Method Differentiator Using Slice-based Cohesion Metrics

Akira Goto[1], Norihiro Yoshida[2], Masakazu Ioka[1], Eunjong Choi[1], Katsuro Inoue[1]
[1]Graduate School of Information Science and Technology, Osaka University, Japan
{a-gotoh, m-ioka, ejchoi, inoue}@ist.osaka-u.ac.jp
[2]Graduate School of Information Science, Nara Institute of Science and Technology, Japan
yoshida@is.naist.jp

ABSTRACT

It is important to understand semantic differences between a pair of Java methods during maintenance. However, textual or syntactic difference is insufficient to give clear idea which code fragment realizes a single functionality in Java methods. In this paper, we present an Eclipse plugin for semantic differentiation of a given pair of Java methods.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Restructuring, reverse engineering, and reengineering*

General Terms

Experimentation

Keywords

Cohesion metrics, Program slicing

1. INTRODUCTION

Developers often need to understand semantic differences between a pair of Java methods during software maintenance (e.g., enhancement, form template method refactoring[3]). However, textual or syntactic difference is insufficient to give clear idea which code fragment realizes a single functionality in Java methods.

Therefore, we have developed an Eclipse plugin for sematic differentiation of a given pair of Java methods. This plugin firstly identifies syntactic differences a given pair of Java methods, and then finds cohesive code fragments including at least one syntactic difference. Finally, this plugin proposes (highlights) two sets of cohesive code fragments as sematic differences between a given pair of Java methods. The cohesiveness of code fragments is calculated by slice-based cohesion metrics [4, 2] (i.e., Tightness, Coverage, and Overlap) which represent the functional uniqueness of a code fragment or method.

AOSD'13 Companion, March 24-29, 2013, Fukuoka, Japan.

In our tool demonstration, our plugin visualizes semantic differences between a pair of methods involved in an open source software (see Figure 5 and 6).

2. PROPOSED TOOL

Figure 1 shows an overview of the proposed tool. The proposed tool accepts a pair of given Java methods as input, and then identifies sets of Semantic Differences (SDs). Given two Java methods(m_a, m_b), code fragments CFs extracted from them are called a set of SDs when CFs satisfy all of the following three conditions:

1. Any SD in CFs does not overlap any other SD in the code fragments.
2. Any SD in CFs involves at least one syntactic difference between m_a and m_b.
3. After all SDs in CFs are removed, m_a and m_b are syntactically same.

After the SD identification, identified sets of SDs are ranked based on slice-based cohesion metrics. Highly-ranked set of SDs means that each SD realizes a single functionality. Once a user selected one of sets of SDs, SDs in the selected set are highlighted (see Figure 5 and 6).

2.1 Detecting Syntactic Differences

First of all, the proposed tool extracts abstract syntax trees (ASTs) from a pair of given Java methods. The AST extraction is performed by Eclipse JDT[1].

Then, syntactic differences are detected by comparing two ASTs. Figure 2 shows an example of a syntactic difference. A node in an AST has a type (e.g., IFStatement, NumberLiteral). In the case of several types, a node has a value (e.g., variable, constant, literal). The detection starts from root nodes of the ASTs. The comparison of two nodes is performed according to their types. Only if those nodes have values, their values are compared after that. In the case that those two nodes are matched, the detection moves to the child nodes of them. Otherwise, they are detected as the difference. Please note that the minimum level of a difference is a statement in the detection.

2.2 Identifying Semantic Differences

To identify SDs from given two Java methods, code regions detected as syntactic differences are expanded. This expansion has two types. The first one is expansion towards

[1]http://www.eclipse.org/jdt/index.php

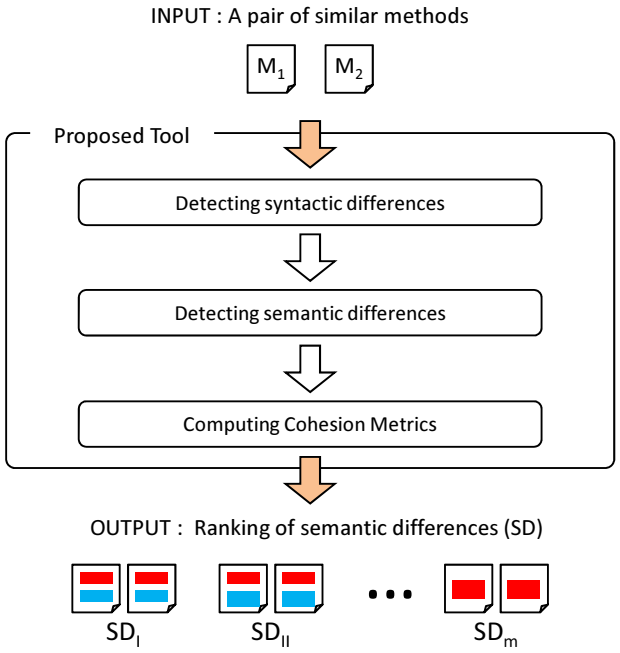

Figure 1: Overview of the proposed tool

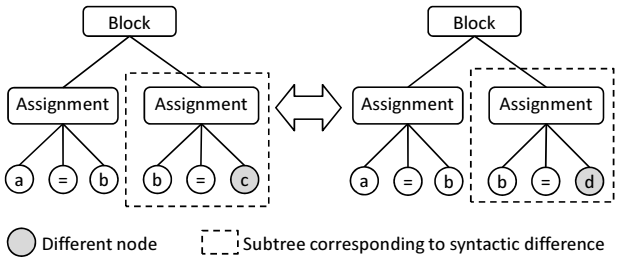

Figure 2: An example of a syntactic difference

sibling nodes (ETS). The other one is expansion towards parent nodes (ETP). The expansion starts from a node recognized as a difference, and then apply ETS to the code region of the node. After ETS is applied to all of siblings of the node, ETP is applied to the siblings. After each expansion, each code region is identified as a SD. The expansion is ended when it moves to a root node of each AST.

Figure 3 represents examples of ETS and ETP. In this figure, this expantion starts from $a = b$ corresponding to the syntactic difference, and then the perform ETS. Finally, ETP is perfromed for moving to the if statement.

Given two Java methods often has one more than syntactic differences. In this case, this expantion is applied to each of the differences, and then all of resultant sets of SDs are combined.

2.3 Sorting by Slice-based Cohesion Metrics

Identified sets of SDs are ranked according to the cohesiveness of each SD. Cohesiveness of each SD is calculated based on slice-based cohesion metrics. To calculate those metrics, program slice and dependence graph is derived using Antlr[2] We selected Tightness, Coverage, and Overlap from slice-based cohesion metrics [4] because the characteristics of the selected metrics are empirically verified[2]. A user can select one of the slice-based metrics in the proposed tool.

[2]http://www.antlr.org

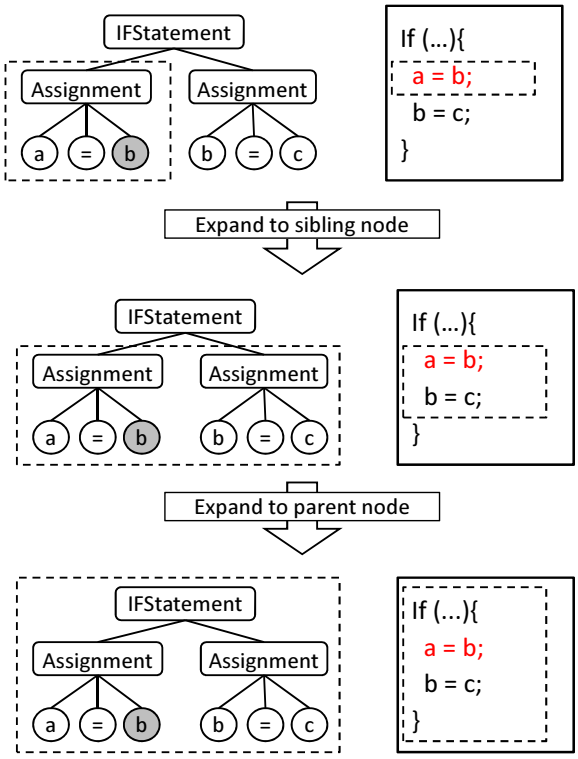

Figure 3: Example of expantions

We redefine Tightness, Coverage, and Overlap as follows for small code fragments. In the following definitions, M is a method, $len(M)$ is the number of statements in M, V is a set of parameters and return values, V_i is a set of parameters in M, V_o is a set of return values in M, FSL_x is a forward slice in the case that the entry point is variable x, BSL_x is a backward slice in the case that the entry point is variable x, SL_{int} is an intersection of forward slices obtained for all variables in V_i and backward slices obtained for all variables in V_o.

$$FTightness(M) = \frac{|SL_{int}|}{len(M)}$$

$$FCoverage(M) = \frac{1}{|V|}\left(\sum_{x \in V_i} \frac{|FSL_x|}{len(M)} + \sum_{x \in V_o} \frac{|BSL_x|}{len(M)}\right)$$

$$FOverlap(M) = \frac{1}{|V|}\left(\sum_{x \in V_i} \frac{|SL_{int}|}{|FSL_x|} + \sum_{x \in V_o} \frac{|SL_{int}|}{|BSL_x|}\right)$$

We suppose that each SD is extracted as a method M, each slice-based metric is calculated according to the above definitions. The slice-based cohesion metric of a set of SDs is an average of metrics of SDs.

3. DEMONSTRATION

As a demonstration, we applied the proposed tool to a pair of two Java methods in Apache Ant 1.7.0[3]. One of those methods is executeDrawOperation method in Arc class. The

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

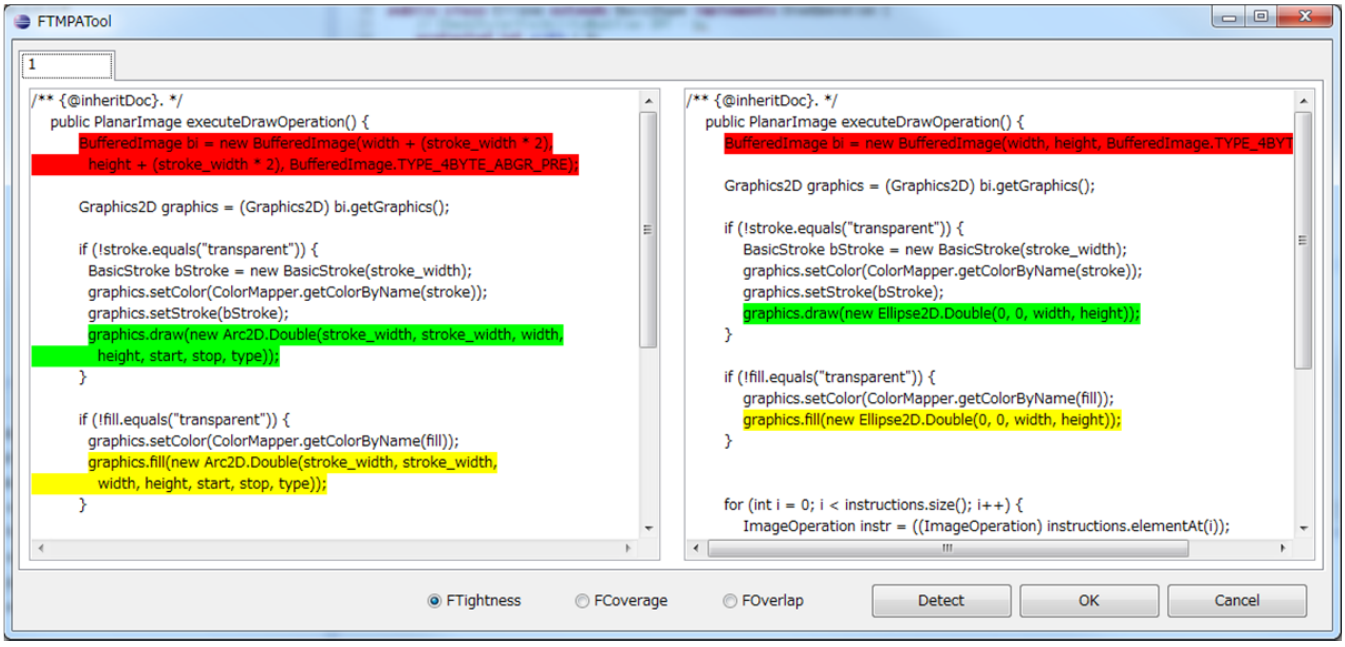

Figure 4: Visualization of syntactic differences

other one is executeDrawOperation method in Ellipse class. Those methods are similar but syntactically different.

At first, the proposed tool simply highlights syntactic differences between the given methods (Figure 4). The corresponding differences are painted by the same color.

Once a user pushes the *Detect* button at the bottom of the window, the expantion, the metric calculation, and the sorting are performed. Figure 5 shows the screenshot after the sorting. In this figure, a tab corresponds each set of SD, and indicates selected slice-based cohesion metric. In this case, 14 sets of SDs are derived, and the tab 2 is selected. FTightness metric in the tab is 0.88.

In this figure, the red SDs correspond buffer allocation, the green SDs correspond rendering of outline of a graphic, and the yellow SDs correspond rendering of the body a graphic. According to this result, the proposed tool successfully presented SDs using FTightness metric, each of which realizes a single functionality.

Figure 6 is a screenshot that illustrates another set of SDs. Although the FTightness metric 0.62 is lower than before, the green SDs correspond whole rendering of a graphic. Appropriate set of SDs depends on the context of development (e.g., bug fixing, refactoring). We believe that suggesting multiple sets of SD by the proposed tool is useful for practical software development.

4. RELATED WORK

In our earlier work, we have developed a method differentiator using Cohesion of Blocks (COB)[1]. It is easier to compute COB compared to slice-base cohesion metrics. We believe that slice-based metrics used by this study are more appropriate to capture the semantic of a program.

Xing, et al. presented CloneDifferentiator[5]. It is a differentiator for duplicated code. We have developed a differentiator plugin for a pair of Java methods. Our plugin can be applied to a pair of partly similar but mostly different Java methods. CloneDifferentiator characterizes identified differences using program dependence graph. On the other hand, we focus on the finding of cohesive code fragment including at least one syntactic difference for giving clear idea of semantic differences in a given pair of Java methods.

5. SUMMARY

We presented an Eclipse plugin for sematic differentiation of a given pair of Java methods. As a demonstration, we showed the visualization of the semantic differences between a pair of methods involved in Apache Ant.

One of future works is conducting empirical studies to show the usefulness of the proposed tool. We plan to confirm that the proposed tool is able to reduce the effort for bug fixing and refactoring. Also, the development of refactoring tool based on the proposed tool is interesting challenge.

6. REFERENCES

[1] 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 IWESEP 2011*, pages 23–24, 2011.

[2] T. M. Meyers and D. Binkley. An empirical study of slice-based cohesion and coupling metrics. *ACM Trans. Softw. Eng. Methodol.*, 17(2):1–27, 2007.

[3] M.Fowler. *Refactoring: Improving the Design of Existing Code*. Addison Wesley, 1999.

[4] M. Weiser. Program slicing. In *Proc of ICSE 1981*, pages 439–449, 1981.

[5] Z. Xing, Y. Xue, and S. Jarzabek. Clonedifferentiator: Analyzing clones by differentiation. In *Proc of ASE 2011*, pages 576–579, 2011.

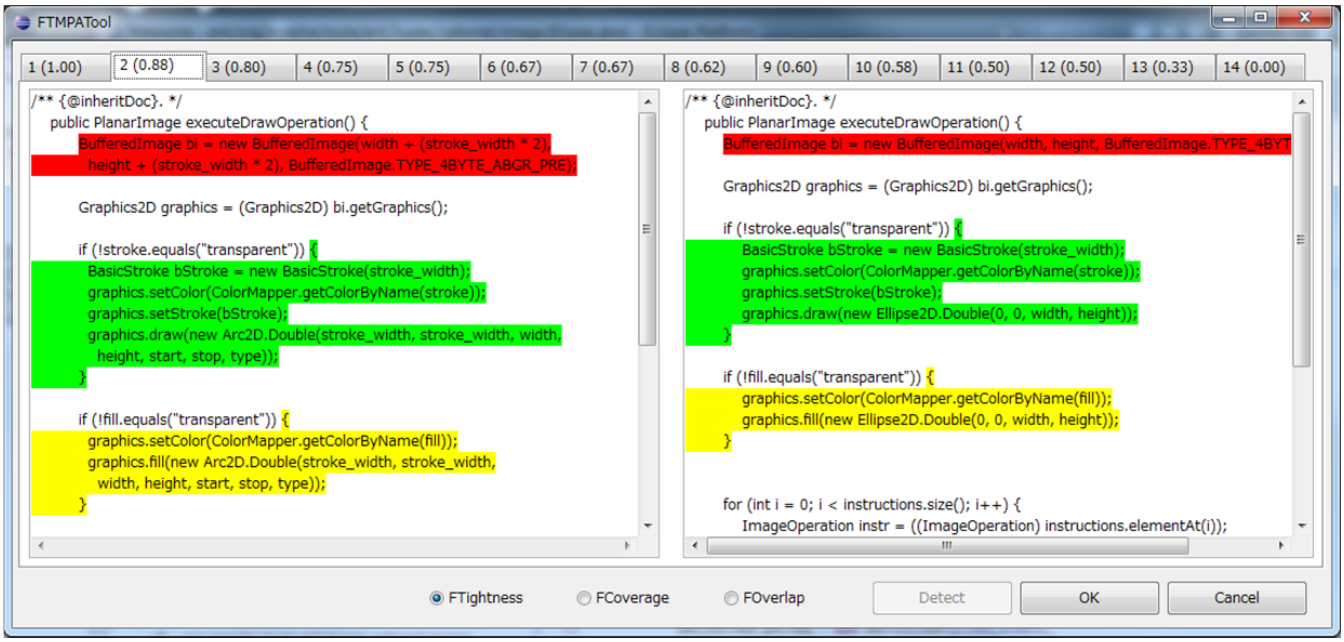

Figure 5: Visualization of semantic differences (FTightness is 0.88)

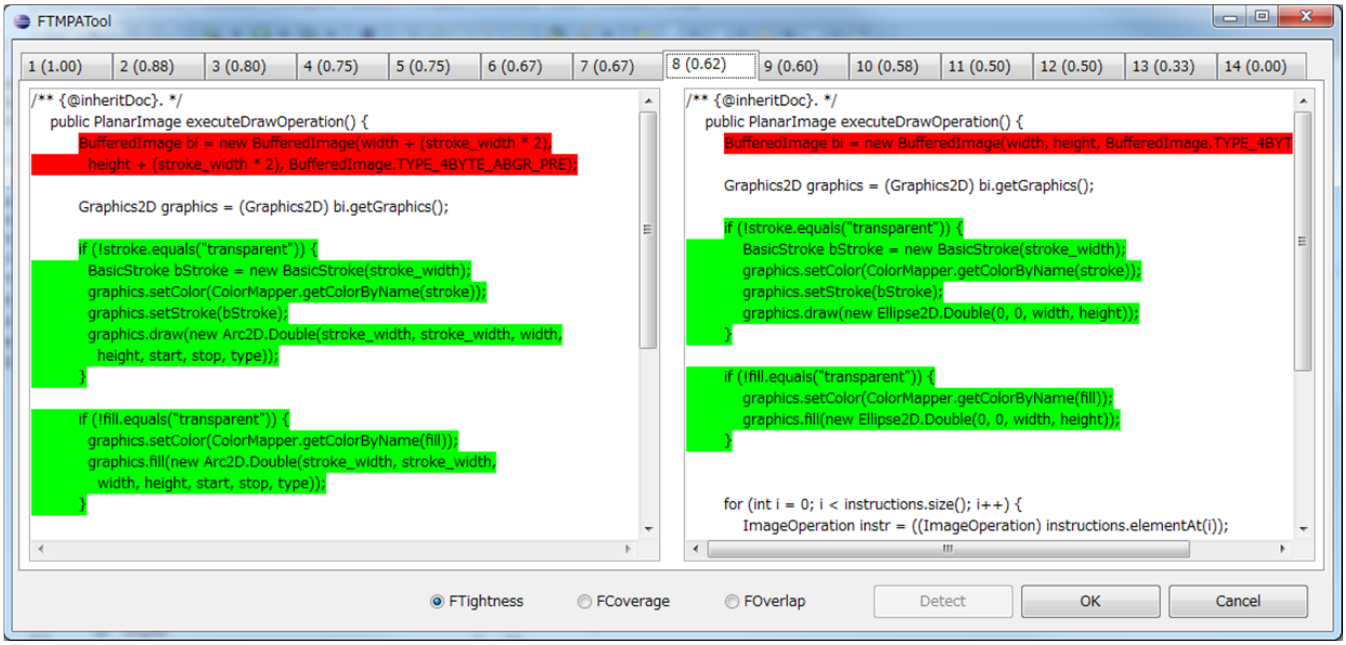

Figure 6: Visualization of semantic differences (FTightness is 0.62)