DETECTION OF SOFTWARE REFACTORABILITY THROUGH SOFTWARE CLONES WITH DIFFRENT ALGORITHMS

Ritika Rani[1],Pooja sachdeva[2]
Department of computer science & engineering himalayan group of professional institutions Kal Amb, Distt- Sirmour (India).

ritu.bagga123@gmail.com[1] , **Poojasachdeva1886@gmail.com[2]**

Abstract- In software programs if the code is similar to each other or we can say if the code is copied then it is called clones, we can also used the term of replication or redundancy for it. Every researcher have purposed a different definitions of clones according to him .we also use the term of duplicate code for it.

Through the occurrence of clones the program efficiency is to be decreases. it can also effects on program cost and maintenance . The code redundancy can be solved by some techniques. we can separately functionalized the clones into a single unit.
Several studies are to be defined for the prevention and detection of a code clone. We have also need to prevent a unification and refactoring of a software clones. And sometimes programmers need to manually understand the clones by the use of clone detection tools, decide how they should be refectories. This obvious gap between the clone detection tools and the clone analysis tools, makes the refactoring and the programmers refactoring the duplicate codes. In this thesis, an approach for the refactoring through different algorithms for unification In software replication of code or we can say clone that have t be overcomes the limitations of previous methods. This technique is used to prevent and solve the raised mismatched between the clones. it can also find a mapping between the similar statements. We have also defined preconditions in particular order to explain whether the duplicated code safely refectories to manage the behavior of existing code.

———————————— ◆ ————————————

Introduction

In This thesis presents a methods for removing the unification and refactoring through different algorithms in java programming. And also used a art of state techniques. The proposed approach takes as entire program or parts of a the codes that have been detected by a specific tool. And also determines whether the clones. And try to fully refectories. The three main steps involved in the process are the following. In the first step, it finds the structures of control dependency within the clones. And now in second step, prevent the matched statements also used to remove the mismatching at the same step. And in the last step, again define the mismatched conditions again and also define whether the program behavior is to be changed or not.
In this thesis the technique is to be only used for a first three types of clones. The technique is compared with Codepro , and a art of state tool is to be used. The same process is to be carried out until the fair results. And the results shows that the our technique is more efficient then codepro tool in java programming .

Related Work

The extraction of code clone differences is an important step toward the process of refactoring code duplicates. This technique is not only used for the detection or prevention of software clones it can also used for a evolution of some another software applications. data copy detection, source code retrieval. The Program Dependence Graphs and their applications, the next two current approaches for code matching and discusses the art of state techniques toward code clone refactoring. We will analysis the mismatching is not be explored and not to be optimal and also face some scalability problems.

Clone Refactoring Techniques

Balazinska et al.defines the code clone differences and perform advanced code clone analysis and provide the a solution to programmer to solve refactoring. .In this technique compare code fragments based on the Pattern Matching algorithm.
The proposed algorithm aligns syntactically unstructured entities and finds the distance of the two code fragments. The solution is to be used to minimize the number of tokens is to be inserted or deleted to change the code fragments into another fragment . However, this overall distance cannot be guaranteed as minimal as it tries to find optimal values at node level without considering the hierarchical structural differences at a higher level. The differences are expressed as programming language entities easily understandable by a programmer. This is done by projecting the tokens forming the differences onto the corresponding AST elements. The differences are also categorized based on the role in refactoring. The categories are:

1. superficial differences such as names of local variables which do not affect the behavior of methods

2. differences which affect the methods such as return value, access modifiers, thrown exceptions etc.

3. differences affecting the types of parameters

4. all other differences.

Clone Unification

The proposed technique for the unification of clones in order to refactor them comprises three major steps as follows:

1. Control Structure Matching: The control structure of the code fragments is extracted into trees called Control Dependence Trees and they are matched for identifying potential refactoring candidates as well as to determine valid clone regions.

2. Program Dependence Graph Matching: The output of this phase is an optimal match of the PDGs corresponding to the matched subtrees from the previous step.

3. Checking Preconditions: A check is done against a set of predetermined conditions to ensure that the code behavior is preserved and to determine whether it is safe to refactor.

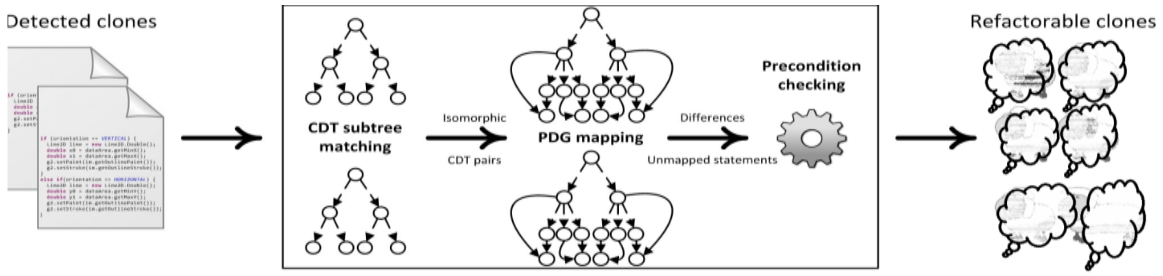

Figure : An overview of the proposed technique

Clone Refactoring

After the completion of the process, we need to define where the duplicated code can be safely extracted into a common method. According to Opdyke , each refactoring should be set with a set of preconditions, which monitor that the where the code is to be refectories. If any precondition is to be failed or not fully refectories the code the the program behavior is to be totally changed..

Conclusionand Future Work

Thi is a first step of research goal. To this end, we developed a clone refactoring technique through different algorithms that overcomes some of the limitations of previous approaches. The important and main feature of this thesis is to be defines the much more differences and detect them and also define through control dependency of code also map the difference and define where is to be mismatched and remove this miss matched . the one more main aspect of this thesis is to be define where is to clone and define if we remove the clone then the program behavior is to be changed or still same and define where to change is required . And currently defines the study of refactorability of clones detected from different clone detection tools such as Codepro , PMD.

In the evaluation of our approach, we compare the Codepro tool for the refactoring the Type-2 clones,.and our technique is to be more efficient then the codepro. And the another code clone is not related to java programs but also it can be revalorized directly.

As future work, we can detect some new and additional techniques for type 3 and type 4 clones.To accomplish this theme first we need to specify a particular base mark technique for type3 and type4 and then using art of state tools. And also define the decision of mismatching and compare the result with some new refactoring removing tool with some graph dependency notations.

References

[1] Mens, Tom, and Tom Tourwé. "A survey of software refactoring." *Software Engineering, IEEE Transactions on* 30.2 (2004): 126-139.

[2] M. Fowler: ―Refactoring. Improving the Design of Existing Code‖, Addison- Wesley, 1999

[3] Nikolaos Tsantalis, DavoodMazinanian,and Giri Panamoottil Krishnan, ―Assessing the Refactorability of Software Clones,‖ in Proc. IEEE Transactionson software engineering, vol.41, no.11, November 2015.

[4] C. K. Roy, J. R. Cordy, and R. Koschke, ―Comparison and evaluation of code clone detection techniques and tools: A qualitative approach,‖ Sci. ComputProgramm., vol. 74, no. 7, pp. 470–495, 2009.

[5] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, E. Merlo, Comparison and evaluation of clone detection tools, Transactions on Software Engineering 33 (9) (2007) 577_591.

[6] Arcelli Fontana, Francesca, et al. "Software clone detection and refactoring.*"ISRNSoftware Engineering* 2013.

[7] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner, ―Do code clones matter‖ in Proc. 31st Int. Conf. Softw. Eng., 2009,pp. 485–495.

[8] Roy C.K. and Cordy J.R., ―A Survey on Software Clone Detection Research‖,Queen's School of Computing, Technical Report No.2007-541, vol.115, September 2007.

[9] C. Roy, M. Zibran, and R. Koschke,
―The vision of software clone management:(Past, present, and future (keynote paper),‖ in Proc. IEEE Conf. Softw.Maintenance, Reeng. Reverse Eng., Softw. Evol.Week, 2014, pp. 18–33.

[10] Lozano and M. Wermelinger,―Assessing the effect of clones onchangeability,‖ in Proc. 24th IEEE Int. onf. Softw. Maintenance,2008, pp. 227–236.

[11] L. Jiang, G. Misherghi, Z. Su, S. Glondu, DECKARD: Scalable and accurate tree-based detection of code clones, in: Proceedings of the 29th InternationalConference on Software Engineering, ICSE 2007, 2007, pp. 96_105.

[12] Z. Li, S. Lu, S. Myagmar, Y. Zhou, CP-Miner: Finding copy-paste and related bugs in large-scale software code, IEEE Transactions on SoftwareEngineering 32 (3) (2006) 176_192.

[13] C.K. Roy, J.R. Cordy, NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization, in: Proceedings.