ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 6, June 2016

An Effective Approach to Find Refactoring Opportunities for Detected Code Clones

Prajakta Ashtaputre [1], Chaitanya Kulkarni [2], Yogesh Lonkar [3]

P.G. Student, Department of Computer Engineering, JSPM Narhe, Pune, Maharashtra, India[1]

Assistant Professor, Department of Computer Engineering, JSPM Narhe, Pune, Maharashtra, India[2]

Assistant Professor, Department of Computer Science and Engineering, KEC, Pandharpur, Maharashtra, India[3]

ABSTRACT: In the evolution and maintenance of software clone codes that is copied code is potentially harmful. This is very serious problem in software evolution and maintenance. But by using concept refactoring, there is little bit support for eliminating software clones because major challenging problem is nothing but unification and merging of the duplicated code. This paper describes an approach which takes input as a source code from which it will detect the clones. Then it will checks that which clones are refactorable safely without changing the program behavior and which are not. Existing approach was able to check it for clone pair i.e. only two clone fragments which are detected as clones. So here is the approach which overcomes this drawback of existing system. One of the advantages of this approach that it is having less or let's say almost negligible computational cost.

KEYWORDS: Code duplication, Software clone management, Clone refactoring, Re-factorability assessment.

I. INTRODUCTION

Reuse of the code fragments by copy-pasting with or without small changes is a common activity in software development. Most of the developers use this idea to reduce their work. Though it is reducing developers work it is having bad effect on maintenance and evolution of the software system[1]-[2]. The presence of these code clones may decrease the design structure as well as software quality such as changeability, maintainability, and readability, and this is one of the major reasons of increasing the maintenance cost[3].

In last few years different techniques are developed to detect and analyze the duplicated code by different research communities [4]. Now the recent research study is focused on clone management activities [5]. These clone management activities includes : analyze the consistent of modifications to code clones, incrementally updating code clone groups as the project evolves, tracing the code clones in the history of a project, and prioritizing the refactoring of code clones. In addition to the development work mentioned above, some of the things have been investigated by the researchers as: software defects, error-proneness due to inconsistent updates, change-proneness, the effect of duplicated code on maintenance effort and cost, and change propagation.

There is a lack of tools which are able to automatically analyze software code clones to check whether they can be safely re-factored or not without changing the behavior of program. Refactorability analysis is the one of the important but missing features of the clone management. When the developers are having interest to find the refactoring opportunities for duplicated code, it could be useful in filtering the clones that can be directly re-factored. Maintainers can use this way to focus on parts of the code that can give immediate benefit from refactoring, and this causes improvement in maintainability of software.

The work in this paper is divided in two stages :1)clone detection, 2) refactorability analysis. This approach takes input as source code file. At first stage it will detect the clones in that source file or source code. At the second stage it will find which clones are refactorable and which clones are not refactorable. The existing approach was able to check the refactorability opportunity for clone pair that is only twu code fragments which are detected as clones.

All these things are done for determining whether without changing behavior of program clones can be parameterized or not.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 6, June 2016

II. RELATED WORK

N. Tsantalis, D. Mazinanian, and G. P. Krishnan [1] had given the existing system of this proposed approach. In the existing system they had extended their work form the approach of paper "Unification and refactoring of clones". In their approach they have added few preconditions and replaced Control Dependence Tree with Program Structure Tree (PST).

The existing system of proposed work is extension of research work of G. P. Krishnan and N. Tsantalis [2]. They had given the system which was responsible for automatically assess the pair of clones. And after assessing, checking whether that assessed pair is safely refactorable or not and that is also without changing the program behavior.
Here some preconditions are checked to check whether they can be refactor or not. If no preconditions are violated then those are refactorable clones.

In research work of C. K. Roy, J. R. Cordy, and R. Koschke [4], they did the comparison of different techniques which are available for clone detection such as lexical approach, semantic approach, textual approach, and metric based approach. And also they have compared and evaluated different tools available for clone detection. Additionally they have introduces the four types of clones : Type-I, Type-II, Type-III, Type-IV.

The Survey did by C. Roy, M. Zibran, and R. Koschke [5] on clone management includes in-depth investigation of available clone management activities. Clone management activities include refactoring, tracing, cost benefit analysis beyond the detection and analysis.

Program Structure tree (PST) is introduced by R. Johnson, D. Pearson, and K. Pingali [6] as a hierarchical representation of program structure. This structure is depends on the single-entry single-exit (SESE) regions of the control flow graph (CFG). The PST is useful for enhancing the performance of program analysis algorithms.

Program Dependence Graph (PDG) is introduced by J. Ferrante, K. J. Ottenstein, and J. D. Warren [7]. PDG is directed graph which consists of multiple edge types. In PDG, the statements of methods or a function are denoted by nodes. Control as well as data flow dependencies between the statements are denoted by the edges. In a program, For each operation, data and control dependence are made explicit by PDG. In a program, the control flow relationships are represented by the control dependence, also the relevant data flow relationships are represented by data dependences.

The Abstract Syntax Tree (AST) concept is introduced by R. Koschke, R. Falke, and P. Frenzel [8]. In their research work they have given that in abstract syntax trees how suffix trees are used to find duplicates. Their approach was responsible to find syntactic duplicates in linear time and space.

An approach for detecting differences across multiple clone instances is proposed by Y. Lin, Z. Xing, Y. Xue, Y. Liu, X. Peng, J. Sun, and W. Zhao [9]. Their approach was Multi- Clone-Instances Differencing (MCIDiff). In this approach, each clone instance in a clone group is parsed into a sequence of tokens. MCIDiff was responsible for computing the Longest Common Subsequence (LCS). This is for determining the differential ranges across the clone instances. Finally it was producing a list of differential multi-sets of tokens as output, which was summarizing the differences those were found in gapped and parameterized clones.

A new method for merging the code clones have been introduced by Y. Higo, S. Kusumoto, and K. Inoue [10] in their research work. This method consists of two phases. First phase was responsible for, from the source code, quickly detection of refactoring oriented code clones. And in the second phase they have measured the metrics which were indicating the manner in which refactoring oriented code clones should be merged. For implementing the method mentioned above one software tool named Arise is available. By using this Aries software tool in the refactoring process, software system maintainers can easily get which and how code clones can be merged.

D. Speicher and A. Bremm [11] have explained the problems which are involved in finding a sequence of changes for clone removal and also they have suggested viewing this problem as stepwise unification process of the clone instances. Therefore this problem can be solved by technique backtracking over the possible unification steps.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 6, June 2016

III. EXISTING SYSTEM

Existing technique was developed for the refactoring of clones in Java programs.

Two different forms of input are processed by this existing approach, and those are :

1) Two code fragments which are declared as clones by any clone detection tool within the body of the same method, or different methods.

2) Two method declarations are considered to be duplicated, or it may contain duplicate code fragments somewhere inside their bodies.

Two code fragments or entire methods are given as input to this approach which are detected as clones. For determining whether this clone pair or a part of code can be safely refactor or not, without disturbing its behavior. Following are the three step procedure which is applied to it :

Step 1 : First step of existing approach is responsible to find the identical nesting structures within the input clones. These identical nesting structures are further treated as candidate refactoring opportunities.

Step 2 : Second step of existing approach is responsible for mapping the statements. Here, in this step, a mapping approach is applied. This mapping approach tries to maximize the number of mapped statements. At the same time it tries to minimize the number of differences between the statements.

Step 3 : Finally, in the third step, the differences detected in the previous step are examined against a set of preconditions. This is carried out to determine whether they can be parameterized without changing the program behavior or not.

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

Following Fig shows the stepwise procedure mentioned above :

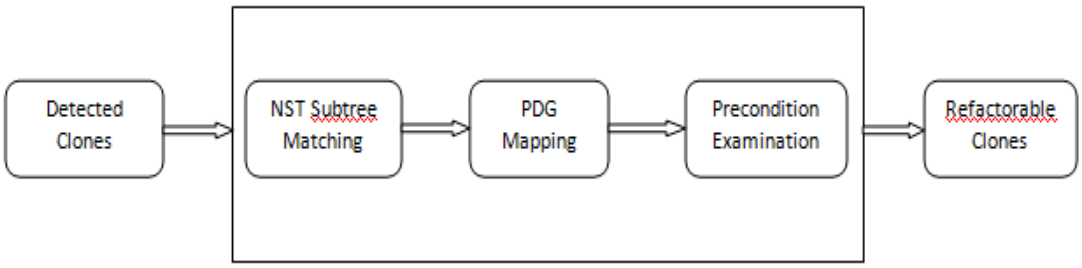

Fig. 1. An overview of the existing refactorability analysis approach.

IV. PROPOSED SYSTEM

The working of the proposed system is divided in two stages :1)clone detection, 2) refactorability analysis. This approach takes input as source code file. At first stage it will detect the clones in that source file or source code. At the second stage it will find which clones are refactorable and which clones are not refactorable.

Stage 1: Clone Detection : The input to this stage will be a source code file. And the output will be the clones detected. There are several clone detection techniques are available. Here we will use Abstract Syntax Tree (AST) clone detection technique to detect the clones in the given source file. These clones detected in this stage will be given as input to the stage 2, which is responsible to find out refactorable clones.

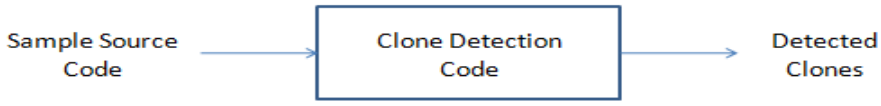

Fig 2. Stage 1 : Clone Detection

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 6, June 2016

Stage 2: Refactorability Analysis : Input to the stage 2 is output of the stage 1 i.e. detected clones or let's say group of code fragments detected as clones. Three steps are there to be carried out to check refactorable opportunities.

Step 1: Nesting Structure Matching: In this step, nesting structure of the input clone fragments is analyzed. This is useful in finding maximal isomorphic sub-trees. It is assumed that the code fragments can be unified only if they are having an identical nesting structure. Each matched sub-tree will be further investigated as a separate clone refactoring opportunity in the further steps.

2) Statement Mapping: The statements which are extracted from the previous step within the sub-tree are mapped in a divide-and-conquer fashion. It takes advantage of the identical nesting structure between the isomorphic sub-trees. Here the global mapping problem is divided into smaller sub-problems by divide-and-conquer method. By applying a Maximum Common Subgraph (MCS) algorithm for each sub-problem the corresponding Program Dependence sub-graphs are mapped. At the end these sub-solutions are combined to give the global mapping solution.

3) Precondition Examination: A set of preconditions which is regards for preserving the 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 preconditions are violated, then the clone fragments cannot be refactored safely. If its not the case then the corresponding mapped statements can be safely refactored.

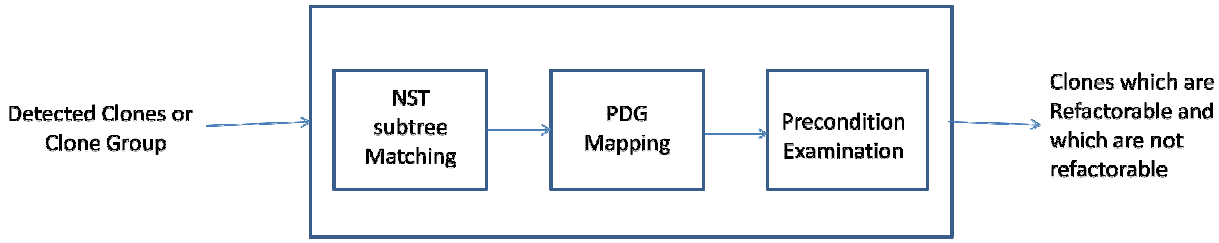

Fig 3. Stage 2 : Refactorability Analysis.

V. CONCLUSION

This approach is responsible to introduce an important and missing feature in clone management i.e. refactorability analysis which was unsupported previously. To achieve this goal, here is a technique which first detects clones and then matches the statements of the clones in such a way which is responsible for minimizing the number of differences between these statements. After this, these differences are examined against a set of preconditions. This is to determine whether they can be parameterized without changing the program behavior. The proposed system having the advantage over the existing that it will check the group of clones instead of only a pair of clones to find which are refactorable and which are not.

REFERENCES

[1] Nikolaos Tsantalis, Member, IEEE, Davood Mazinanian, and Giri Panamoottil 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. 104113.

[3] Balwinder Kumar, Dr. Satwinder Singh "Code Clone Detection and Analysis Using Software Metrics and Neural Network-A Literature Review", International Journal of Computer Science Trends and Technology (IJCST) – Volume 3 Issue 2, Mar-Apr 2015.

[4] 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.

[5] 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. 1833.

[6] R. Johnson, D. Pearson, and K. Pingali, "The program structure tree: Computing control regions in linear time", in Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, 1994, pp. 171185.

[7] 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. 319349, Jul. 1987.

[8] R. Koschke, R. Falke, and P. Frenzel, "Clone detection using abstract syntax suffix trees,in Proceedings of the 13th Working Conference on Reverse Engineering, 2006, pp. 253262.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 6, June 2016

[9] Y. Lin, Z. Xing, Y. Xue, Y. Liu, X. Peng, J. Sun, and W. Zhao, "Detecting differences across multiple instances of code clones", in Proceedings of the 36th International Conference on Software Engineering, 2014, pp. 164174.

[10] 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. 435461, 2008.

[11] D. Speicher and A. Bremm, "Clone removal in java programs as a process of stepwise unification" , in Proceedings of the 26th Workshop on Logic Programming, 2012.