International Journal of Computer Communication and Information System (IJCCIS)
– Vol2. No1. ISSN: 0976–1349 July – Dec 2010

Method Level Detection and Removal of Code Clones in C and Java Programs using Refactoring

[1]Mrs. E.Kodhai , [2]V.Vijayakumar, [3]G. Balabaskaran, [4]T.Stalin, [5]B.Kanagaraj

[1]Sri Manakula Vinayagar Engineering College,Department of Information Technology, Puducherry, India
Email: baas.1989@gmail.com
[2,3,4,5]Sri Manakula Vinayagar Engineering College,Department of Information Technology, Puducherry, India
Email: Viji_kum_truth@yahoo.com, kodhaiej@yahoo.co.in, stalin_waiting4u@yahoo.com

***Abstract* - Clone detection and refactoring is the major role in software maintenance and evaluation. A well-known bad code smell in refactoring and software maintenances is duplicated code, or code clones. A code clone is a code fragment that is identical or similar to another. Unjustified code clones increase code size, make maintenance and comprehension more difficult, and also indicate design problems such as lack of encapsulation or abstraction. This paper proposes to automatically detecting code clones in c/java programs, underlying a collection of refactoring to support user-controlled automatic clone removal, and examines their application in substantial case studies. Both the clone detector and the refactoring will be done using new refactoring methods.**

***Index terms* - Detection, Refactoring, Duplicated code**

1. INTRODUCTION

The software comprises both programs and data. The paper mainly contributes on the process of software evolution and maintenance. In software engineering the software maintenance is the delivery to correct faults to improve performance or other attributes adapts the product to a modified environment. Software evolution is the process which refers to the process of developing software initially and then repeatedly updating it for various reasons.

Code clones

A code clone is a pair (or set) of code fragments in source files of a software product.

Clone detection

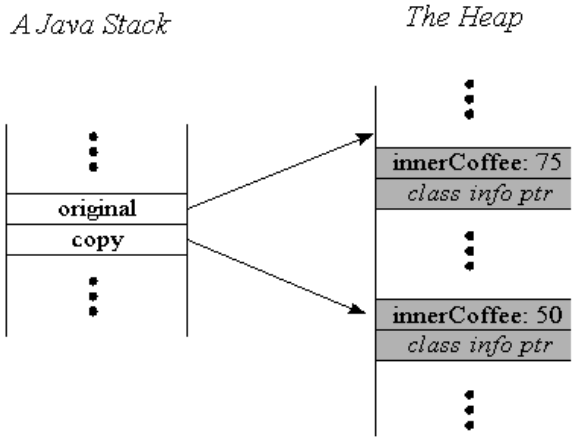
Common terminologies for the clone relations between two or more code fragments are the phrases clone pair and clone class. A clone pair is a pair of code fragments which are identical or similar to each other; a clone class is the maximal set of code fragments in which any two of the code fragments form a clone pair. In this paper, we distinguish the following four types of clones. All these four types of clones ignore variations in literals, layout and comments.

Type 1: Identical code fragments.

Type 2: Code fragments that are identical after consistent (i.e. semantic-preserving) renaming of variable names.

Type 3: Code fragments that are identical after renaming all variable names to the same name.

Type 4: Code fragments that are identical after renaming all function names and variable names to the same name, respectively



Obviously, these four types of clones satisfy a subset relation, i.e. clones of Type i(i=1;2;3) form a subset of clones of Type (i+1).Among the four types of clones, Type 1 and Type 2 represent the clones that are most suitable for automatic clone removal because of the semantic equivalence between cloned code fragments, and they are also the kinds of clones that are reported by the Wrangler clone detector. Type 3 and Type 4 clones are not suitable for mechanical removal, but they somehow reveal structure-level duplication, and are obtainable from the intermediate results of the Wrangler clone detector.

2. REFACTORING

Refactoring is the process of changing the structure of a program while maintaining all of its functionality. There are many types of refactoring that you can do such as renaming a class, changing a method signature

or extracting some code into a method. With each refactoring, you carry out a number of steps that keep your code consistent with the original code.

Why Refactoring is Important

When refactoring by hand, it is easy to introduce errors into your code such as spelling mistakes or missing a step in the refactoring. To prevent and quickly fix these errors, thorough testing should be performed before and after each refactor. You may wonder if refactoring is worth going through all this.

There are several reasons why refactoring should be used. You may want to update a program that is poorly coded. Perhaps none of the original design team is present and no one on the current design team understands the code. In order to update it, you will have to redesign and restructure the program to fit what you want it to do. Another reason is that you may want to add a feature that the original design cannot accommodate. In order to add it, you will have to restructure the code. The third reason is that an automatic refactoring tool, such as the refactoring in Eclipse, can generate code for you.

By using refactoring, you can easily change the structure of a program to what makes logical sense while rewriting code as little as possible and still keeping its functionality. If refactoring is used on a regular basis to constantly keep a good structure, less time will be needed to fix any bugs and it will be easy to add new code to the design.

Types of Refactoring

The first type contains refactoring that change the physical structure of the code and classes such as Rename and Move. The second type contains refactoring that change the code structure on a class level such as Pull Up and Push Down. The third type contains refactoring that change the code within a class such as Extract Method and Encapsulate Field. The sections and their refactoring are shown below.

Type 1 – Physical Structure

• Rename
• Move
• Change Method Signature
• Convert Anonymous Class to Nested
• Convert Nested Type to Top Level (Eclipse 2 only)
• Move Member Type to New File (Eclipse 3 only)

Type 2 – Class Level Structure

• Pull Up
• Extract Interface
• Generalize Type (Eclipse 3 only)
• User Supertype Where Possible

Type 3 – Structure inside a Class

• Inline
• Extract Method
• Extract Local Variable
• Extract Constant
• Introduce Parameter (Eclipse 3 only)
• Introduce Factory (Eclipse 3 only)
• Encapsulate Field

3. APPROACH

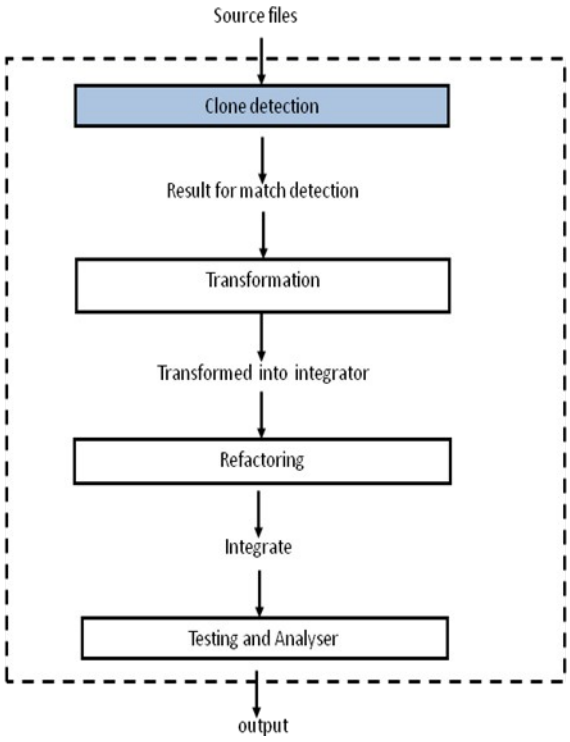
Detecting Functions

For detecting functions in the source file the following information is need. They are beginning and end of the body, beginning of the declaration. The two important things which is necessary to calculate the similarity between the functions are as follows,

- Compare function signatures
- Name of the function

The generation of the name of the function is impossible because the conditional compilation may change the location of a function depending on compile-time switches.

The first approach in this paper is to detect the possible clones in the source file and preserve it for future use. For detecting the clones first have to detect all the possible functions by using the necessary functions. For detecting the functions the transform code is converted into the preprocessed form. After that organize the code and extract that organized code. After extracting the code the code is split led into the number of tokens for comparing and detecting the similarity.



CLONE ELIMINATION

After detecting functions or method in c/java programs, to evaluate clone elimination by means of refactoring, we underwent the process of removing the clones.

In this system, we are taking the methods.

The methods are:

- Rename method.
- Add parameter.
- Replace constructor with factory methods.
- Replace parameter with explicit methods.
- Remove setting method.

The main approach in this paper into detect the possible clones and removing the clones using refactoring methods which is not supporting for existing systems.

4. CONCLUSION

In this paper, we have presented clone detection which makes use of detecting the clones to improve performance and efficiency, and a collection of refactoring which together help to remove clones from code under the user's control. In main approach in our system is both the detection and removal are done in the C and JAVA language programming.

ACKNOWLEDGEMENT

The authors would like to thanks Mr. Martin Fowler and Kent Beck(Refactoring: Improving the Design of Existing code) and Simon Thompson (Computing Laboratory, University of Kent) for their support.

REFERENCES

[1] Huiqing Li, Simon Thompson, Clone Detection and Removal for Erlang/OTP within a Refactoring Environment, PEPM'09, January 19–20, 2009, Savannah, Georgia, USA.

[2] Martin Fowler, Kent Beck (Contributor), John Brant (Contributor), William Opdyke, don Roberts, Refactoring: Improving the Design of Existing Code**,** Addison-Wesley Longman Publishing Co., Inc., Boston, MA,USA, 1999. ISBN0-201-48567-2.

[3] H. Roy and R. Cordy. A Survey of Software Clone Detection Research.Technical report, School of Computing, Queen's UniversityatKingston,Ontario, Candada, 2007.

[4] Eytan Adar. GUESS: a language and interface for graph exploration. In *Proceedings of the 2006 Conference on Human Factors in Computing Systems (CHI'06), pp. 791-800,* Montreal, Quebec, Canada, April 2006. (PDF)Eytan Adar and Miryung Kim. SoftGUESS: Visualization and Exploration of Code

[5] Clones in Context. In *the proceedings of the 29th International Conference on Software Engineering (ICSE'07), Tool Demo, pp.762-766, Minneapolis, MN, USA, May 2007* .

[6] R. Agrawal and R. Srikant. Mining Sequencial Patterns. In *Proceddings of the 11th In-fernation Conference of Data Engineering (ICDE'95), pp. 3-14, Taipei, Taiwan, March* 1995.