CRat: A Refactoring Support Tool for Form Template Method

Keisuke Hotta, Yoshiki Higo, Hiroshi Igaki, Shinji Kusumoto
Graduate School of Information Science and Technology, Osaka University
1-5, Yamadaoka, Suita, Osaka, 565-0871, Japan
{k-hotta, higo, igaki, kusumoto}@ist.osaka-u.ac.jp

***Abstract*—Refactoring is important for efficient software maintenance. However, manual operations for refactoring are complicated, and human-related errors easily occur. Tool support can help users to apply such a complicated refactoring. This paper proposes a refactoring support tool with Form Template Method pattern. The developed tool automatically identifies method pairs that can be refactored with Form Template Method, and suggests information that is required for Form Template Method application. It also has a function that metrics-based filtering for detected method pairs. The function helps users to select method pairs that should be refactored.**

***Index Terms*—Refactoring, Form Template Method, Program Dependence Graph, Software Maintenance**

I. Introduction

Refactoring is a set of operations for improving internal structure of software without changing its external behavior [1]. It has been reported that maintainability of software systems decays over time [2]. Refactoring is effective in such a case because it can reduce the decay of maintainability. However, applying refactorings takes much effort. Also, applying manual refactorings is a complicated task, so that human related errors easily occur. Consequently, techniques or tools for supporting refactoring are required, and many techniques have been proposed to assist refactorings [3].

In this paper, we focus on **Form Template Method** pattern. This pattern targets similar methods (see Fig 1). In this pattern, programmers write an outline for similar methods into the base class and implement detail processes in each derived class. By applying Form Template Method refactoring, code duplication (code clones) existing between the similar methods is merged into the base class. One of the advantages of duplication removal with Form Template Method is that this pattern can be applied to methods having some gaps.

Some researchers have proposed techniques to support Form Template Method refactorings [4], [5]. These techniques use Abstract Syntax Tree (in short, AST) as their data structures. However, these techniques cannot support removing code clones if they include the following differences even if these differences have no impacts on the behavior of the program:

- different order of code fragments, and
- different implementation (such as for- and while- loops).

We have proposed techniques for supporting Form Template Method applications [6]. This technique uses Program Dependence Graph (in short, PDG) to resolve the above issues.

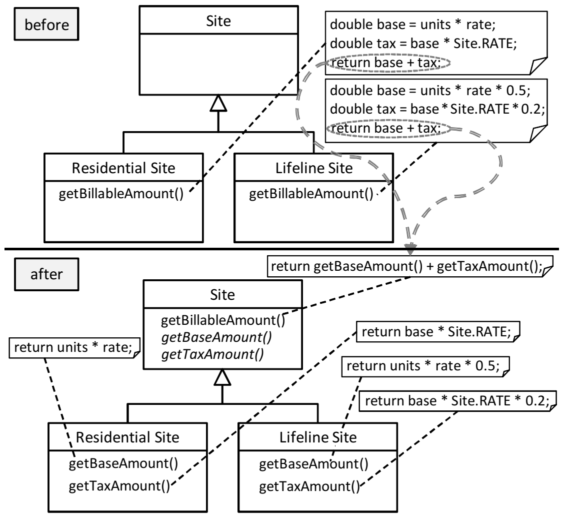


Fig. 1. An Example of the Application of *Form Template Method* [1]

This paper focuses a tool, CRat (*Clones Removal Assistant Tool*), which implements the proposed refactoring support techniques. The features of CRat are as follows:

- detects refactoring candidates automatically,
- has metrics-based filtering function for selecting refactoring targets, and
- visualizes each candidate with information used for applying refactoring.

II. Developed Tool

A. Overview

CRat requires source code of target software systems as its input. It analyzes the source code and creates PDGs with MASU, a source code analysis platform [7]. Then it detects code clones on PDGs with the existing clone detector, Scorpio [8]. CRat identifies refactoring candidates with information about code clones, and detects common and unique processes for each refactoring candidate. It visualizes each refactoring candidate by highlighting common and unique processes. Note that CRat does not modify the source code automatically, thus users need to modify the source code by themselves.

Here, a refactoring candidate indicates a pair of similar methods. CRat suggests a pair of methods as a refactoring candidate if they satisfy the following requirements:

- the two methods are defined in different classes,
- the two methods have the same base class, and
- there exists at least one clone pair (a pair of duplicate code fragments) between the two methods.

B. *Functionalities for Suggesting Refactoring Candidate*

Figure 2 shows a snapshot of CRat. The table shows all the candidate method pairs that CRat detected. When users select a method pair from the table, the source code of the methods included in the pair is shown in the right panel. We call the right panel as source code view.

Figure 3 shows a snapshot of the source code view. Common statements are colored with red, which means that they should be pulled up into the base class. On the other hand, unique statements are colored with other colors. Statements surrounded by the same color rectangles means that they can be extracted as a single method. In addition, if users click statements that are not highlighted by red, a set of statements including the statement are highlighted. Moreover, if users click a set of statements in one method, CRat also highlights the corresponding set of statements in the other method. Here, the term 'correspond' indicates that the two sets of statements can be extracted as the same signature methods. Additionally, CRat shows the signature of the method created from a set of statements if users put the cursor on the set.

C. *Functionalities for Refactoring Candidate Identifications*

CRat has a metrics-based filtering function of candidate method pairs. All metrics are calculated for each method pair. Users can specify the upper and the lower thresholds for each metric, and then CRat suggests candidates whose metrics are in the specified thresholds. The metrics are as follows.

- **SIM:** The degree of similarity between the two methods.
- **CN:** The number of statements that can be pulled up into the base class.
- **DN:** The number of statements that need to remain in each derived class.
- **LOC:** The number of lines of code.
- **DG:** The number of new methods that need to be created for Form Template Method application.
- **DOI:** The depth of inheritance from the common base class to the owner classes of the two methods.

CRat has a graphic interface to support users' specification of thresholds. This interface uses the metrics graph [9]. Figure 4 shows a snapshot of the metrics graph. Users specify the thresholds of each metric by dragging the graph. The area whose background color is gray indicates the range of thresholds for every metric, and the area whose background color is white indicates the outside of the range. In the metric graph, each polygonal curve corresponds a method pair. The polygonal curve becomes red if and only if all the metrics of the method pair represented by the polygonal curve are in the specified threshold. If any of the metrics is not in the threshold, the polygonal curve becomes gray. By using the metrics graph, users can specify thresholds of each metric graphically, and can find how many candidates satisfy the thresholds.

Method pairs represented by red polygonal curve satisfy the thresholds that users specified. Therefore, CRat shows these method pairs after users specified thresholds for each metric.

III. Related Work

Some techniques have been proposed to support Form Template Method refactorings [4], [5]. They use ASTs as their data structure, which is the most different point from the proposed method. In addition, users need to detect refactoring candidates to use previously described techniques. On the other hand, CRat detects refactoring candidate automatically, which helps users to detect refactoring opportunities.

IV. Conclusion

This paper proposed a refactoring support tool with Form Template Method. This tool is an implementation of the refactoring support technique that our research group has proposed [6]. The tool automatically detects and suggests method pairs that can be refactored with Form Template Method. Also, the tool provides the functionality of method pairs filterings, which helps users to decide which candidates should be refactored. As future work, we are going to conduct experiments to confirm the effectiveness of the developed tool.

Acknowledgment

This study has been supported in part by Grants-in-Aid for Scientific Research (A) (21240002), Grant-in-Aid for Exploratory Research (23650014), and Grand-in-Aid for Young Scientists (A) (24680002) from the Japan Society for the Promotion of Science.

References

[1] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1999.

[2] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus, "Does code decay? assessing the evidence from change management data," *IEEE Transactions on Software Engineering*, vol. 27, no. 1, pp. 1–12, Jan. 2001.

[3] T. Mens and T. Tourwé, "A Survey of Software Refactoring," *IEEE Transactions on Software Engineering*, vol. 30, no. 2, pp. 126–139, Feb. 2004.

[4] N. Juillerat and B. Hirsbrunner, "Toward an implementation of the "Form Template Method" Refactoring," in *Proc. of the 7th International Working Conference on Source Code Analysis and Manipulation*, Sep. 2007, pp. 81–90.

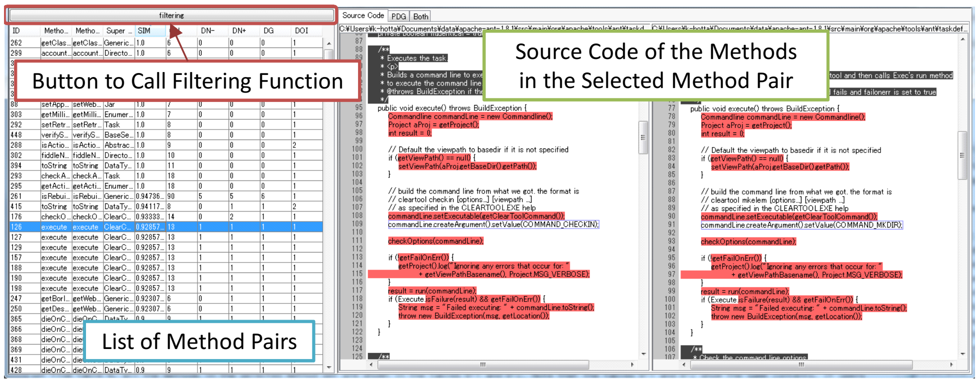
[5] M. Ioka, N. Yoshida, T. Masai, Y. Higo, and K. Inoue, "A Tool Support to Merge Similar Methods with a Cohesion Metric COB," in *Proc. of the 3rd International Workshop on Empirical Software Engineering in Practice*, Nov. 2011, pp. 23–24.

[6] K. Hotta, Y. Higo, and S. Kusumoto, "Identifying, Tailoring, and Suggesting Form Template Method Refactoring Opportunities with Program Dependence Graph," in *Proc. of the 16th European Conference on Software Maintenance and Reengineering*, Mar. 2012.

[7] Y. Higo, A. Saitoh, G. Yamada, T. Miyake, S. Kusumoto, and K. Inoue, "A Pluggable Tool for Measuring Software Metrics from Source Code," in *Proc. of the Joint Conference of the 21th International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*, Nov. 2011, pp. 3–12.

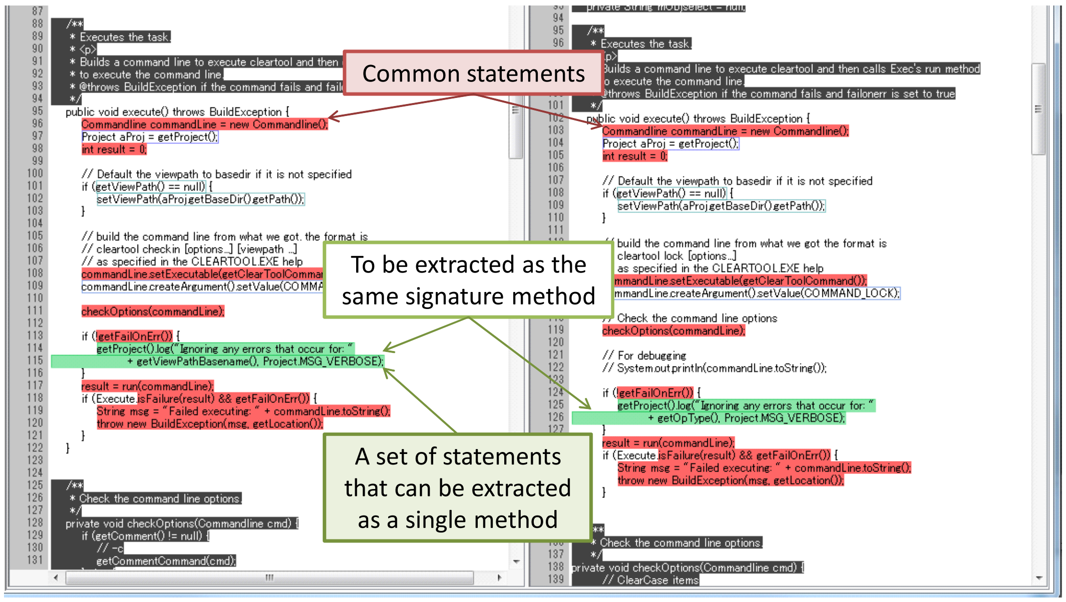
[8] Y. Higo and S. Kusumoto, "Code Clone Detection on Specialized PDGs with Heuristics," in *Proc. of the 15th European Conference on Software Maintenance and Reengineering*, Mar. 2011, pp. 75–84.

[9] Y. Higo, S. Kusumoto, and K. Inoue, "A metric-based Approach to Identifying Refactoring Opportunities for Merging Code Clones in a Java Software System," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 20, no. 6, pp. 435–461, 11 2008.


filtering
Source Code
PDG
Both
Button to Call Filtering Function
Source Code of the Methods in the Selected Method Pair
List of Method Pairs

ID	Metho...	Metho...	Super ...	SIM		DN-	DN+	DG	DOI
262	getClas...	getClas...	Generic...	1.0	6	0	0	0	1
299	account...	account...	Directo...	1.0	6	0	0	0	1
88	setApp...	setWeb...	Jar	1.0	7	0	0	0	1
303	getMilli...	getMilli...	Enumer...	1.0	7	0	0	0	1
292	setRetr...	setRetr...	Task	1.0	8	0	0	0	1
448	verifyS...	verifyS...	BaseSe...	1.0	8	0	0	0	1
288	isActio...	isActio...	Abstrac...	1.0	9	0	0	0	2
302	fiddleN...	fiddleN...	Directo...	1.0	10	0	0	0	1
394	toString	toString	DataTy...	1.0	11	0	0	0	1
293	checkA...	checkA...	Task	1.0	18	0	0	0	1
295	getActi...	getActi...	Enumer...	1.0	18	0	0	0	1
261	isRebui...	isRebui...	Generic...	0.94736...	90	5	5	6	1
415	toString	toString	DataTy...	0.94117...	8	0	1	1	2
176	checkO...	checkO...	ClearC...	0.93333...	14	0	2	1	1
126	execute	execute	ClearC...	0.92857...	13	1	1	1	1
127	execute	execute	ClearC...	0.92857...	13	1	1	1	1
129	execute	execute	ClearC...	0.92857...	13	1	1	1	1
157	execute	execute	ClearC...	0.92857...	13	1	1	1	1
188	execute	execute	ClearC...	0.92857...	13	1	1	1	1
190	execute	execute	ClearC...	0.92857...	13	1	1	1	1
198	execute	execute	ClearC...	0.92857...	13	1	1	1	1
247	getBorl...	getWeb...	Generic...	0.92307...	6	0	1	1	1
250	getDes...	getWeb...	Generic...	0.92307...	6	0	1	1	1
365	dieOnC...	dieOnC...	DataTy...	0.9	9	1	1	1	1
366	dieOnC...	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
368	dieOnC...	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
369	dieOnC...	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
431	dieOnC...	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
428	dieOnC...	dieOnC...	DataTy...	0.9	9	1	1	1	1

Fig. 2. A Snapshot of CRat


Common statements
To be extracted as the same signature method
A set of statements that can be extracted as a single method

```
    /**
     * Executes the task.
     * <p>
     * Builds a command line to execute cleartool and then
     * to execute the command line.
     * @throws BuildException if the command fails and fail
     */
    public void execute() throws BuildException {
        Commandline commandLine = new Commandline();
        Project aProj = getProject();
        int result = 0;

        // Default the viewpath to basedir if it is not specified
        if (getViewPath() == null) {
            setViewPath(aProj.getBaseDir().getPath());
        }

        // build the command line from what we got. the format is
        // cleartool checkin [options...] [viewpath ...]
        // as specified in the CLEARTOOL.EXE help
        commandLine.setExecutable(getClearToolComma
        commandLine.createArgument().setValue(COMMA

        checkOptions(commandLine);

        if (!getFailOnErr()) {
            getProject().log("Ignoring any errors that occur for: "
                    + getViewPathBasename(), Project.MSG_VERBOSE);
        }
        result = run(commandLine);
        if (Execute.isFailure(result) && getFailOnErr()) {
            String msg = "Failed executing: " + commandLine.toString();
            throw new BuildException(msg, getLocation());
        }
    }

    /**
     * Check the command line options.
     */
    private void checkOptions(Commandline cmd) {
        if (getComment() != null) {
            // -c
            getCommentCommand(cmd);
```

```
    private String mObjselect = null;

    /**
     * Executes the task.
     * <p>
     * Builds a command line to execute cleartool and then calls Exec's run method
     * to execute the command line.
     * @throws BuildException if the command fails and failonerr is set to true
     */
    public void execute() throws BuildException {
        Commandline commandLine = new Commandline();
        Project aProj = getProject();
        int result = 0;

        // Default the viewpath to basedir if it is not specified
        if (getViewPath() == null) {
            setViewPath(aProj.getBaseDir().getPath());
        }

        // build the command line from what we got the format is
        // cleartool lock [options...]
        // as specified in the CLEARTOOL.EXE help
        commandLine.setExecutable(getClearToolCommand());
        commandLine.createArgument().setValue(COMMAND_LOCK);

        // Check the command line options
        checkOptions(commandLine);

        // For debugging
        // System.out.println(commandLine.toString());

        if (!getFailOnErr()) {
            getProject().log("Ignoring any errors that occur for: "
                    + getOpType(), Project.MSG_VERBOSE);
        }
        result = run(commandLine);
        if (Execute.isFailure(result) && getFailOnErr()) {
            String msg = "Failed executing: " + commandLine.toString();
            throw new BuildException(msg, getLocation());
        }
    }

    /**
     * Check the command line options.
     */
private void checkOptions(Commandline cmd) {
        // ClearCase items
```

Fig. 3. A Snapshot of Source Code View

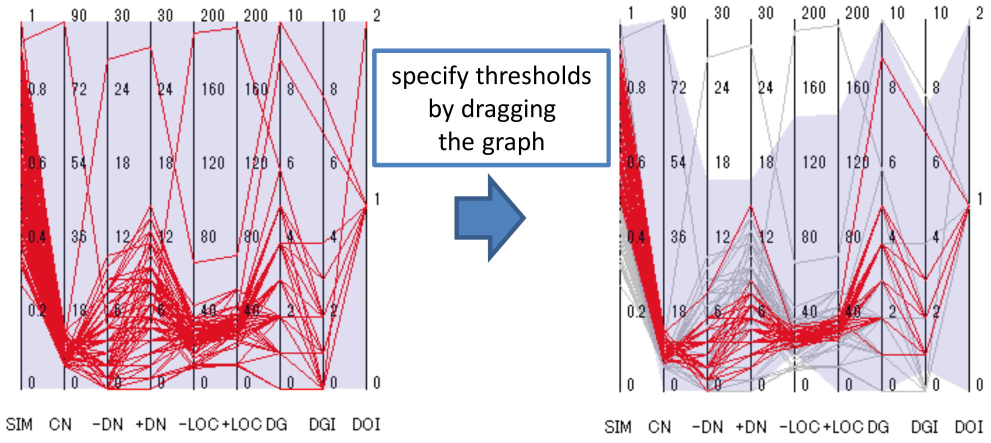


Fig. 4. A Metrics Graph