ISSN No. 0976-5697

Volume 7, No. 7, Nov-Dec 2016

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

An Improved Code Clone Tracking Approach in Evolving Software

Nadia Nahar
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh

Sujon Ali
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh

Md. Nurul Ahad Tawhid
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh

Abstract: Copying code fragments and then reusing those by pasting with or without modifications or adaptations are common activities in software development. This type of reuse approach of existing code is called code cloning and the pasted code fragment is called a clone of the original. Several studies show that about 5% to 20% of software systems can contain duplicated code, causing modification of all the fragments if a bug is detected in one code fragment. Although being a prime issue in software maintenance, refactoring of certain clones are not desirable and there is a risk of removing those. However, it is also widely agreed that clones should at least be detected and monitored across successive versions. Clone Region Descriptor (CRD) is an existing way for monitoring clones in evolving software systems. However, it has robustness issues for changes in code nesting level and block matcher parts called as anchors. This research works with overcoming these limitations by improving the CRDs through making those more abstract and adding more metrics for resolving block matching conflicts. The justification of the new approach is done by showing two case studies on the two proposed improvements.

Keywords: code clone; clone region descriptor; clone tracker; clone monitoring;

I. INTRODUCTION

Code clones are the duplicate or similar code blocks in the source code. These are mostly generated because of the habit of programmers to reuse code through copy & paste [1]. The presence of code clones in a system means that identical or similar logic in the code is not co-located [2]. Studies [7], [6] show that code clones occupy 7-23% of the code, and even in extreme case [8], can occupy 59% of the code. Code clones are considered to be an obstacle to software maintenance. Various approaches have been proposed for code clone detection and visualization [6], [9].

Normally, code clones considered harmful [10] for software system because, code cloning may duplicate faulty code regions, resulting in the multiple occurrence of same bug problem, where solved bugs seems to reappear as cloned code gets executed [4]. This increases the software maintenance work because bugs have to be resolved several times. Again presence of code clones means that duplicate or similar code is not co-located, which enforces to modify multiple sections of code reliably [5]. Otherwise clone regions may lead to regression faults. Some researchers advocated the removal of clones through source code refactoring [14]. For these reasons, many researches have done on the detection [10, 17, 18] and removal of code clones from software systems [19, 20]. However, it is not always possible to completely remove code clones due the associated risks [3, 11, 15]. Rajapakse and Jarzabek [13] show that co-locating code clones can be inefficient and difficult for understanding. Also refactoring code clones can increase running time [12]. In this situation, the code clone tracking needs to be done for monitoring the code clones in different versions of software, in order to maintain the consistency of the code.

There are many situations in which it may not be cost effective or even possible to refactor code clones. Kapser and Godfrey [21] showed several circumstances where code duplication seems to be a beneficial design option, like duplication in exploratory development or experimental changes to core subsystems. However, accepting the presence of code clones does not mitigate their caused problems. Thus, a technique is needed for documenting and monitoring clones in evolving software as well as for locating clone regions independent from specifications based on lines of source code, annotations, or other similarly fragile markers. A handful of tools to manage and support consistent modification to clone regions have been developed: such as, CReN [25], LAPIS [26], CodeLink [27]. However, tools like LAPIS and CReN characterize clone regions using character offsets or line ranges, and rely exclusively on the Integrated Development Environment (IDE) to update the location of the clone regions. So, modifications that change the line ranges of a clone region, or refactoring, such as pulling up a method to its superclass, may invalidate the clone relationships if performed outside the host environment. Similarly tools like CCFinder [10], NiCad [23], Simian [24] provide large information about the clones but those information is too much for users to use. To be effective, clone management techniques require a representation that is robust to evolutionary changes and applicable to both existing and future source code.

Clone genealogy is one method to help tracking and managing clones in evolving systems. Duala-Ekoko et al. [2] proposed an abstract Clone Region Descriptor (CRD) for tracking clones in evolving software systems. CRD describes the clone instances within methods in such a way that it is independent of the exact text or their location in the code. They developed an Eclipse plug-in named CloneTracker, which identifies the clone region in a code base for a given CRD through a series of automatic searches. Although CloneTracker showed better performance but it has some limitations like CRDs become invalid for changes in code nesting level or small changes in the CRDs anchor string. To overcome those limitations, we have proposed two improvements in CRDs: one improvement of block anchor and another improvement of CM. experimental result shows that the improved CRD was able to identify the clone regions in which the original clone tracker failed to track the clone.

The rest of this paper is organized as follows. Section 2 describes related work. Section 3 describes how the CloneTracker tools work and its limitations. Section 4 describes our proposed methods to overcome the limitations of the CloneTracker tool. Section 5 describes our study results. Section 6 discusses and summarizes our work.

II. RELATED WORK

A substantial amount of research has been dedicated to code cloning in recent years. Our approach is related to the following research fields:

A. Clone Detection

A huge amount of work exists on techniques to efficiently detect and analyze clones in source code [10, 17, 18]. In this approach, a clone detection tool takes the source code of a software system as input, pre-processes the text like breaking lines into tokens and removing nonessential differences, and finally performs a similarity analysis on the converted input. In CCFinder tool, Kamiya et al. provide a detailed description of this type of clone detection technology [10]. Simian, a text-based technique compares entire lines to each other, with little preprocessing of the text. Successive lines are then clustered to form larger clone regions. Abstract Syntax Tree (AST) based approaches, such as SimScan, transform the code into an AST, and execute a pairwise comparison of the nodes to identify similar subtrees. DECKARD transforms the code into a parse tree, and represents subtrees with numerical vectors. To detect clone, those numerical vectors are clustered using Euclidean distance, and subtrees having vectors in the same cluster are reported as clones.

B. Clone Management

Duala-Ekoko et al. [3] introduce a tool named CloneTracker that can provide the developers with the assists to track evolutionary clone groups. It uses CRD, a light-weight clone region descriptor to generate the correspondence between the clone groups in the successive versions. Another tool named JSync was introduced by Nguyen et al. [28] to detect the code clone and to make consistency changes when any change made in cloned code. Lin et al. [29] proposed an interactive approach to assist the developers to edit and modify copy & paste code. CCSync [30] reduces the harmfulness of editing code clones by synchronizing code clones whose structures are quite different.

C. Clone Genealogy

Kim et al. [31] present the clone genealogy, which associates related clone groups that have originated from the same ancestor clone group. In addition, the genealogy holds information about how each element in a clone group has changed with respect to other elements in the same group. They develop a Clone Genealogy Extractor (CGE), which integrates CCFinder clone detector, to get a software systems clone genealogy. Using CGE, they research how each clone region has evolved. They also find that some clones cannot be refactored, but CGE can help clone maintenance using clone genealogy information.

Saha et al. [32] extended the study by Kim. They develop a prototype tool named gCad and evaluate it on 17 open source systems in four different programming languages. Their findings is same with Kim that clones may not always be dangerous in software maintenance, and clone’s study need focus on tracking and managing clones in evolving system instead of refactoring them.

III. CLONETRACKER

Duala-Ekoko et. al. proposed a clone tracking approach to develop a technique for documenting and monitoring clones in evolving software [2]. The methodology and the outcome of the approach are described briefly in this section.

A. Methodology

First of all, a way to describe clone regions within methods was proposed, named as Clone Region Descriptor (CRD), which was independent from the exact text of the clone region or its location in a file. It was a lightweight and abstract description of the location of a clone region in a code base. CRD represented the clone regions by providing the top level information (file, class, method) along with nested block descriptions (block type - for, while, if, switch, try, catch, etc., anchors - terminating statement for loops, branching predicate for if, exception list for catch, etc.). Additionally, a Corroboration Metric (CM) was provided with each level for resolving conflicts (same CRDs for different code, which were not clone). Here, CM was calculated by simply adding the cyclomatic complexity of a block with the fan-out of the block.

A clone region lookup algorithm was proposed for searching the clone regions automatically in the code base. It relied on the Abstract Syntax Tree (AST) representation of the code. First the root AST node was identified by using the file, class, and method name in the CRD. Then the branches were traversed to find the leaf block. The conflicts were resolved by comparing CM. Finally, a simultaneous editing approach was proposed that relies on the clone region lookup algorithm with additional computation to map an individual line within a clone region to the corresponding line in another clone region. The line mapping technique was based on the Levenshtein Distance (LD). It compared the lines to have a similarity of at least a threshold simth with each other, for being mapped.

B. Implementation and Result

The approach was implemented as a plugin of eclipse named as CloneTracker [3]. For analyzing the efficiency of CloneTracker, it was applied on five open-source projects. A total of 1184 clone groups consisting of between 2 and 9 clone regions (inclusively), for a total of 3275 clone regions, was monitored using it. A large majority of clone regions (96%) were correctly tracked by the CRDs. 81% of the conflicts could be resolved by the CM. The simultaneous editing feature also has a high success rate of 80%.

C. Limitations

The identified limitations of the tool are –

- CRDs are invalidated for changes that simply remove a nesting level
- Associating else branches with the closest if prevents it from discriminating between the two types of blocks
- Storing anchors as strings implies that even small changes to the code in an anchor will invalidate the CRD. These limitations decrease the robustness of the CRDs.

IV. PROPOSED IMPROVEMENTS

From the above mentioned limitations, in this research, several improvements are proposed. Here, the string matching of anchors is focused for improvement. Also, the improvement of CM is another focus of this work. The existing accuracy of conflict resolution using CM is 81%. Thus, there is scope of improvement for the remaining 19% of the conflicts. The improvements are reported in details here.

A. Improvement-1: Improvement of block anchor

The blocks of CRDs are represented by block type, anchor and CM. The anchors are stored as strings, and matched to the source code blocks using native string match. This might work in case of matching with the source code of the same version as the generated CRDs. However, the anchors may not remain the same in the next versions. Even small changes in the anchors will affect the matching and the CRDs will be invalidated.

One of the most expected changes in the anchors is the refactoring of variable names. In the matching of the anchors, the variable names should not have any effect as these names do not have any significance in the code clones. Thus, as an improvement, a new field is introduced in describing the blocks, named as abstract anchor. These abstract anchors contain the anchors that are independent of the variable names. The field will come into action when the block anchors could not be matched with the source code due to changes in new version.

To be described from the technical point of view, the variable names are eliminated from the block anchors, and stored in abstract anchors. While looking up for the CRDs, at first the anchors are matched in code. For mismatch of the anchors, the abstract anchors are matched. Thus, this improvement provides an extra level of finding of the CRDs in code, letting those not to be invalidated for changes in variable names.

B. Improvement-2: Improvement of CM

In CloneTracker, CM is calculated by adding the Cyclomatic Complexity and the fan-out of the code block. Yet the CM of different code blocks might still be the same. This is the reason why 19% of the conflicts could not be resolved. Thus, for covering these blocks Halstead metric is added to the CM. The reason behind considering Halstead metric in spite of the other code metrics is described here.

The Cyclometic Complexity is the quantitative measure of the number of linearly independent paths in source code. It cannot measure the entropy of the program. Thus, Halstead metric is used to measure the program difficulty based on the number of operators and operands. The combination of Cyclometric Complexity, Halstead metric, and fan-out can measure the complete complexity of the program. This is why Halstead metric is added to the calculation of CM.

V. RESULT ANALYSIS

The approach of CloneTracker [2] was justified by using the tool on five open-source projects. Thus, those projects were also collected to justify the improved tool. Then the tool was evaluated on the projects. Also, the limitations of the tool were analyzed.

A. Environmental Setup

The tool is developed in Java programming language. The equipment used to develop the prototype are as follows:

- Eclipse Luna (4.4.1): java IDE for implementation
- ASTParser: static code analyzer
- Simian: clone detection tool

B. Data Collection

Different versions of the five mentioned projects in [2] have been collected. The projects and the collected versions are shown in Table I. Among these project versions, the versions that were used in [2] are JBossAOP 4.0, JEdit 4.0, FreeMind 0.8.0, Ant 1.6.5, JCommander 0.6.4. The exact versions of JEdit, FreeMind and Ant were found, and used as subject systems. However, for JBossAOP and JCommander, the mentioned version source codes were not found. Thus, for these two, version 2.1.8.GA and 1.3.4 have been used accordingly.

Table I. Experimented Projects

Sr. No.	*Project Name*	*Versions*
1	FreeMind	0.0.2 – 1.1.0
2	JBossAOP	2.0.0.alpha5 – 2.1.8.GA
3	JEdit	2.4pre1 – 5.2pre1
4	Ant	1.5.2 – 1.9.6
5	JCommander	1.1 – 1.48

C. Evaluation

For evaluating the improvements in the CloneTracker, two case studies are conducted for the two reported improvements. The case studies are described here.

1) *Case Study-1 (Improvement of block anchor):* For conducting the first case study, a clone group of the project JCommander (Table 1) is taken. The clone group contains two clone regions. The regions are –

- file: *`\jcommander–master\jcommander-jcommander1:34\src\main\java\com\beust\jcommander\JCommander:java,`*
  line: 1288 – 1292
- file: *`\jcommander–master\jcommander-jcommander1:34\src\main\java\com\beust\jcommander\Parameterized:java,`*
  line: 137 – 141

The CRDs of the clone regions are generated using both CloneTracker and the improved version of it. The generated CRDs of the 1st clone region by these two CloneTracker versions are –

CRD using CloneTracker
JCommander.java, JCommander, 665
convertValue(Parameterized parameterized, Class type, String value), 51
CATCH, IllegalAccessException e, 3
CATCH, InvocationTargetException e, 3

CRD using Improved CloneTracker
JCommander.java, JCommander, 825.4422604422605
convertValue(Parameterized parameterized, Class type, String value), 101.07345739471106
CATCH, IllegalAccessException e: 'IllegalAccessException e', 47.388910581803984
CATCH, InvocationTargetException e: 'InvocationTargetException e', 47.49260987942435

In these two CRDs, *IllegalAccessException e* and *InvocationTargetException e* are the anchors. And in the generated CRD of the improved CloneTracker, *IllegalAccessException* and *InvocationTargetException* are the abstract anchors. These CRDs were looked up in next version (1.35) of JCommander using the lookup algorithm. For testing the efficiency of the improved CloneTracker, the clone region of this version was slightly changed. The variable names of the anchor, e was changed to ex. The regained clone regions by the lookup algorithms of CloneTracker is –

- file: `\jcommander–master\jcommander-jcommander- 1:34\src\main\java\com\beust\jcommander\JCommander:java`, from line- 500000 to-0
- file: `\jcommander – master\jcommander - jcommander - 1:34\src\main\java\com\beust\jcommander\JCommander:java,` from line-1289 to-1294

Here, it can be seen that the regained region of CloneTracker failed to identify the line range, as it provides a fake line range of (500000-0). On the contrary, the improved CloneTracker successfully identified the line ranges (1289-1294).

2) *Case Study-2 (Improvement of CM):* In this case study, the efficiency of the new CM calculation is measured. For this measurement, the following code block is analyzed as shown in Figure 1.

Suppose, in the code block, a clone region is -

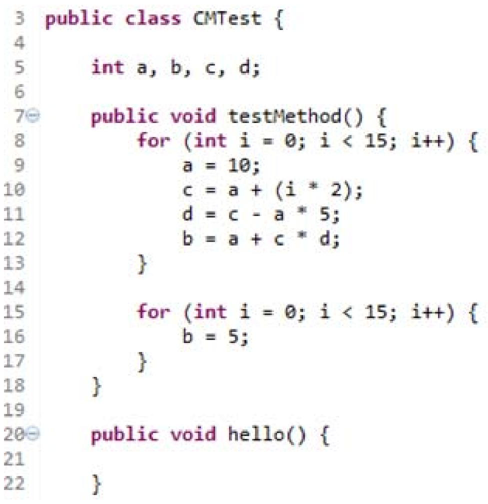
- file: `..\CMTest:java`, from line-8 to-14

The CRD of this clone region using CloneTracker is –

```
CMTest.java, CMTest, 4
testMethod(), 3
FOR, i <15, 1
```

The improved CRD of this clone region is –

```
CMTest.java, CMTest, 12.333333333333332
testMethod(), 8.14851485148515
FOR, i <15: '<15', 4.656716417910447
```



```
3   public class CMTest {
4   
5       int a, b, c, d;
6   
7       public void testMethod() {
8           for (int i = 0; i < 15; i++) {
9               a = 10;
10              c = a + (i * 2);
11              d = c - a * 5;
12              b = a + c * d;
13          }
14  
15          for (int i = 0; i < 15; i++) {
16              b = 5;
17          }
18      }
19  
20      public void hello() {
21  
22      }
```

Figure 1. Sample Code Block

Now, while looking up for the CRD in the code, a conflict is generated due to similar CRDs of region 8-14, and 15-17. The CRD of region 15-17 using CloneTracker is –

```
CMTest.java, CMTest, 4
testMethod(), 3
FOR, i <15, 1
```

Thus, it can be seen that the conflicts cannot be resolved due to the same CM values in these two CRDs (CRD of line range 8-14 and 15-17). However, the generated CRD of range 15-17 using the improved CloneTracker is –

```
CMTest.java, CMTest, 12.333333333333332
testMethod(), 8.14851485148515
FOR, i < 15: '< 15', 4.0
```

The CM (4.0) of this for block (15-17) in this CRD is different from the CM value (4.66) of the clone region for block (8-14). Thus, the conflict is resolved in the improved CloneTracker by using the new CM calculation.

VI. Conclusion

Cloned code in software systems creates extra work for developers and can increase the risk of introducing regression faults during software maintenance. However, eliminating certain clone groups from a software system is not always possible or practical.

Duala-Ekoko et. al. proposed a clone tracking method for monitoring important clone regions as a system evolves. They implemented a system, called CloneTracker that can automatically generate abstract representations for clone regions using Clone Region Descriptors (CRDs), which identify clone regions at the granularity of code blocks using the structural properties, lexical layout, and similarities of the clone region. However, the CRD definition has some limitations in detecting clone regions if the nesting levels are changed or anchor string changed. We have proposed remedy of those limitations for improving clone monitoring. The experimental result shows that the improved CRD can track those clone regions, in which the original CRD failed.

VII. References

[1] R. Koschke, R. Falke, and P. Frenzel, "Clone detection using abstract syntax suffix trees," In 2006 13th Working Conference on Reverse Engineering, pp. 253-262. IEEE, 2006.

[2] E. Duala-Ekoko, and M. P. Robillard, "Tracking code clones in evolving software," In 29th International Conference on Software Engineering (ICSE'07), pp. 158-167. IEEE, 2007.

[3] E. Duala-Ekoko, and M. P. Robillard, "Clonetracker: tool support for code clone management," In Proceedings of the 30th international conference on Software engineering, pp. 843-846. ACM, 2008.

[4] L. Aversano, L. Cerulo, and M. Di Penta, "How clones are maintained: An empirical study," In 11th European Conference on Software Maintenance and Reengineering (CSMR'07), pp. 81-90. IEEE, 2007.

[5] R. Geiger, B. Fluri, H. C. Gall, and M. Pinzger, "Relation of code clones and change couplings," In International Conference on Fundamental Approaches to Software Engineering, pp. 411-425. Springer Berlin Heidelberg, 2006.

[6] A. K. Kontogiannis, R. DeMori, E. Merlo, M. Galler, and M. Bernstein, "Pattern matching for clone and concept detection," In Reverse engineering, pp. 77-108. Springer US, 1996.

[7] B. Lague, D. Proulx, J. Mayrand, E. M. Merlo, and J. Hudepohl, "Assessing the benefits of incorporating

function clone detection in a development process," Proc. International Conference on Software Maintenance, IEEE Press, 1997, pp. 314-321.

[8] S. Ducasse, M. Rieger, and S. Demeyer, "A language independent approach for detecting duplicated code," Proc. IEEE International Conference on Software Maintenance (ICSM'99), IEEE Press 1999, pp. 109-118.

[9] Y. Zhang, H. A. Basit, S. Jarzabek, D. Anh, and M. Low, "Query-based filtering and graphical view generation for clone analysis." Proc. IEEE International Conference on Software Maintenance (ICSM 2008), IEEE Press 2008, pp. 376-385.

[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] G. Zhang, X. Peng, Z. Xing, and W. Zhao, "Cloning practices: Why developers clone and what can be changed," Proc. 28th IEEE International Conference on Software Maintenance (ICSM 2012), IEEE Press 2012, pp. 285-294.

[12] L. Aversano, L. Cerulo, and M. Di Penta, "How clones are maintained: An empirical study," Proc. 11th European Conference on Software Maintenance and Reengineering (CSMR'07), IEEE Press 2007, pp. 81-90.

[13] C. D. Rajapakse, and S. Jarzabek, "Using server pages to unify clones in web applications: A trade-off analysis," In 29th International Conference on Software Engineering (ICSE'07), IEEE Press 2007, pp. 116-126.

[14] M. Fowler, Refactoring: improving the design of existing code. Pearson Education India, 2009.

[15] M. Kim, L. Bergman, T. Lau, and D. Notkin, "An ethnographic study of copy and paste programming practices in OOPL," Proc. International Symposium on Empirical Software Engineering (ISESE'04), IEEE Press 2004, pp. 83-92.

[16] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, "Deckard: Scalable and accurate tree-based detection of code clones," Proc. 29th international conference on Software Engineering, IEEE Computer Society, 2007, pp. 96-105.

[17] A. H. Basit, and S. Jarzabek, "Efficient token based clone detection with flexible tokenization," Proc. 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, ACM 2007, pp. 513-516.

[18] R. Tairas, "Bibliography of code detection literature." (2008).

[19] R. Koschke, "Identifying and removing software clones." Software Evolution, Springer Berlin Heidelberg, pp. 15-36, 2008.

[20] C. Kapser, and M. W. Godfrey, ""Cloning considered harmful" considered harmful." Proc. 13th Working Conference on Reverse Engineering, IEEE Press 2006, pp. 19-28.

[21] J. R. Cordy, and C. K. Roy, "The NiCad clone detector," IEEE 19th International Conference on Program Comprehension (ICPC), IEEE Press 2011, pp. 219-220.

[22] M. S. Uddin, C. K. Roy, and K. A. Schneider, "Simcad: An extensible and faster clone detection tool for large scale software systems." Proc. 21st International Conference on Program Comprehension (ICPC), IEEE Press 2013, pp. 236-238.

[23] P. Jablonski and D. Hou, "CReN: a tool for tracking copy-and-paste code clones and renaming identifiers consistently in the IDE," Proc. OOPSLA workshop on eclipse technology eXchange, ACM 2007, pp. 16-20.

[24] R. C. Miller, and B. A. Myers, "Interactive Simultaneous Editing of Multiple Text Regions," In USENIX Annual Technical Conference, General Track 2001, pp. 161-174.

[25] M. Toomim, A. Begel, and S. L. Graham, "Managing duplicated code with linked editing," IEEE Symposium on Visual Languages and Human Centric Computing, 2004, pp. 173-180.

[26] 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. 38, no. 5, pp. 1008-1026, 2012.

[27] Y. Lin, X. Peng, Z. Xing, D. Zheng, and W. Zhao, "Clone-based and interactive recommendation for modifying pasted code." Proc. 10th Joint Meeting on Foundations of Software Engineering, ACM 2015, pp. 520-531.

[28] X. Wang, Y. Dang, L. Zhang, D. Zhang, E. Lan, and H. Mei, "Can I clone this piece of code here?," Proc. 27th IEEE/ACM International Conference on Automated Software Engineering, ACM 2012, pp. 170-179.

[29] M. Kim, V. Sazawal, D. Notkin, and G. Murphy, "An empirical study of code clone genealogies," ACM SIGSOFT Software Engineering Notes, vol. 30, no. 5, pp. 187-196.

[30] R. K. Saha, C. K. Roy, and K. A. Schneider, "gCad: A Near-Miss Clone Genealogy Extractor to Support Clone Evolution Analysis," ICSM 2013, pp. 488-491.