Towards A Collection of Refactoring Patterns Based on Code Clone Categorization

Masayuki Tokunaga
Osaka University
m-toku@ist.osaka-u.ac.jp

Norihiro Yoshida
Nara Institute of Science and Technology
yoshida@is.naist.jp

Kazuki Yoshioka
Osaka University
k-yosiok@ist.osaka-u.ac.jp

Makoto Matsushita
Osaka University
matusita@ist.osaka-u.ac.jp

Katsuro Inoue
Osaka University
inoue@ist.osaka-u.ac.jp

ABSTRACT

Code clone is a code fragment that has identical or similar fragments to it in the source code. Fowler and Kereivsky wrote several techniques to remove code clones in refactoring patterns that he has developed. Those refactoring patterns include characteristics of code clone and corresponding steps to merge code clones. However, according to our experience in code clone research field, a lot of difference types exist between code clones similar to each others, and most of those difference types are unmentioned in previous refactoring pattern. It is necessary to develop a collection of patterns for clone refactoring based on precise code clone categorization. In this paper, we describe an approach to developing our intended collection of refactoring patterns and then propose an example of refactoring pattern that we have developed.

Categories and Subject Descriptors

D.1.5 [Programming Techniques]: Software—Object-oriented Programming

General Terms

Documentation

1. INTRODUCTION

A code clone is a code fragment that has other code siblings identical or similar to it in the source code [1]. When developers modify a code fragment, they must find code clones corresponding the code fragment and then determine whether or not to modify those code clones [2, 3, 4, 5].

Some researchers argue that clone siblings make software maintenance more difficult [2, 3, 4]. Yet, recent research has pointed out that code clone is not always harmful but still indicates considerable opportunities for refactoring [6, 7].

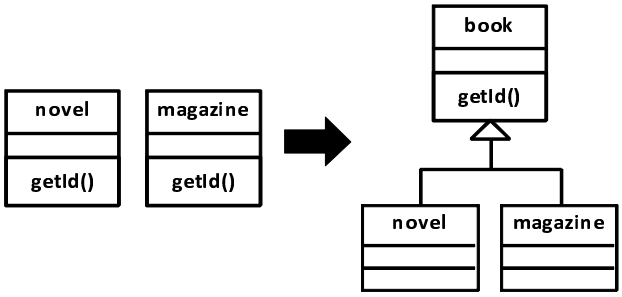

Figure 1: Example of application of refactoring

A clone set (i.e., a set of code clones identical or similar to each other) can be merged into one module through refactoring by developers[8] (c.f., there are some code clones that are unable to extract applying refactoring [6]). Refactoring is the process of changing a software system in such a way that it is unaltered alter the external behavior of the code yet improves its internal structure [8]. Figure 1 is an example of refactoring for code clone. When there are exactly matching methods between sibling classes, the method is pulled up to parent class.

When programmers would like to apply refactoring for software maintenance, refactoring patterns can be used. Thanks to refactoring patterns, the effort of refactoring decreases and the quality of code increases. A refactoring pattern is not a general pattern like a design pattern but a pattern of refactoring processes (i.e., series of actions to achieve a refactoring). With refactoring pattern, beginner in refactoring can learn and apply experimental modifications of refactoring.

Code clone categorization is collection of code clone characteristics that correspond some code clone categorization criteria defined by author of categorization. Steps of clone

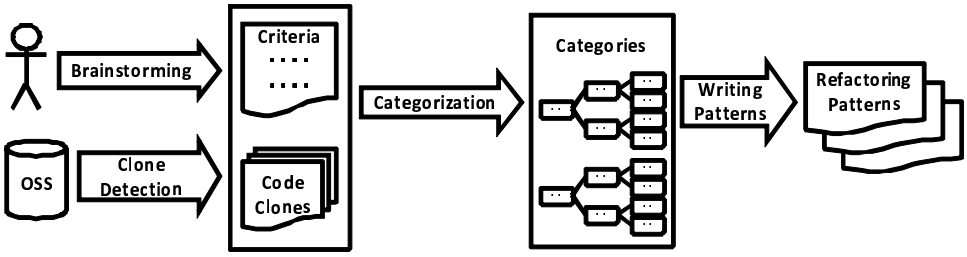

Figure 2: An overview of our propose solutions

refactoring are strongly under the influence of code clone categorization of target code. Even if programmers understand existing patterns for clone refactoring, refactoring processes often need additional modifications that are unmentioned in any pattern for clone refactoring if cloning type is slightly different from types described in their books. When the programmers apply the unmentioned modifications, the risk of introducing bugs can occur. Therefore, refactoring pattern must be based on the detailed categorization of code clone characteristics.

Fowler [8] and Kerievsky [9] proposed catalogues of refactoring patterns. Their catalogues include refactoring patterns to merge a clone set into a module. Each of those refactoring patterns explains cloning type (i.e., how code is cloned) and how to merge corresponding cloned code into one module. Code clone categorizations are just briefly explained by Fowler and Kerievsky in their books because those catalogues are not only focused on clone refactoring. For example, they describe wide range of refactoring like a refactoring for shortening method. They describe only five categorizations of code clone in Fowler's book [8]. The suggested categorizations don't include detail characteristics of code clone for application of refactoring (e.g., difference among code fragments in a clone set).

There is lacking of the existence of refactoring patterns based on the detail categorization of code clone characteristics. Therefore, we suggest a method to create such refactoring patterns. In this paper, at first we propose a method to create detailed clone categorization, then describe a method to create refactoring pattern based on fine-grained clone categorization. Finally, we introduce an example of our pattern for clone refactoring. The concept of overall method is described in Figure 2.

2. CLONE CATEGORIZATION

2.1 Problems

- Lack of a detailed categorization corresponding code clone characteristics (Problem A)

  – Existing books describing the way to merge code clones categorize clone types briefly because these books are not specialized in merging code clones. The processes of refactoring depend on advanced code clone characteristics because code clone characteristics correspond the external behavior of the code and refactoring must not change the external behavior of the code.

  When programmers apply refactoring patterns using existing books, they often need additional modifications. For example, code clones having the instanceof statement should be applied especial refactoring but there is no description written of this case in existing books.

- Lack of reality and Integrity (Problem B)

  – Code clone categorizations must be related to actual codes and be comprehensive toward all kinds of clone characteristic.

2.2 Solution

2.2.1 Method to categorize code clones

The following is the steps of the proposed method of clone categorization.

Step1: Brainstorming for defining code clone categorization criteria

Step2: Code clone detection from open source software

Step3: Random selection of clone sets

Step4: Clone set categorization

Step5: Refinement of code clone categorization criteria

Step6: Refinement of clone set categorization

Toward detail categorizations (Problem A), we define criteria of categorization based on the characteristics of code clone needed to determine the way of refactoring in Step 1. We develop categorizations about all defined criteria of categorization in Step 4.

Toward assurance of correspondence between categorizations and actual code (Problem B), we suggest the processes using actual code in Step 2. Specifically, we detect clone sets with CCFinder [1] from eleven open source software projects. There were so many detected clone sets from OSS projects. We selected 100 clone sets randomly because it is impossible to use all detected clone sets.

Toward assurance of completeness of categorization (Problem B), we evaluate the corresponding coverage of categorization using other clone sets and refine the categorization in Step 5, 6.

2.2.2 *Deliverable*

The followings describe the result that we created with suggested method for Java code.

Step 1: Brainstorming for code clone categorization criteria

We performed brainstorming on criteria of code clone categorization. We first considered the important code clone categorization criteria of refactoring based on applying the Fowler's refactoring pattern. As a result, we found basic criteria including the ones that Fowler and Kerievsky proposed. It might be insufficient because it is the deliverable of our experiments. By repeated application of suggested steps, it will improve accuracy.

The followings are five examples of criteria found during these steps.

- Differences among code fragments in a clone set
  - e.g., Type name, local variable name, constant name
- Class hierarchy relationship among code fragments in a clone set
  - e.g., code fragments belong to same class, code fragments belong to classes have direct common parent classes, code fragment belong to classes no common parent class
- Granularity of code fragments in a clone set
  - e.g., method-level clone, statement-level clone
- Existence of jump statement
  - Existence of break or continue statements: due to this criterion, we can grasp the range of code fragments with a coherent semantic.
- Existence of instanceof statement
  - Existence of instanceof statement: there are huge differences of refactoring processes in whether to have "instanceof statement" or not.

Step 2: Code clone detection from open source software

We detected clone sets from eleven OSS projects (e.g., ANTLR, Ant, Tomcat, JBoss) by a code clone detection tool CCFinder.

Step 3: Random selection of clone sets

We randomly selected 100 clone sets from CCFinder [1] output because of constraint of time.

Step 4: Clone Set Categorization

We categorized selected clone sets into several categories according to the criteria found in Step 1.

We describe examples of the detail categories of criteria "class hierarchy relationship among code fragments" in Figure 2 and "granularity of code fragments" in Figure 3.

Step 5: Refinement of code clone categorization criteria

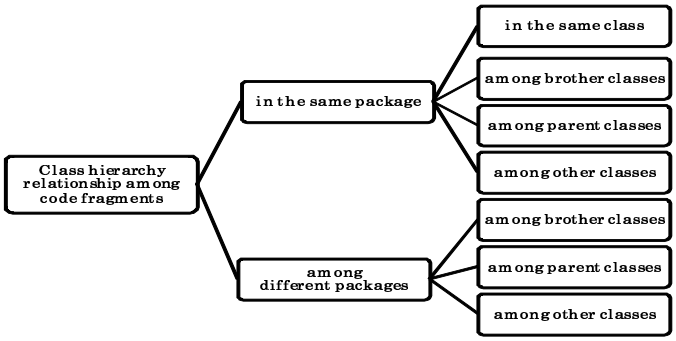

Figure 3: Example of categorization (class hierarchy relationship among code fragments)

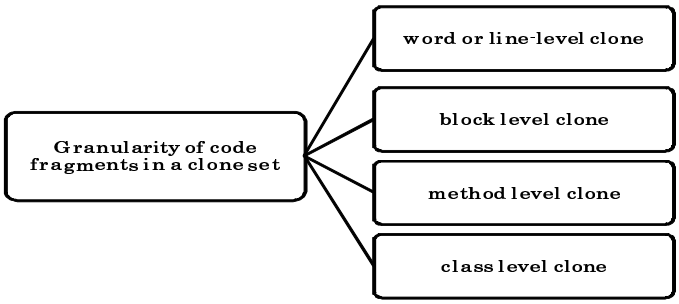

Figure 4: Example of categorization (granularity of code fragments)

In Step 4, we refined code clone categorization criteria to categorize all of the selected clone sets successfully because of the ambiguous of criteria found in Step 1.

Step 6: Refinement of clone set categorization

We categorized selected clone sets again based on the refined criteria. In Step 3, practitioners can go back to Step 3 then select other clone sets when they believe that refined criteria are insufficient and have enough time. It is describe characteristics of one code clone to combine categorizations of every criterion. Following is described the example of the characteristics of one code clone based on above criterion.

- Difference among code fragments in a clone set
  - type name
- Class hierarchy relationship among code fragments in a clone set
  - code fragments belong to classes have direct common parent classes
- Granularity of code fragments in a clone set
  - method-level clone
- Existence of jump statements

– A break statement is involved.

- Existence of instanceof statements

  – An instanceof statement is involved.

3. DEVELOPMENT OF PATTERNS FOR CLONE REFACTORING

3.1 Problems

- Lack of reusability of patterns (Problem A)

  –Refactoring patterns should be written by semi-formal form for reusability.

- Lack of the comprehensive correspondences (Problem B)

  –We would like to suggest some patterns to every characteristics of code clone, so patterns must be corresponding to every categorization and categorizations include all clone characteristics.

- Lack of validity of patterns (Problem C)

  –Refactoring patterns must be evaluated by reviews or tests because invalid patterns for software qualities are meaningless.

3.2 Solution

3.2.1 Method to develop refactoring patterns for clone refactoring

Step A: Selection of one category having clone sets

We selected one of clone set categories found in Section 2. Selected category should have several clone sets detected from OSS projects.

Step B: Writing refactoring pattern for selected category

We wrote a refactoring pattern for selected categories based on our refactoring experiences while being careful in consistency of external behavior of code. The example of refactoring pattern described in Section 4.

Step C: Application of refactoring pattern

We performed refactoring using written pattern. In this refactoring, we applied refactoring to clone sets in selected categorization.

Step D: Refinement of written pattern

In Step C, we often needed to additional modifications that are unmentioned in written pattern. Therefore, we added those modifications to written pattern, and then go back to Step C for trying refined pattern.

Toward formalizing of pattern description (Problem A), we defined the form of description of refactoring pattern having following four items, "Summary", "Characteristics of code clone", "Refactoring procedures", and "Concrete example". The item of "Characteristics of code clone" has descriptions of clone categorization per every defined code clone criteria. We describe refactoring patterns following the above form in Step B.

Toward consistent association (Problem B), we select categorizations unselected yet in turn in Step A.

Toward assurances of availability of suggested patterns (Problem C), we check that not be changed its external behavior (e.g., Input data and Output data) before of applying refactoring and after. Then, patterns are reviewed by expert on code clone. At the same time, patterns are experimented on the time or easy of application in Step D.

4. EXAMPLE OF REFACTORING PATTERNS

In this section, we describe an example of refactoring pattern that we have developed. Followings are the example of refactoring pattern.

- Summary
  Pull up a method of code clones having user-defined data with a difference to a parent class.

- Characteristics of code clones
  - Difference among code fragments in a clone set
    - user definition data
  - Class hierarchy relationship among code fragments in a clone set
    - code fragments belong to classes have a common parent class
  - Granularity of code fragments in a clone set
    - method-level clone
  - Existence of jump statements
    - none
  - Existence of instanceof statements
    - none

- Refactoring procedures
  1. Pretreatment
     (a) Comment out two methods in a relationship of code clone
  2. Declare a new method in a common parent class
     (a) Set the return value as void
     (b) Give a suitable name to the Java method according role of an original method which be extracted
     (c) Set variables which declaration is outside of the original method as the arguments of the new method
     (d) Organize the new method (e.g., brace addition), compile
     (e) Describe the method processing unit of the original method which be extracted into the method processing unit of new method in parent class
     (f) Define a common name of an user-defined data with a difference
     (g) Replace the defined common name the user-defined data with a difference
     (h) Add the common name to argument

(i) Compile the parent class and test using test cases to assure of consistency with extracted method.

3. Remove the code clones among brother classes and add a call statement to the Java method
4. Compile all classes and test using test case to assure of consistency with previous external behavior

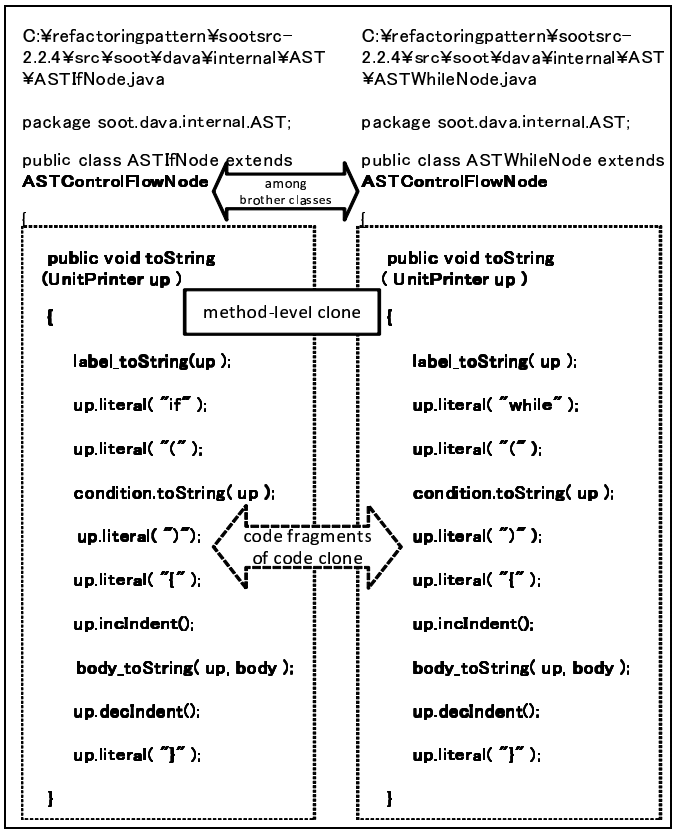

Figure 5: Example of target code clones for refactoring

Figure 5 shows an example of code clone for applying refactoring pattern. Characters in bold describe code fragments of code clone. Figure 6 and 7 describe the process of applying example of refactoring pattern. In Figure 6, a new method is introduced in the parent class. The method has two arguments which scope is outside of the original method. In Figure 7, a new method includes a processing unit of original method and a variable name of substitution.

5. SUMMARY AND FUTURE WORK

In this paper, we proposed a method to create fine-grained categorization for code clone, then describe a method to develop refactoring pattern based on fine-grained clone categorization. Finally, we introduce an example of our pattern for clone refactoring.

Currently, we are refining our clone categorization using actual clone sets detected from several OSS projects. Also, we are planning to develop refactoring patterns based on refined categorization, and propose a collection of clone refactoring pattern.

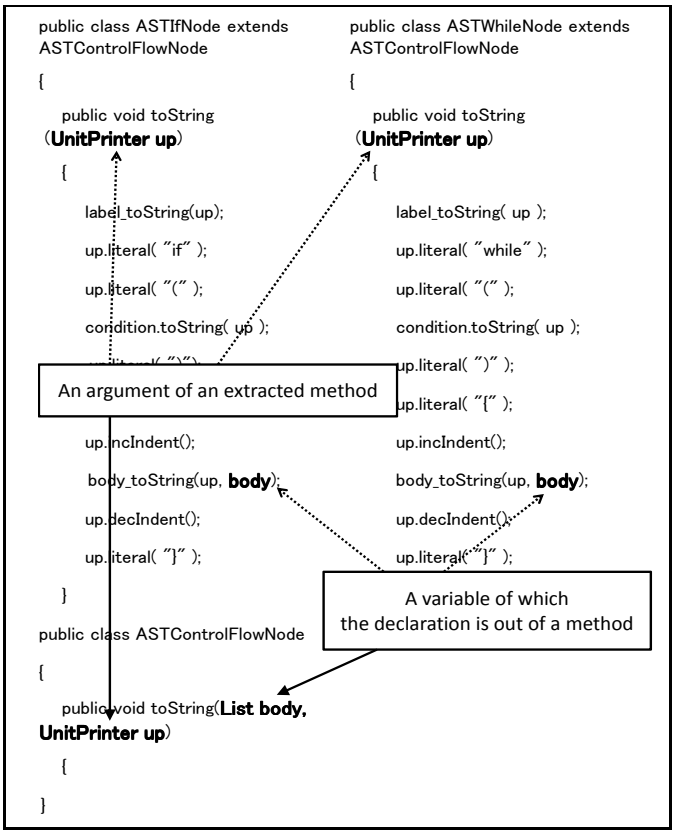

Figure 6: Application of developed refactoring pattern

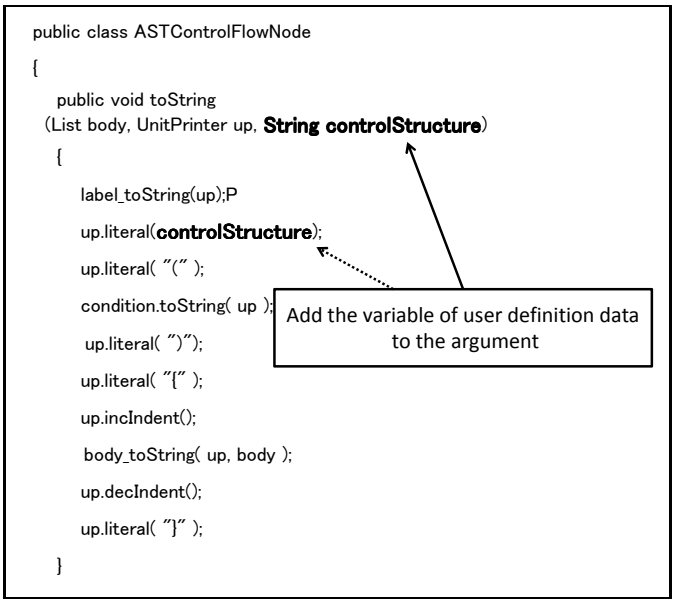

Figure 7: Derived code in a parent class after refactoring

Acknowledgments

We express our great thanks to Dr. Yann-Gael Gueheneuc for helpful comments on earlier revisions of this paper. This work is partially supported by JSPS, Grant-in-Aid for Scientific Research (A) (21240002) and Grant-in-Aid for Research Activity start-up (22800040).

6. REFERENCES

[1] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. IEEE Trans. Softw. Eng., 28(7):654–670, 2002.

[2] B. Laguë, D. Proulx, J. Mayrand, E. M. Merlo, and J. Hudepohl. Assessing the benefits of incorporating function clone detection in a development process. In Proc. of ICSM '97, pages 314–321, 1997.

[3] Z. Li, S. Lu, S. Myagmar, and Y. Zhou. CP-Miner: Finding copy-paste and related bugs in large-scale software code. IEEE Trans. Softw. Eng., 32(3):176–192, 2006.

[4] Y. Higo, Y. Ueda, S. Kusumoto, and K. Inoue. Simultaneous modification support based on code clone analysis. In Proc. of APSEC 2007, pages 262–269, 2007.

[5] A. Zeller. Why Programs Fail. Morgan Kaufmann Pub., 2005.

[6] Miryung Kim, Vibha Sazawal, David Notkin, and Gail Murphy. An empirical study of code clone genealogies. In Proc. of ESEC/FSE 2005, pages 187–196, 2005.

[7] Cory Kapser and Michael W. Godfrey. "Cloning considered harmful" considered harmful. In Proc. of WCRE 2006, pages 19–28, 2006.

[8] M. Fowler. Refactoring: improving the design of existing code. Addison Wesley, 1999.

[9] J. Kerievsky. Refactoring to Patterns. Addison Wesley, 2004.