tasks:

1) *Identify targets for clone refactoring*: One scenario for the use of CCEvovis is that a user can identify targets for clone refactoring. Generally, it is difficult to identify targets for clone refactoring among detected code clones.
2) *Confirm refactored code:* Another group of tasks is that confirming the refactored code clones. It is important for developers to confirm the refactored code clones because inconsistent changes to code clones introduce software defects.

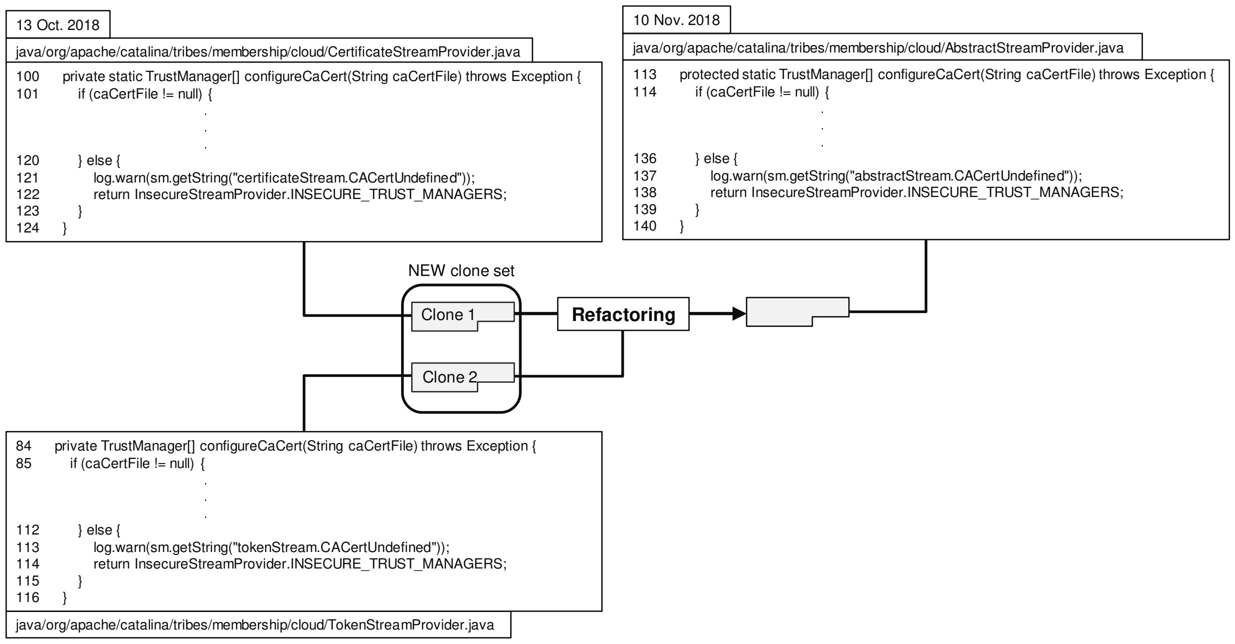
To accomplish these tasks, the developer uses CCEvovis with the ten versions of the Apache Tomcat[3], an open source implementation of the Java Servlet, JavaServer Pages, Java Expression Language, and Java WebSocket technologies. The code clones were analyzed within the time-frame from September 8, 2018, to November 10, 2018, with a one-week interval. To detect code clones from these versions, the developer selects SourcererCC as a code clone detection tool.

Fig.3 depicts the Stacked Bar Graph page of the Apache Tomcat. From this figure, the developer can confirm the following:

(A) A New clone set (shown in a red bar) appeared on October 13, 2018
(B) A Deleted clone set (shown in a dark blue bar) appeared on November 10, 2018

With this information, the developer can identify a target for clone refactoring. In order to refactor this clone set, the

[3]http://tomcat.apache.org/


13 Oct. 2018
java/org/apache/catalina/tribes/membership/cloud/CertificateStreamProvider.java
10 Nov. 2018
java/org/apache/catalina/tribes/membership/cloud/AbstractStreamProvider.java
NEW clone set
Clone 1
Clone 2
Refactoring
java/org/apache/catalina/tribes/membership/cloud/TokenStreamProvider.java

```
private static TrustManager[] configureCaCert(String caCertFile) throws Exception {
    if (caCertFile != null) {
                    .
                    .
                    .
    } else {
        log.warn(sm.getString("certificateStream.CACertUndefined"));
        return InsecureStreamProvider.INSECURE_TRUST_MANAGERS;
    }
}
```

```
protected static TrustManager[] configureCaCert(String caCertFile) throws Exception {
    if (caCertFile != null) {
                    .
                    .
                    .
    } else {
        log.warn(sm.getString("abstractStream.CACertUndefined"));
        return InsecureStreamProvider.INSECURE_TRUST_MANAGERS;
    }
}
```

```
private TrustManager[] configureCaCert(String caCertFile) throws Exception {
    if (caCertFile != null) {
                    .
                    .
                    .
    } else {
        log.warn(sm.getString("tokenStream.CACertUndefined"));
        return InsecureStreamProvider.INSECURE_TRUST_MANAGERS;
    }
}
```

Fig. 4: An example of the NEW code set

developer can determine whether or not refactor this clone set by clicking the New clone set appeared on October 13, 2018, and then confirming details of each code clone contained in this clone set and its source code in the Clone Set List and Source Code pages.

Moreover, the developer can confirm refactored code by clicking the Deleted clone set that appeared on November 10, 2018. He/she can check the details of each clone contained in the Deleted clone set and its source code in the Clone Set List and Source Code pages. This means that the developer is able to confirm that the refactored clone sets with CCEvovis.

To better understand this phenomenon, we analyze the details of these clone sets and confirm that the New clone set appeared on October 13, 2018, was refactored on November 10, 2018. As a result, the New clone set appeared as the Deleted clone set on November 10, 2018. Fig.4 depicts the detailed of information of these clone sets. In this figure, cloned methods named *configureCaCert* in the *CertificateStreamProvider* class and *TokenStreamProvider* class were newly appeared in October 13, 2019 and then refactored in November 10, 2018.

CCEvovis can provide information on the New clone set to be refactored. In addition, the developer can confirm the completion of refactoring of the New clone set with information on the Deleted clone set. These suggest that CCEvovis is able to support identifying a target for clone refactoring and confirming the refactored clone.

IV. Conclusion and Future work

This paper presents CCEvovis, a system for visualizing the evolution of code clones to support code clone maintenance. CCEvovis detects code clones across multiple versions of a program and provides and visualizes and highlights the evolution of code clones based on their evolution patterns with three main pages. CCEvovis supports the maintenance of code clones. In particular, CCEvovis is able to support identifying a target for clone refactoring and confirming the refactored clone.

As a part of future work, we plan to extend CCEvovis to be integrated into the IDE for helping developers maintain code clones in time. We are also interested in improving the visualization of clone evolution by allowing developers to hide/filter out certain clone categories of clone sets.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Numbers JP25220003, JP18H04094 and JP16K16034.

References

[1] C. K. Roy, M. F. Zibran, and R. Koschke, "The vision of software clone management: Past, present, and future (keynote paper)," in *Proc. of CSMR-WCRE*, 2014, pp. 18–33.
[2] M. Fowler, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.
[3] Y. Yamanaka, E. Choi, N. Yoshida, K. Inoue, and T. Sano, "Applying clone change notification system into an industrial development process," in *Prof. ICPC*, May 2013, pp. 199–206.
[4] Y. Yamanaka, E. Choi, N. Yoshida, and K. Inoue, "A high speed function clone detection based on information retrieval technique," *IPSJ Journal*, vol. 55, no. 10, pp. 2245–2255, 2014, in Japanese.
[5] K. Yokoi, E. Choi, N. Yoshida, and K. Inoue, "Fine-grained block clone detection based on information retrieval techniques," *JSSST journal Computer Software*, vol. 35, no. 4, pp. 16–36, 2018, in Japanese.
[6] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "SourcererCC: Scaling code clone detection to big–code," in *Proc. of ICSE*, 2016, pp. 1157–1168.
[7] J. Krinke, "Is Cloned Code more Stable than Non-cloned Code?" in *Proc. SCAM*, 2008, pp. 57–66.