ClonePacker: A Tool for Clone Set Visualization

Hiroaki Murakami, Yoshiki Higo, Shinji Kusumoto
Graduate School of Information Science and Technology, Osaka University, Japan
{h-murakm, higo, kusumoto}@ist.osaka-u.ac.jp

***Abstract*—Programmers often copy and paste code fragments when they would like to reuse them. Although copy-and-paste operations enable programmers to realize rapid developments of software systems, it makes code clones. Some clones have negative impacts on software developments. For example, if we modify a code fragment, we have to check whether its clones need the same modification. In this case, programmers often use tools that take a code fragment as input and take its clones as output. However, when programmers use such existing tools, programmers have to open a number of source code and move up/down a scroll bar for browsing the detected clones. In order to reduce the cost of browsing the detected clones, we developed a tool that visualizes clones by using Circle Packing, named ClonePacker. As a result of an experiment with participants, we confirmed that participants using ClonePacker reported the locations of clones faster than an existing tool.**

Keywords*—*Software maintenance, Code clone, Visualization.

I. Introduction

A code clone (in short, clone) is a code fragment that has identical or similar code fragments in source code. It is generated by various reasons such as copy-and-paste operations. Recent research has revealed that some clones make software maintenances more difficult [1]. For example, if we modify a code fragment for fixing a bug or adding a new function, we have to check whether its clones need the same modification or not. In order to find clones of a given code fragment, some researchers have developed tools that took a code fragment as input and took its code clones as output. Libra [2] is one of such tools. Libra receives a code fragment from a user, and uses CCFinder [3] to detect clones of the input code fragment. Libra has two views when the user browses the detected clones. One is a tree view representing all files that are targeted for the clone detection, and the other is a source code view representing source code that is selected by the user. The detected clones are highlighted in the source code view. However, we consider that Libra has an issue. Programmers using Libra cannot browse detected clones efficiently because programmers have to open a number of source code and move up/down a scroll bar for browsing all detected clones. We consider that the source code of detected clones should be viewable easily. It is necessary to understand detected clones to a certain extent (e.g. which types of clones are detected?) without browsing source code.

In order to resolve this issue, we developed a clone set visualization tool, named ClonePacker. ClonePacker uses Circle Packing [4] for visualizing detected clones. We evaluated ClonePacker by comparing with Libra through an experiment with participants. In the experiment, the participants reported the locations of clones by using ClonePacker and Libra, then we compared the reporting time between the tools. Consequently, the contributions of this paper are followings.

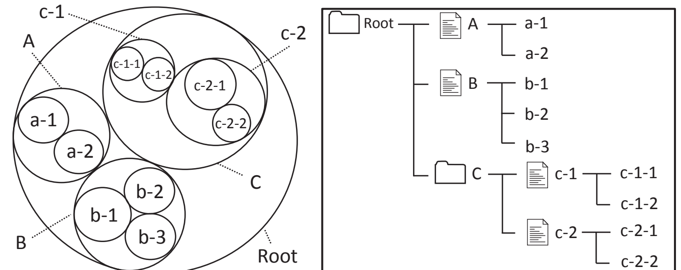

Fig. 1. Example of Circle Packing

- We proposed a technique to visualize detected clones and developed the proposed technique as a tool, named ClonePacker. Programmers using ClonePacker can understand detected clones to a certain extent without browsing source code.
- We confirmed that programmers using ClonePacker reported the locations of clones faster than Libra and the accuracy was unchanged.

The remainder of this paper is organized as follows: Section II describes types of clones and Circle Packing. Sections III and IV show the proposed technique and details of ClonePacker. Section V reports the evaluations of ClonePacker by comparing with Libra. Sections VI and VII describe some threats to validity and some related works. Finally, we conclude this paper in Section VIII.

II. Preparations

We explain the types of clones and Circle Packing.

A. Types of Clones

Bellon et al. categorized clones into the following three types [5].

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

We used above terms in this paper.

B. Circle Packing

Circle Packing is one of the enclosure diagrams. Figure 1 shows an example of Circle Packing. In the figure, Circle Packing represents three categories A, B and C. Each of the categories has some elements. For example, category A has two elements a-1 and a-2 and category C has two sub-categories,

978-1-4799-8468-8/15/$31.00 © 2015 IEEE

SANER 2015, Montréal, Canada

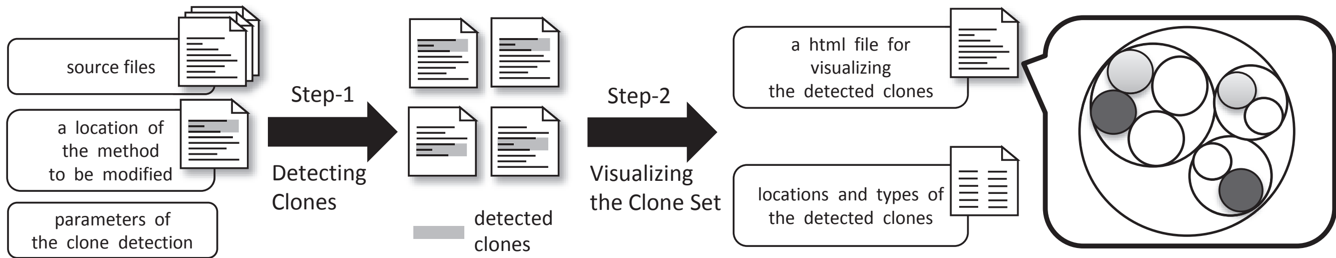

Fig. 2. Overview of the proposed technique

c-1 and c-2. Circle Packing is useful for representing hierarchical data structures. In Fig. 1, it is clear that elements a-1 and a-2 are in the same category. On the other hand, elements a-1 and b-1 are in different categories. Furthermore, both of elements c-1-1 and c-2-1 are in category C, however, they are in different sub-categories, c-1 and c-2.

We use Circle Packing for visualizing detected clones with file hierarchies. In this paper, we assume that the innermost circles represent methods, and the circles covering the innermost circles represent files. Moreover, the circles covering file circles represent directories. For example, in Fig. 1, the outermost circle represents directory Root. The directory contains two files A and B, and one directory C. File A has two methods a-1 and a-2. Directory C has two files c-1 and c-2, and file c-1 has two methods c-1-1 and c-1-2.

III. Proposed Technique

Figure 2 shows an overview of the proposed technique. The proposed technique consists of two steps.

Step-1: Detecting Clones
Step-2: Visualizing the Clone Set

First, users prepare a set of source files that is targeted for the clone detection (in short, target source files). Second, users specify a method to be modified (in short, a target method). Then, the proposed technique detects clones of the target method from the target source files. Lastly, detected clones are visualized by using Circle Packing.

The inputs of the proposed technique are followings:

- target source files,
- a file name and a start line of a target method, and
- parameters of the clone detection (*minimum token length* and *the number of allowed gapped statements*).

The outputs are followings:

- file names, start lines, end lines and types of the detected clones, and
- a html file for visualizing the detected clones with Circle Packing.

In the rest of this section, we describe each step.

A. Step-1: Detecting Clones

The proposed technique detects clones of the target method from the target source files by considering the input *minimum token length* and *the number of allowed gapped statements*. In this step, the proposed technique uses a version of customized our previous technique [6] to detect method clones[1]. The technique can detect all types of clones in a short time. The technique detects clones as a clone set[2].

B. Step-2: Visualizing the Clone Set

The proposed technique visualizes the clone set obtained in Step-1. The clone set is visualized as Circle Packing. Furthermore, locations and types of the detected clones are also reported.

IV. Tool: ClonePacker

A. Implementation

We have implemented the proposed technique as a tool, ClonePacker. ClonePacker has been developed as an Eclipse plugin. It is downloadable from our website[3]. We used JavaScript library D3[4] for visualizing the clone set. The proposed technique creates a html file representing the clone set. Then, the proposed technique visualizes the clone set by giving the html file to D3.

B. How to Use ClonePacker

Figure 5 shows a screenshot of ClonePacker at a startup time. First, users select a target method by setting a caret position on the method. In Fig. 5, the caret position exists at 116th line. In this case, method draw (111th - 129th lines) is selected as the target method. After the users push the button A, ClonePacker finds clones of the target method.

After ClonePacker finishes detecting clones, the users can see the detection results. Figure 6 shows a screenshot of ClonePacker for viewing the detected clones. The right view B shows the detected clones with Circle Packing. The yellow circle represents the target method that the users selected. The red one is Type-1 clone, the blue one is Type-2 clone and the green one is Type-3 clone. In this case, four Type-1 clones, three Type-2 clones and one Type-3 clone were detected. One of Type-1 clones locates in the same directory with the target method and the others locate in different directories. The size of each innermost circle represents LOC of the method. Location and type of each clone are showed in the bottom table by clicking each circle. The location of the clone is represented as a combination of its file path, its method name, its start line

[1]Method clones are methods that have identical or similar methods in source code.
[2]Clone set is a set of clones that are identical or similar to each other.
[3]*http://sdl.ist.osaka-u.ac.jp/~h-murakm/clonepacker/*
[4]*http://d3js.org/*

trunk/source/org/jfree/chart/plot/CombinedDomainXYPlot.java

```
  604: protected void setFixedRangeAxisSpaceForSubplots(AxisSpace space) {
  605:   Iterator iterator = this.subplots.iterator();
  606:   while (iterator.hasNext()) {
  607:      XYPlot plot = (XYPlot) iterator.next();
- 608:      plot.setFixedRangeAxisSpace(space);
+          plot.setFixedRangeAxisSpace(space, false);
  609:   }
  700: }
```

This modification was occurred in **04/Dec./2007**

trunk/source/org/jfree/chart/plot/CombinedDomainCategoryPlot.java

```
  465: protected void setFixedRangeAxisSpaceForSubplots(AxisSpace space) {
  466:   Iterator iterator = this.subplots.iterator();
  467:   while (iterator.hasNext()) {
  468:      CategoryPlot plot = (CategoryPlot) iterator.next();
- 469:      plot.setFixedRangeAxisSpace(space);
+          plot.setFixedRangeAxisSpace(space, false);
  470:   }
  471: }
```

This modification was occurred in **28/Mar./2008**

Fig. 3. Modifications in JFreeChart

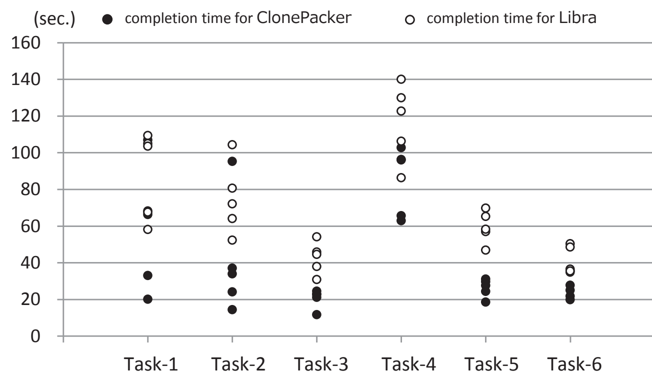

Fig. 4. Results of the task completion time

and its end line. The users can also browse the source code of the clones at the bottom view C.

C. Example of Supporting Scenario

Figure 3 shows two code fragments in JFreeChart. The 608th line of CombinedDomainXYPlot.java and the 469th line of CombinedDomainCategoryPlot.java include the same method invocations. One was modified in 04/Dec./2007 and the other was modified in 28/Mar./2008. From the commit log of 28/Mar./2008, the modification in CombinedDomainCategoryPlot.java was for a bug fix. Thus, the two method invocations must have been modified simultaneously. However, the programmers overlooked the modification in CombinedDomainCategoryPlot.java. In Fig 3, the two code fragments are clones. By using ClonePacker in 04/Dec./2007, the programmers would have understood that the two method invocations must have been modified simultaneously. Therefore, ClonePacker is useful for preventing code fragments that should be modified simultaneously from being overlooked.

V. Experiment

A. Methodology

In order to evaluate ClonePacker, we conducted an experiment with participants. The participants performed some tasks with ClonePacker and Libra. Then, we compared task completion time of ClonePacker and Libra. In this experiment, ten participants took part in the experiment. Eight participants were master's course students, and the other two participants were undergraduate students at Osaka University.

First, we divided the participants into two groups, called G_A and G_B. Since the number of the participants was ten, each group had five participants.

Second, each group worked on the tasks. All of the tasks were very simple, *"Please report locations of all clones of the given method"*. In each task, the participants were given one target method, then they found its clones by using the tools and reported the locations of detected clones. In this experiment, we set *minimum token length* as 30 and *the number of allowed gapped statements* as 2. We made the participants measure their task completion time from the beginning to the end in each task. Table I shows details of the tasks. All of the target methods were found in JHotDraw 6.0 beta 1. For example, in Task-2, ClonePacker found three clones (two Type-2 clones and one Type-3 clone). However, Libra found only two Type-2 clones because Libra used CCFinder for detecting clones and CCFinder did not have a capability of detecting Type-3 clones.

Although ClonePacker reported types of the detected clone, the participants had to report only the locations of the detected clones. The reason is that Libra did not report types of detected clones and we would like to provide a fair comparison between ClonePacker and Libra. Furthermore, in order to achieve a fair comparison, both the groups changed the tools at the timing of finishing a half of the tasks. G_A used ClonePacker and G_B used Libra for working on Task-1, Task-2 and Task-3. Then, G_A used Libra and G_B used ClonePacker for working on Task-4, Task-5 and Task-6.

B. Experimental Results

Figure 4 shows results of the task completion time. Its horizontal axis represents each task and the vertical axis represents time. Each dot represents the reporting time per clone. For example, in Task-1, the fastest participant using ClonePacker took about 20 seconds per clone to report locations of detected clones. From Fig. 4, it was likely that the participants using ClonePacker reported the locations of clones faster than Libra.

In order to show that there was a significant difference between the completion time for ClonePacker and Libra, we introduced the following null and alternative hypotheses.

TABLE I. Details of the experimental tasks

Target project	Tasks	Target method	Locations of the target method (start line - end line)	# Type-1	# Type-2	# Type-3
JHotDraw (6.0 beta 1)	Task-1	suite	src/org/jhotdraw/test/samples/minimap/MinimapSuite.java (37 - 57)	0	2	0
	Task-2	handles	src/org/jhotdraw/figures/GroupFigure.java (67 - 74)	0	2	1
	Task-3	draw	src/org/jhotdraw/contrib/PolygonScaleHandle.java (111 - 129)	4	3	1
	Task-4	store	src/org/jhotdraw/util/SerializationStorageFormat.java (62 - 68)	0	1	0
	Task-5	fillRoundRect	src/org/jhotdraw/contrib/zoom/ScalingGraphics.java (212 - 217)	0	2	2
	Task-6	handles	src/org/jhotdraw/contrib/TextAreaFigure.java (299 - 303)	5	0	1

H_0: The null hypothesis is that there is no significant difference between the completion time for ClonePacker and Libra.

H_1: The alternative hypothesis is that there is a significant difference between the completion time for ClonePacker and Libra.

We confirmed that completion time for ClonePacker and Libra have equal variances and do not follow a normal distribution at 0.05 level of a significance by using F-test and Shapiro-Wilk test, respectively. Thus, we conducted Wilcoxon test. The p-value obtained from Wilcoxon test was 6.724e-05. Since p-value was less than 0.05, we rejected H_0 and adopted H_1. Therefore, there was a significant difference between the completion time for ClonePacker and Libra. From the result of Wilcoxon test and Fig.4, we confirmed that the participants using ClonePacker reported the locations of clones faster than Libra.

VI. Threats to Validity

A. Configurations of Clone Detection

In this experiment, we set *minimum token length* as 30 and *the number of allowed gapped statements* as 2. In general, configurations of a clone detection strongly affect the detection results. Wang et al. proposed a technique to find suitable configurations of a clone detection automatically [7]. If we use their technique for finding suitable configurations, we may obtain different results from this experiment.

B. Target Software System

We used only one target software system in this study. If we use other software systems, the results might be different. In order to minimize this threat, we should apply ClonePacker to many other systems. Furthermore, ClonePacker visualizes many circles for large software systems including many clones. In such a case, in order to visualize all detected clones, each circle is likely to be small. Thus, the user may not be able to browse detected clones efficiently. In the future, we are going to tackle this problem.

C. Participants

Ten participants used ClonePacker and Libra for conducting the given tasks. All of the participants had experiences of Java programming more than one year. If their programming skills are differed widely, the differences would affect their completion time of the given tasks. However, we tried our best to allocate the participants by considering their programming experiences. Moreover, the participants changed the tools at the timing of finishing a half of the tasks. Hence, we considered that we were able to minimize the differences of skills in G_A and G_B.

VII. Related Works

Asaduzzaman et al. developed a clone analysis support tool, named VisCad [8]. Inputs of VisCad are the target source files and the clones obtained from one of some clone detectors (e.g. NiCad [9], CCFinder [3] and so on). Then, users can analyze the clones with a scatter plot, a tree map and a hierarchical dependency graph. The biggest difference between ClonePacker and VisCad is its usage. VisCad was designed to be used when programmers would like to investigate all clones in a software system. On the other hand, ClonePacker was designed to be used when programmers modify a code fragment and check its clones.

Hauptmann et al. proposed a technique that shows clone detection results by using edge bundles [10]. The characteristic of their technique is that it associates the clone detection results with a file hierarchy. Both of their and our technique focus on a file hierarchy. However, granularities of clone results presented by these techniques are different. The technique of Hauptmann et al. shows file-based clone results. On the other hand, our technique shows method-based clone results.

VIII. Conclusions

In this paper, we introduced our Eclipse plugin, named ClonePacker. It helps programmers when they modify a code fragment and check its clones. ClonePacker receives a set of source files and a method that is to be modified from programmers. Then, ClonePacker detects clones of the method from the source files. Finally, ClonePacker visualizes the detection results by using Circle Packing.

We conducted an experiment with participants to compare task completion time of ClonePacker and Libra. As a result, we confirmed that programmers using ClonePacker reported the locations of clones faster than Libra. In the future, we are going to apply ClonePacker to many systems.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Numbers 25220003, 24650011, and 24680002.

References

[1] N. Bettenburg, W. Shang, W. M. Ibrahim, B. Adams, Y. Zou, and A. E. Hassan, "An empirical study on inconsistent changes to code clones at the release level," *Science of Computer Programming*, vol. 77, no. 6, pp. 760–776, 2012.

[2] Y. Higo, Y. Ueda, S. Kusumoto, and K. Inoue, "Simultaneous modification support based on code clone analysis," in *APSEC*, 2007, pp. 262–269.

[3] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A multi-linguistic token-based code clone detection system for large scale source code," *TSE*, vol. 28, no. 7, pp. 654–670, 2002.

[4] "Circle packing - blocks," http://bl.ocks.org/mbostock/4063530.

[5] S. Bellon, R. Koschke, G. Antniol, J. Krinke, and E. Merlo, "Comparison and evaluation of clone detection tools," *TSE*, vol. 33, no. 9, pp. 577–591, 2007.

[6] H. Murakami, K. Hotta, Y. Higo, H. Igaki, and S. Kusumoto, "Gapped code clone detection with lightweight source code analysis," in *ICPC*, 2013, pp. 93–102.

[7] T. Wang, M. Harman, Y. Jia, and J. Krinke, "Searching for better configurations: A rigorous approach to clone evaluation," in *FSE*, 2013, pp. 455–465.

[8] M. Asaduzzaman, C. K. Roy, and K. A. Schneider, "Viscad: Flexible code clone analysis support for nicad," in *IWSC*, 2011, pp. 77–78.

[9] C. K. Roy and J. R. Cordy, "Nicad: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization," in *ICPC*, 2008, pp. 172–181.

[10] B. Hauptmann, V. Bauer, and M. Junker, "Using edge bundle views for clone visualization," in *IWSC*, 2012, pp. 86–87.

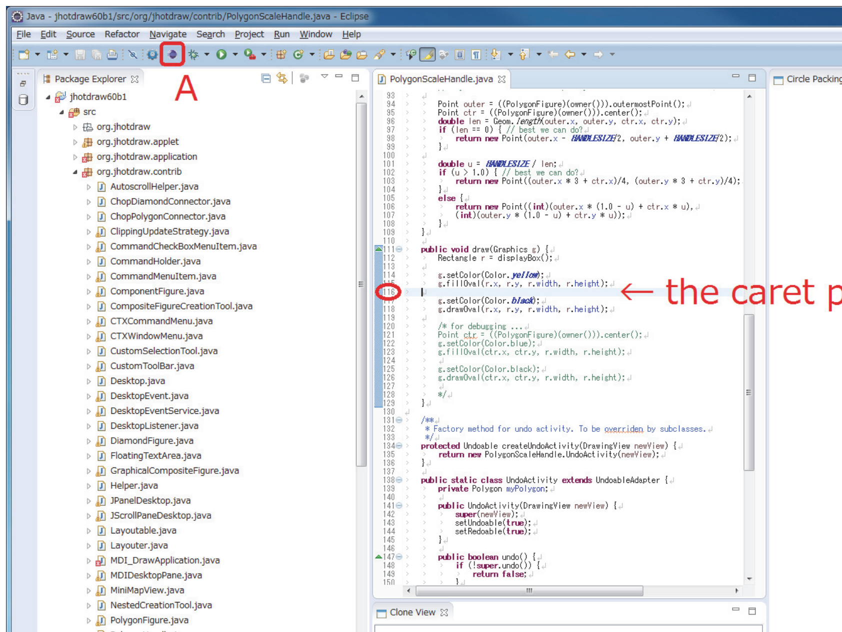

Fig. 5. Screenshot of ClonePacker at a startup time

Fig. 6. Screenshot of ClonePacker for viewing the detected clones