CLIONE: Clone Modification Support for Pull Request Based Development

Tasuku Nakagawa, Yoshiki Higo, and Shinji Kusumoto
Graduate School of Information Science and Technology
Osaka University
1-5 Yamadaoka, Suita, Osaka, Japan
{t-nakagw, higo, kusumoto}@ist.osaka-u.ac.jp

***Abstract*—A code clone (clone) is known as one of the factors that makes software maintenance difficult. Thus, in software maintenance, clone modification is essential. An existing study proposed a tool that notifies developers of information about clone changes so that the developers can modify clones efficiently. However, the existing tool is premised on regular execution and not designed to be triggered by external factors except for time. Hence, the existing tool is difficult to be executed triggered by development workflow, such as modifying source code or merging branches , and we think this causes some issues. Consequently, in this study, we propose a new clone modification support technique aimed to integrate into pull request (PR) based development for solving those issues. The proposed technique detects code fragments that need modifications by tracking clones at the time of creating PRs. Moreover, we made three improvements for more accurate clone change tracking. Additionally, we implemented the proposed technique as a software tool, CLIONE. To evaluate CLIONE, we investigated the proportion of PRs in which clones have been modified non-simultaneously, and also we compared the results of clone change tracking with the existing tool. As a result, 11.9%$\sim$30.4% of PRs included non-simultaneously modified clones, and we confirmed that CLIONE was able to track clone changes more accurately than the existing tool. CLIONE is available at https://github.com/T45K/CLIONE.**

***Index Terms*—Code Clone, Software Maintenance, Pull Request Based Development**

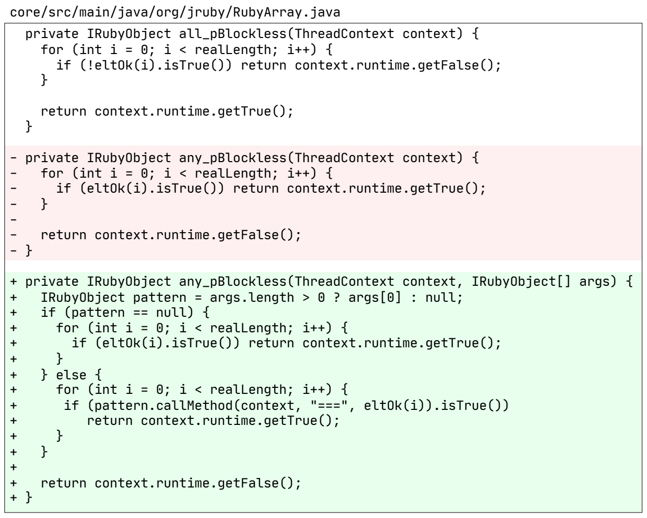
I. Introduction

A code clone (in short, clone) is a code fragment that is identical or similar to other code fragments in source code. Code cloning has been pointed out as one of the major problems in the maintenance process of software development [1]. Thus, there are many studies on simultaneous modification and refactoring support for clones [2], [3].

An existing study proposed a clone change notification tool, Clone Notifier, to improve the efficiency of these tasks [4]. Clone Notifier takes two revisions of a target project as inputs and notifies developers of information about clone changes. Clone Notifier is designed for industry use [5] and is premised on one execution a day. On the other hand, Clone Notifier is difficult to be executed triggered by development workflow, such as modifying source code or merging branches because it is not designed to be triggered by external factors except for time. For this reason, we think there are the following issues when considering the use of Clone Notifier in development, such as OSS development, in which modifying source code or merging branches are frequently conducted [6].

- The first issue is that even if code fragments need modifications as a result of clone changes, the code fragments will not be notified promptly. Because Clone Notifier is executed once a day, once Clone Notifier has been executed, there is a day interval until the next execution. On the other hand, if Clone Notifier users set its execution interval shorter (e.g., five minutes) to solve this issue, information of clone changes is notified to the users frequently even if the source code is not modified. Hence, the notification probably annoys the users. If clone change notification is triggered by development workflow, code fragments to be modified will be notified in line with the development workflow, and the developers will be able to respond to the code fragments promptly.
- The second issue is that if a large amount of source code modifications are conducted, and as a result, many clones may be changed improperly (i.e., many code fragments need modifications), the information is notified all at once. In this case, the users will be forced to check a large amount of clone change information, and they may overlook serious clone changes. If clone change notification is triggered by development workflow, code fragments to be modified are notified in line with each source code modification, and the developers will be able to avoid overlooking serious clone changes.
- The third issue is that code fragments to be modified are probably merged into the main branch when developing with version control systems (in short, VCS). In such development, the main branch should be bug-free and always ready for release. Therefore, code fragments to be modified that may contain bugs should be checked before the topic branch is merged. However, Clone Notifier, which is premised on regular execution, is difficult to detect such code fragments before merging topic branches. If clone change notification is triggered by development workflow, the possibility of bugs merged into the main branch can be reduced because such code fragments are detected before merging topic branches.

Consequently, in this study, we propose a new clone modification support technique aimed to integrate into pull request based development for solving these issues. The proposed technique detects code fragments that need modifications by



```
core/src/main/java/org/jruby/RubyArray.java
  private IRubyObject all_pBlockless(ThreadContext context) {
    for (int i = 0; i < realLength; i++) {
      if (!eltOk(i).isTrue()) return context.runtime.getFalse();
    }

    return context.runtime.getTrue();
  }

- private IRubyObject any_pBlockless(ThreadContext context) {
-   for (int i = 0; i < realLength; i++) {
-     if (eltOk(i).isTrue()) return context.runtime.getTrue();
-   }
-
-   return context.runtime.getFalse();
- }

+ private IRubyObject any_pBlockless(ThreadContext context, IRubyObject[] args) {
+   IRubyObject pattern = args.length > 0 ? args[0] : null;
+   if (pattern == null) {
+     for (int i = 0; i < realLength; i++) {
+       if (eltOk(i).isTrue()) return context.runtime.getTrue();
+     }
+   } else {
+     for (int i = 0; i < realLength; i++) {
+      if (pattern.callMethod(context, "===", eltOk(i)).isTrue())
+         return context.runtime.getTrue();
+     }
+   }
+
+   return context.runtime.getFalse();
+ }
```

Fig. 1: A non-simultaneous modification in JRuby

tracking clone changes at the time of creating pull requests (in short, PR). Moreover, we made three improvements for more accurate clone change tracking than Clone Notifier. Additionally, we implemented the proposed technique as a software tool, CLIONE[1].

To evaluate CLIONE, we conducted two experiments, an investigation of the proportion of PRs in which clones have been modified non-simultaneously (i.e., some of clones were modified but the others were not modified) and a comparison of the results of CLIONE's clone change tracking with the Clone Notifier's one. As a result, 11.9%∼30.4% of PRs are in which clones were modified non-simultaneously, and we confirmed that CLIONE was able to track clone changes more accurately than Clone Notifier.

II. Research Motivation

Figure 1 shows two methods in JRuby. In PR#5096[2], while any_pBlockless method was modified, all_pBlockless method was not modified. This non-simultaneous modification introduced a bug that caused an error when calling a Ruby API, which internally calls all_pBlockless method in JRuby into the main branch. As a result, JRuby, which included this bug, was released at version 9.2.0.0. This bug was fixed in PR#5298[3]. If the developers used CLIONE, they could avoid introducing the bug into the main branch. Besides, Clone Notifier cannot find this non-simultaneous modification because any_pBlockless method was entirely changed, and the clone detectors used in Clone Notifier cannot detect the modified method as a clone of all_pBlockless method (detail is explained in Section IV-B).

[1]available at https://github.com/T45K/CLIONE
[2]https://github.com/jruby/jruby/pull/5096
[3]https://github.com/jruby/jruby/pull/5298

III. Preliminaries

A. Code clone

A *clone* is a code fragment that is identical or similar to another code fragment. A set of code fragments in which any pair is a pair of clones is called *clone set*.

Clones are classified as follows based on the degree of similarity to their correspondences [7].

- **Type-1** is an exact copy without modifications (except for white spaces and comments).
- **Type-2** is a syntactically identical copy; only variables, types, or function identifiers were changed.
- **Type-3** is a copy with further modifications; statements were changed, added, or removed.

So far, many clone detection techniques and tools have been proposed and developed [8]. In this paper, we introduce a couple of popular detection techniques that are especially related to this research.

Text-based clone detection

This technique firstly normalizes the target source code and detects clones by comparing code fragments, such as methods or code blocks in the normalized source code. NiCad [9], which detects Type-3 clones from normalized source code by Longest Common Subsequence algorithm, is a well-known text-based clone detection tool.

Token-based clone detection

This technique firstly tokenizes the target source code and detects clones by using the token information. CCFinder [10] and SourcererCC [11] are well-known token-based clone detection tools. CCFinder detects Type-2 clones by comparing token sequences. SourcererCC detects Type-3 clones by using token types and their appearance frequencies.

B. Clone Notifier

Tokui et al. developed a tool, Clone Notifier, which supports developers to modify clones [4]. Clone Notifier takes two revisions of a project as inputs, and based on the changes of the source code between the two revisions, it classifies clone sets under four categories: *Stable*, *New*, *Deleted*, and *Changed*. First, Clone Notifier detects clones from each revision by using existing clone detectors [11]–[13]. Next, it tracks changes of the detected clones. Finally, it classifies clone sets from the results of clone change tracking. Clone Notifier is designed to be premised on regular execution (e.g., once a day).

C. Pull request based development

PR-based development is a development process using PR, which is one of the features of GitHub. PR is a feature that notifies developers of development information, such as source code changes before a branch is merged into another one. In PR-based development, the developers work on not the main branch but topic branches. Once the assigned work is done, the developer creates a PR and notify other developers of the

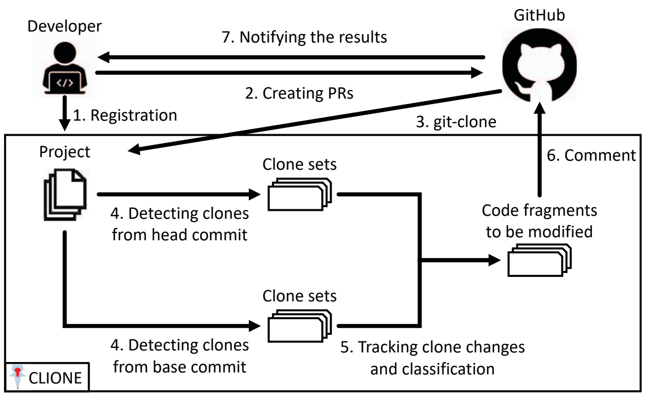

Developer
GitHub
7. Notifying the results
2. Creating PRs
1. Registration
3. git-clone
Project
Clone sets
6. Comment
4. Detecting clones from head commit
Code fragments to be modified
Clone sets
4. Detecting clones from base commit
5. Tracking clone changes and classification
CLIONE

Fig. 2: An overview of CLIONE

change information, and if the changes have no problems, the topic branch is merged to the main branch.

PR-based development is widely adopted by OSS development [14]. Moreover, some studies have proposed techniques that integrate existing source code analysis techniques into PR based development for more efficient software development [15], [16].

IV. Proposed Tool: CLIONE

In this study, we propose a new clone modification support technique aimed to integrate into development workflow. The proposed technique detects code fragments that need modifications by tracking clone changes at the time of creating PRs. We implemented the proposed technique as a tool, CLIONE. CLIONE is a server-side application and receives HTTP requests which GitHub sends when developers create PRs. Additionally, CLIONE is implemented based on GitHub Apps[4], so that it is easy to install CLIONE on GitHub projects.

A. Overview

Figure 2 shows an overview of CLIONE. First, developers who want to use CLIONE register their GitHub accounts and their repositories with CLIONE. After this registration, the developers can use CLIONE by creating PRs. CLIONE tracks clone changes between the head commit of the PR and the commit at the point where the PR branch was created (in short, base commit).

When the developer creates a PR, CLIONE downloads (git-clone) the project onto its local environment. Next, CLIONE detects clones from the head and the base commits of the PR, respectively, by using NiCad [9] or SourcererCC [11]. Why we select these clone detectors is each of them can detect Type-3 clones precisely.

Next, CLIONE tracks clones to detect clone changes between the two commits. As well as Clone Notifier, to track clones between the two commits, CLIONE calculates the overlapping location of clones, based on the location overlapping function of Kim et al. [17]. Location overlapping measures how much two code fragments $cf1$ and $cf2$ overlap each other ($0 \leq LO(cf1, cf2) \leq 1$). CLIONE uses the difference between the same file in each commit, without the added and deleted lines. It computes the relative proportion of an overlapped region between $cf1$ and the calibrated $cf2$.

$$LO(cf1, cf2) = \frac{min(n_e, o_e) - max(n_s, o_s)}{n_e - n_s} \quad (1)$$

where $cf1$ in the base commit spans from the line o_s to the line o_e, and the calibrated location of $cf2$ in the head commit spans from the line n_s to the line n_e. If the location overlapping between the two clones is 30% or more, CLIONE tracks from the clone at the base commit to the clone at the head commit.

After clone tracking, based on changes of clones, CLIONE detects non-simultaneously modified clone sets and clone sets to be refactored as code fragments that need modifications. Specifically, CLIONE treats clone sets where some clones were modified but the others were not modified as non-simultaneously modified and clone sets where some or all of clones were newly added as targets of refactoring. Moreover, in CLIONE, we made three improvements for more accurate clone change tracking. We explain the improvements in Section IV-B.

Finally, CLIONE makes a PR comment for each the code fragments to be modified to notify the results to the developer. In this comment, the code fragments are shown. In the case of non-simultaneously modified clones, changed code fragments and non-changed code fragments are shown. In the case of clones to be refactored, all of the target code fragments are shown. Figure 3 shows an example of PR comment about a non-simultaneously modified clone set. In the top of the figure, the changed piece of the changed code fragment is shown in the diff format, the whole of changed code fragment is shown in the middle, and the non-changed code fragment is shown in the bottom.

By getting notifications of the results as PR comments, developers can receive feedback promptly.

B. Improvements of clone change tracking

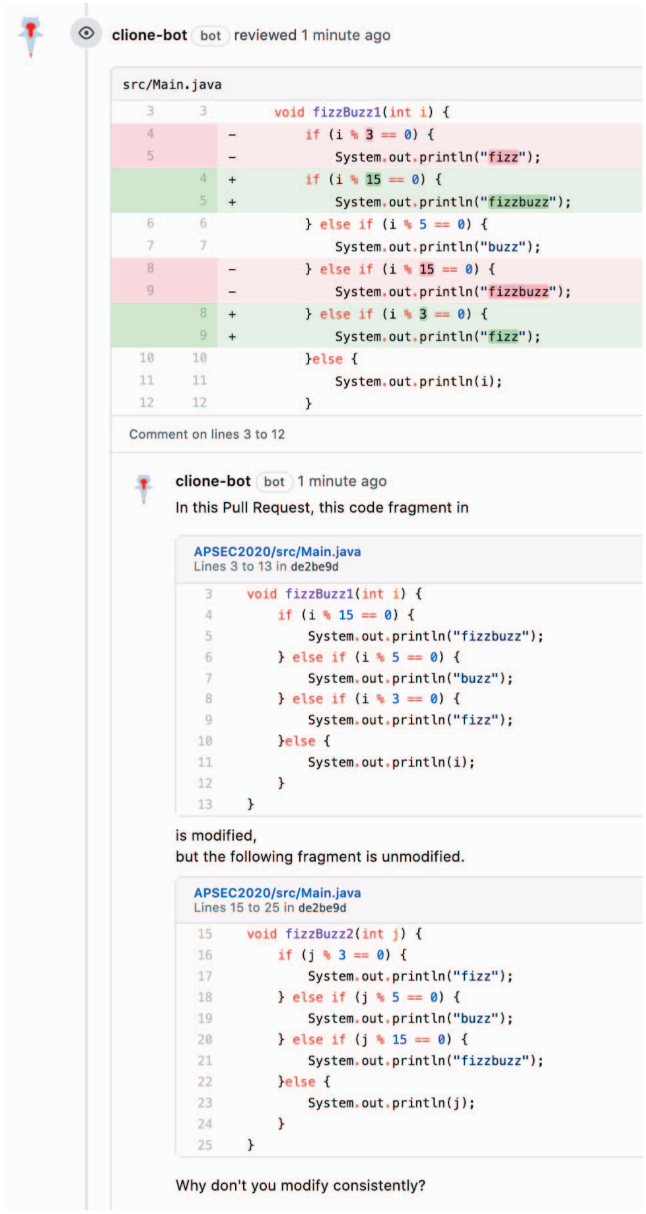
In CLIONE, we made three improvements to the clone change tracking technique used in Clone Notifier,

1) code fragment tracking,
2) file rename detection, and
3) clone change judgment by comparing token sequences.

In this paper, we explain only 1) code fragment tracking because of space limitations.

Clone Notifier treats only clone instances (code fragments that are detected as clones) as targets of tracking. Thus, if code fragments that are detected as clones in one of the commits are not detected in the other commit, the code fragments are classified added or deleted clone instances by Clone Notifier. On the other hand, in some cases, a code fragment is not detected as a clone because it has been modified. For example, figure 4(a) shows two methods that are detected as clones in the base commit, but not detected as clones in the head commit because one of them has been modified. In this case, Clone Notifier classifies the clone set (the two methods) as *Deleted*

[4]https://developer.github.com/apps/



clione-bot bot reviewed 1 minute ago

src/Main.java

```diff
       void fizzBuzz1(int i) {
-          if (i % 3 == 0) {
-              System.out.println("fizz");
+          if (i % 15 == 0) {
+              System.out.println("fizzbuzz");
           } else if (i % 5 == 0) {
               System.out.println("buzz");
-          } else if (i % 15 == 0) {
-              System.out.println("fizzbuzz");
+          } else if (i % 3 == 0) {
+              System.out.println("fizz");
           }else {
               System.out.println(i);
           }
```

Comment on lines 3 to 12

clione-bot bot 1 minute ago

In this Pull Request, this code fragment in

APSEC2020/src/Main.java
Lines 3 to 13 in de2be9d

```java
    void fizzBuzz1(int i) {
        if (i % 15 == 0) {
            System.out.println("fizzbuzz");
        } else if (i % 5 == 0) {
            System.out.println("buzz");
        } else if (i % 3 == 0) {
            System.out.println("fizz");
        }else {
            System.out.println(i);
        }
    }
```

is modified,
but the following fragment is unmodified.

APSEC2020/src/Main.java
Lines 15 to 25 in de2be9d

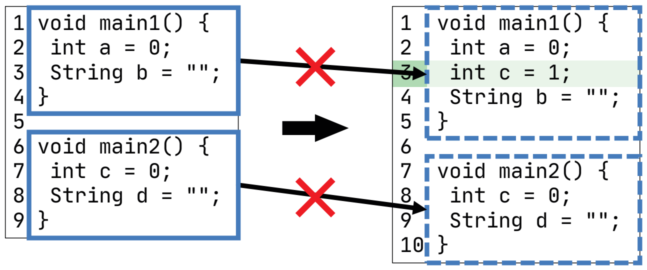
```java
    void fizzBuzz2(int j) {
        if (j % 3 == 0) {
            System.out.println("fizz");
        } else if (j % 5 == 0) {
            System.out.println("buzz");
        } else if (j % 15 == 0) {
            System.out.println("fizzbuzz");
        }else {
            System.out.println(j);
        }
    }
```

Why don't you modify consistently?

Fig. 3: An example of comment about
a non-simultaneously modified clone set

because the methods are not detected as clones in the head commit. However, this clone set should be classified as non-simultaneous modification because only main1 method was modified.

On the other hand, CLIONE tracks not only clone instances but also code fragments, such as methods or code blocks. Even if clones are not detected in one of the commits, clone changes are trackable by tracking code fragments themselves. In this example, CLIONE tracks the methods themselves (figure 4(b)). As a result, CLIONE can detect a non-simultaneously modified clone set (figure 4(c)) and classify it appropriately.



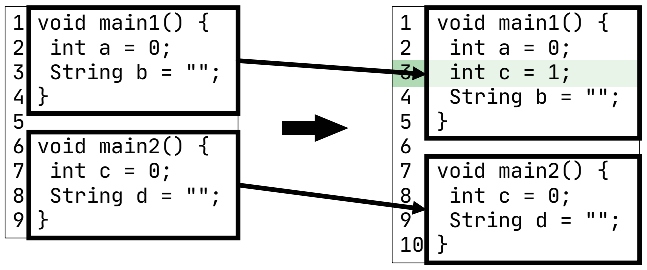
```java
void main1() {
 int a = 0;
 String b = "";
}

void main2() {
 int c = 0;
 String d = "";
}
```

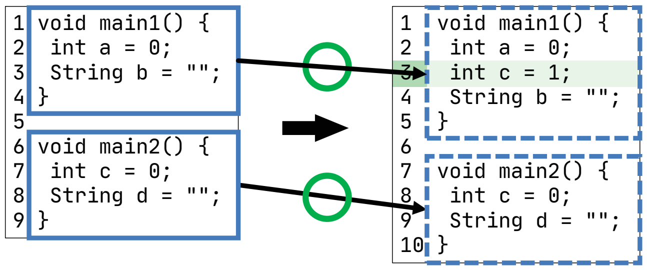
```java
void main1() {
 int a = 0;
 int c = 1;
 String b = "";
}

void main2() {
 int c = 0;
 String d = "";
}
```

(a) Clone instance tracking



(b) Code fragment tracking



(c) Accurate clone change tracking

Fig. 4: Improvement of clone tracking

V. Evaluation

To evaluate CLIONE, we conducted two experiments shown below.

Experiment 1

Experiment 1 is an investigation of the proportion of PRs in which clones have been modified non-simultaneously. In this experiment, we manually executed CLIONE on PRs of target projects and investigated whether clones had been modified simultaneously or not for each of the PRs.

Experiment 2

Experiment 2 is a comparison of the results of CLIONE's clone change tracking with Clone Notifier's one. In this experiment, we executed CLIONE and Clone Notifier on PRs of target projects and manually counted the number of PRs in which the results of clone change tracking of CLIONE and Clone Notifier are different due to code fragment tracking (explained in Section IV-B).

A. Experimental Targets

We selected three OSS as our experimental targets. Table I shows the names of the projects, the number of PRs merged by 20/7/2020, and the number of PRs in which at least a Java file was changed (in short, target PRs). The reason why we selected those OSS is that they are often targeted in clone research [18]–[20] and developed in PR-based development on GitHub.

B. Results

Results of experiment 1: Table II shows the results of experiment 1. In this table, "improper PRs" means PRs in which clones were modified non-simultaneously. All the projects have PRs in which clones were modified non-simultaneously, and their proportions are the range from 11.9% to 30.4%. We consider that if the developers of the projects had used CLIONE, they could have responded to non-simultaneously modified clones in PRs because the clones were notified at the time of creating PRs.

Results of experiment 2: As a result of Experiment 2, we confirmed that in total, there were 35 PRs (27.5% of improper PRs) in which CLIONE tracked clone changes more accurately than Clone Notifier. On the other hand, there was no clone change where Clone Notifier succeeded in tracking but CLIONE failed. Therefore, CLIONE is better at clone change tracking than Clone Notifier.

VI. Conclusion

In this study, we proposed a new clone modification support technique aimed to integrate into PR-based development. The proposed technique detects code fragments that need modifications at the time of creating PRs. Moreover, we implemented the proposed technique as a software tool, CLIONE. As evaluation, we investigated the proportion of PRs in which clones have been modified non-simultaneously, and also we compared the results of clone change tracking with the Clone Notifier. As a result, 11.9%∼30.4% of PRs included non-simultaneously modified clones, and we confirmed that CLIONE was able to track clone changes more accurately than Clone Notifier. Currently, CLIONE is available at https://github.com/T45K/CLIONE.

As future works, we consider evaluating usefulness of CLIONE for developers and measuring the response time of CLIONE's notification after a PR was created.

TABLE I: Target OSS

Name	# PRs	# Target PRs
jruby/jruby	2,248	292
junit-team/junit4	848	236
google/gson	317	69

TABLE II: Results of experiment 1

Name	# Target PRs	# Improper PRs	Proportion
jruby/jruby	292	89	30.4%
junit-team/junit4	236	28	11.9%
google/gson	60	10	16.7%
total	588	127	21.5%

Acknowledgment

This work was supported by MEXT/JSPS KAKENHI 20H04166.

References

[1] M. Mondal, C. K. Roy, and K. A. Schneider, "Bug propagation through code cloning: An empirical study," in *2017 IEEE International Conference on Software Maintenance and Evolution*, 2017, pp. 227–237.

[2] Y. Higo, Y. Ueda, S. Kusumoto, and K. Inoue, "Simultaneous Modification Support based on Code Clone Analysis," in *14th Asia-Pacific Software Engineering Conference*, 2007, pp. 262–269.

[3] Y. Higo, T. Kamiya, S. Kusumoto, and K. Inoue, "Refactoring Support Based on Code Clone Analysis," in *Product Focused Software Process Improvement*, 2004, pp. 220–233.

[4] S. Tokui, N. Yoshida, E. Choi, and K. Inoue, "Clone Notifier: Developing and Improving the System to Notify Changes of Code Clones," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering*, 2020, pp. 642–646.

[5] Y. Yamanaka, E. Choi, N. Yoshida, K. Inoue, and T. Sano, "Applying clone change notification system into an industrial development process," in *2013 21st International Conference on Program Comprehension*, 2013, pp. 199–206.

[6] J. Feller and B. Fitzgerald, *Understanding Open Source Software Development*. USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[7] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, "Comparison and Evaluation of Clone Detection Tools," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 577–591, 2007.

[8] A. Sheneamer and J. K. Kalita, "A Survey of Software Clone Detection Techniques," *International Journal of Computer Applications*, 2016.

[9] J. R. Cordy and C. K. Roy, "The NiCad Clone Detector," in *2011 IEEE 19th International Conference on Program Comprehension*, 2011, pp. 219–220.

[10] 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. 7, pp. 654–670, 2002.

[11] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "SourcererCC: Scaling Code Clone Detection to Big-Code," in *2016 IEEE/ACM 38th International Conference on Software Engineering*, 2016, pp. 1157–1168.

[12] "CCFinderX," http://www.ccfinder.net/.

[13] K. Yokoi, E. Choi, N. Yoshida, and K. Inoue, "Investigating Vector-Based Detection of Code Clones Using BigCloneBench," in *2018 25th Asia-Pacific Software Engineering Conference*, 2018, pp. 699–700.

[14] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu, "Wait for It: Determinants of Pull Request Evaluation Latency on GitHub," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015, pp. 367–371.

[15] V. Alizadeh, M. A. Ouali, M. Kessentini, and M. Chater, "RefBot: Intelligent Software Refactoring Bot," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering*, 2019, pp. 823–834.

[16] A. Carvalho, W. Luz, D. Marcílio, R. Bonifácio, G. Pinto, and E. Dias Canedo, "C-3PR: A Bot for Fixing Static Analysis Violations via Pull Requests," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering*, 2020, pp. 161–171.

[17] M. Kim, V. Sazawal, D. Notkin, and G. Murphy, "An Empirical Study of Code Clone Genealogies," in *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2005, p. 187–196.

[18] T. Nakagawa, Y. Higo, J. Matsumoto, and S. Kusumoto, "How Compact Will My System Be? A Fully-Automated Way to Calculate LoC Reduced by Clone Refactoring," in *2019 26th Asia-Pacific Software Engineering Conference*, 2019, pp. 284–291.

[19] C. Ragkhitwetsagul and J. Krinke, "Using compilation/decompilation to enhance clone detection," in *2017 IEEE 11th International Workshop on Software Clones*, 2017, pp. 1–7.

[20] K. Uemura, A. Mori, E. Choi, and H. Iida, "Tracking method-level clones and a case study," in *2019 IEEE 13th International Workshop on Software Clones*, 2019, pp. 27–33.