ISSN 2395-1621

Refactoring of Code Clone Using Preconditioned Structure

[#1]Chaitanya S.Kulkarni, [#2]Dr.S.D.Joshi

[1]ckulkarni47@gmail.com,
[2]sdj@live.in

[#1]Shri Jagdishprasad Jhabarmal Tibrewala University,
Rajasthan, India.

ABSTRACT

On account of maintain and development of source code, many examinations have demonstrated that the copied code or code clone in programming are conceivably destructive. In spite of the fact that this is the major issue in programming, through refactoring, there is smidgen bolster for dispensing with programming clones. An exceptionally difficult issue is unification and consolidating the copied code, particularly after starting prologue to the product clone they are experiencing the few adjustments in them. This paper exhibits an approach in which a couple of clone is consequently surveyed and without changing the program conduct that clone combine is re-considered securely. The distinctions display in the clones is inspected by this approach and those are securely parameterized without event of any reaction.

ARTICLE INFO

Article History

Received: 1st December 2017

Received in revised form :

1st December 2017

Accepted: 4th December 2017

Published online :

4th December 2017

Keywords: Code Refactoring Code, Code Clone, Program Structure Tree.

I. INTRODUCTION

In software systems, it has been recognized that the code duplication is serious problem. One of the major activities for clone management is the refactoring of code clones that aims to minimize the number of clones in the source code Refactoring is a process of transforming the program without affecting the behavior and semantics and to improve the quality. The refactoring process also involves in the removal of duplication and simplification of unclear code. There is no study investigating the refactorability of software clones. What portion of the clones detected by tools can be actually refactored? Additionally, there is a lack of tools that can automatically analyze software clones to determine whether they can be safely refactored without changing the program behavior.

Change-proneness, and change propagation have been investigated empirically by several researchers. There is a lack of tools which can automatically analyze software clones to determine whether they can be safely re-factored without changing the behavior of program. One of the important but missing features from clone management is re-factorability analysis. When the developers are interested in finding refactoring opportunities for duplicated code it could be used to filter clones that can be directly re-factored. This is the way by which maintainers can focus on parts of the code that can immediately benefit from refactoring, and thus causes improvement in maintainability. This paper presents an approach that takes two clone fragments as input which are detected from any tool and it applies following three steps to determine whether they can be re-factored without any side effects or not.

Step 1: in this step, this approach In the first step, it tries to find identical control dependence structures within the clones that will serve as candidate refactoring opportunities.

Step 2: In the second step, it applies a mapping approach that tries to maximize the number of mapped statements and at the same time minimize the number of differences between them.

Step 3: Finally, in the last step, the differences detected in the previous step are examined against a set of preconditions to determine whether they can be parameterized without changing the program behavior.

This technique supports the refactoring of Type-1, Type-2, type-3 clones

II. BASIC APPROACH

Following are the two core program structures that are used in this approach:

a) Program Structure Tree.

b) Program Dependence Graph.

a) Program Structure Tree

The Program Structure Tree (PST) [5] was presented by Johnson et al. as a various leveled portrayal of program structure and this structure depends on single-section single-exit (SESE) locales of the control stream diagram. The settling relationship of SESE areas and chains of successively created SESE locales are caught by basically PST.

b) Program Dependence Graph

The Program Dependence Graph (PDG)[6] is a coordinated diagram which comprises of various edge sorts. In PDG the hubs signifies the announcements of a capacity or technique, and the edges indicates control and information stream conditions between proclamations. In this approach PDG portrayal is utilized as a part of two ways. In first way, the composite factors are presents which speak to the condition of articles which are alluded in assortment of strategy and it additionally makes information conditions for these factors. In second way, two more sorts of edges are included the PDG, which are useful in the examination of preconditions. These two sorts of edges are: hostile to conditions and yield conditions [3].

III. IMPLEMENTED WORK

Though duplication code having large importance in software systems. Many research studies have proven that in the maintenance and evolution of source code, clones can be potentially harmful. Though this is the serious problem in software, through refactoring, there is little bit support for eliminating software clones.

IV. MOTIVATION

There is no study investigating the refactorability of software clones. What portion of the clones detected by tools can be actually refactored? Additionally, there is a lack of tools that can automatically analyze software clones to determine whether they can be safely refactored without changing the program behavior.

V. OBJECTIVE

Differentiating classes from given input file which have multiple classes.

A. Detecting similar code parts.
B. Finding out clones from detected similar code parts.
C. Detecting code clones according to types.
D. Finding refactorable clones from detected clones.
E. Generate Report.

VI. PROPOSED SYSTEM

This technique is for the refactoring of software clones in Java programs.

Here are the three major steps for assessing the refactorability in this approach:

Step 1: In the first step, it tries to find identical control dependence structures within the clones that will serve as candidate refactoring opportunities.

Step 2: In the second step, it applies a mapping approach that tries to maximize the number of mapped statements and at the same time minimize the number of differences between them.

Step 3: Finally, in the last step, the differences detected in the previous step are examined against a set of preconditions to determine whether they can be parameterized without changing the program behavior. This technique supports the refactoring of Type-1, Type-2, type-3 clones Statement mapping [1]:

The statements extracted from the previous step within the subtree pairs are mapped in a divide-and-conquer fashion. It takes advantage of the identical nesting structure between the isomorphic subtrees, the global mapping problem is divided into smaller subproblems. The corresponding Program Dependence subgraphs are mapped by applying a Maximum Common Subgraph (MCS) algorithm for each sub-problem [10].

These subsolutions are combined to give the global mapping solution at the end.

VII. INVESTIGATION OF PRECONDITION

A set of preconditions regarding the preservation of program behavior is examined based on the differences between the mapped statements in the global solution, as well as the statements that may have not been mapped. If no preconditions are violated, then the clone fragments corresponding to the mapped statements can be safely refactored, and thus those are considered to be refactored.

Following Fig shows the steps mentioned above:

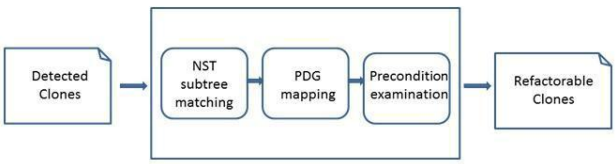

Fig 1. Schematic representation of Refactored Code Clone

VIII. CONCLUSION

The Program Dependence Graph (PDG)[6] is a coordinated diagram which comprises of various edge sorts. In PDG the hubs signifies the announcements of a capacity or technique, and the edges indicates control and information stream conditions between proclamations. In this approach PDG portrayal is utilized as a part of two ways. In first way, the composite factors are presents which speak to the condition

of articles which are alluded in assortment of strategy and it additionally makes information conditions for these factors. In second way, two more sorts of edges are included the PDG, which are useful in the examination of preconditions. These two sorts of edges are: hostile to conditions and yield conditions.

REFERENCES

1) NikolaosTsantalis, Member, IEEE, DavoodMazinanian, and GiriPanamoottil Krishnan "Assessing the Refactorability of Software Clones", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. XX, NO. XX, MONTH 2015.

2) G. P. Krishnan and N. Tsantalis, "Unification and refactoring of clones," in Proceedings of the IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE),2014 Software Evolution Week, 2014, pp. 104–113.

3) C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," Science of Computer Programming, vol. 74, no. 7, pp. 470 – 495, 2009.

4) C. Roy, M. Zibran, and R. Koschke, "The vision of software clone management: Past, present, and future (keynote paper),"in Proceedings of the IEEE Conference on Software Maintenance,Reengineering and Reverse Engineering, 2014 Software Evolution Week, 2014, pp. 18–33.

5) R. Johnson, D. Pearson, and K. Pingali, "The program structure tree: Computing control regions in linear time," inProceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, 1994, pp. 171–185.

6) J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," ACM Transactions on Programming Languages and Systems, vol. 9, no. 3, pp.12-13

7) James J. McGregor, "Backtrack search algorithms and the maximalcommon subgraph problem", Software: Practice and Experience, Volume 12, Issue 1, Jan 1982, pp. 23-34.

8) Yu Wang, "A novel efficient algorithm for determining maximumcommon subgraphs", Ninth International Conference on Information Visualization, July 2005, pp. 657-663.

9) Mohammad Alshayeb, "Empirical Investigation of RefactoringEffect on Software Quality", Volume 51, Issue 9, 2009, Elsevier, pp.1319-1326.

10) S. A. M. Rizvi, Zeba Khanam, "A methodology for refactoringlegacy code", ICECT 2011, pp. 198-200.

