ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 5, Issue 12, December 2016

Recommending Move method refactoring using type constraints

S L Hemanth Chandra M.Tech, **G Ramesh** M.Tech, **Geetha Sreeram** M.Tech

Abstract— Refactoring is the process of restructuring the internal structure of the code without changing external output in order to improve quality of the program. There are several refactorings to correct code smells for improving quality of code in implementation of code. This paper presents an approach of an existing framework of type constraints to address various aspects of refactoring related to generalization. Type constraints are used to verify pre-conditions and to determine code modifications to improve quality in the source code. Prior to refactoring, detection of code smells in source code is suggested and based on the code smell its related refactoring should be applied to restructure the code without affecting the output .The proposed approach is that more than one method should be moved when system functionality is restructured. This work is implemented in the standard distribution of Eclipse(IDE).

Index Terms— Software refactoring, bad smells, renaming opportunities, move method, type constraints.

I. INTRODUCTION

In the process of software development developers do mistakes at every phase, specifically at coding phase there is an expose of generating errors which leads to faults, so maintenance cost increases. One of the best solutions for this problem is refactoring. The process of changing the internal structure of code in such a manner that, it does not alter the external behavior of the code yet basically improves its internal structure is termed as refactoring the code. The main prospect of refactoring is recovering the non-functional attributes of the software. Refactoring operation is identified by a name, a set of preconditions under which it is allowed and the actual source-code transformation (refactoring) that is performed. Several popular Integrated Development Environments (IDE's) such as Eclipse [8], IntelliJ IDEA, NetBeans supports software refactoring. One of the advantages of using these IDEs is for tool support. For example, Eclipse is one of the multi-language software development environments, which is written generally in java. Eclipse can be used to develop applications in Java and other programming language like C, C++, PHP, etc.

This paper presents an approach of an existing framework of type constraints [2] to address various aspects of refactoring related to code transformation. The type constraints imply number of common Java features, which were carefully designed to reflect the semantics of Java. The proposed approach is likely that more than one method should be moved when system functionality is restructured. The approach of identifying renaming opportunities by expanding conducted ones help to avoid incomplete restructuring. There are a dozen of well known refactoring types exists among all of them rename refactoring [1] is the best. The second prevalent refactoring is move which means moving software entities, such as software methods, fields and classes.

A model of type constraints for the purpose of checking type violations in a program that is of language type system is described [7]. If a program satisfies all type constraints then no type violations will occur at runtime. Type constraints are used to avoid ambiguity. A Type constraint is a relationship between two or more constraint variables that should hold in order for a program to be type correct. A constraint variable is one of the constant, for example: [E] the type of expression 'E', a constraint variable [E] represents the type of expression 'E', a type constraint contains the relationship between two or more constraint variables.

For example, a type constraint has one of the following forms:

(i) $\alpha 1 = \alpha 2$, indicating that $\alpha 1$ is defined to be the same as $\alpha 2$

(ii) $\alpha 1 \leq \alpha 2$, indicating that $\alpha 1$ must be equal to or be a subtype of $\alpha 2$,

(iii) $\alpha 1 < \alpha 2$, indicating that $\alpha 1 \leq \alpha 2$ but not $\alpha 2 \leq \alpha 1$,

II. RELATED WORK

The refactoring behavior consists of the following distinct activities. Identifying places where software should be refactor and verify which refactoring should be applied [3].The applied refactoring preserves behavior of code, apply the refactoring on that code. Assess the chance of the refactoring on quality characteristics of the software. Maintain the essence during the refactor code and distinct software artifacts (such as requirements documentation, design documents, tests, etc.).

A number of suggestions have been obligated to revive the problems by all of preconditions. Tip represented type constraints to efficiently prove preconditions that confide on interprocedural relationships surrounded by variable types. This is particularly pleasant for refactorings that are concerned with generalization. Roberts suggests augmenting refactoring with post conditions. Palsberg [2] designed an algorithm for "type safe approach named inlining that makes evaluate of type constraints to refine type violations".

Fowler [2] presents a comprehensive categorization of a rich number of smells in the code and exact refactorings, which includes steps on at which point to perform different types of refactoring manually yet issues familiar to generalization[2] related refactorings are not addressed.

Restructuring can besides be done at the phase of requirements specifications. Russso et al. convey "restructuring inherent language requirements specifications by decomposing them into a structure of viewpoints". Snelting and Tip used concept experiment to refactor Object oriented class hierarchies, based on the nature of this hierarchy by a set of software systems [6]. T. Mens implicit that each move method refactoring has to be proceeded by a rename refactoring [3]. Opdyke intended that the preconditions of refactoring as requirements on source level constructs [4], using a number of predicates that represents structural properties of code. Opdyke does not give the issue of using invariants to count a set of sufficient source code modifications. Snelting and Tip [6] indicated a concern for generating refactoring proposals for java applications. This field is based on already work by Tip in which type constraints record relationships between variables and members that intend be preserved.

III. PROPOSED WORK

In the implementation phase due to programmer mistakes smells occurs in code. Smell in computer programming refers to symptom in the source code that indicates a deeper problem which leads to a bug. Some common code smells are long method, lazy classes, large class, duplicated code, etc. Code smells are also called as bad smells. According to Kent Beck [3], bad smells are "structures in the code that suggest the possibility of refactoring". Detecting code smells in the program is simple by using refactoring tools such as Jdeodrant, RefactoringCrawler, Eclipse [8], etc. After detecting the code smells with the help of plug-in identify where to apply and which refactoring technique should be applied to minimize the code smells. This paper shows identifying duplicated methods and subsequently applied rename and move method refactoring to decrease the code smell, which results in improved quality in the source code.

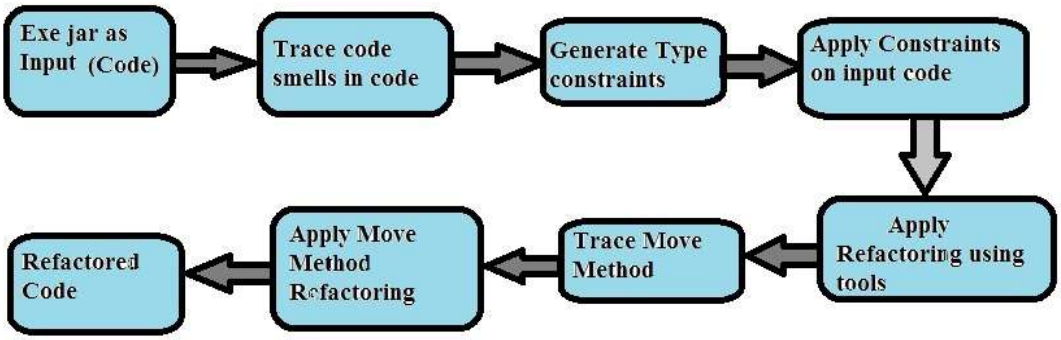

Figure 1: Overview of proposed approach

Move Method refactoring means moving method to another class when a class has too much behavior or when classes collaborate a lot and are too highly coupled.

To summarize, our methodology exhibits the following:

- It clearly indicates which methods and to which class they should be moved.
- It suggests refactoring which is applicable and behavior-preserving by examining a list of preconditions.
- It efficiently ranks multiple Move method refactoring suggestions based on their positive influence on the design of the system.
- It has been evaluated on large-scale open-source projects.
- It has been completely automated and implemented in an Eclipse [8], allowing the developer to apply the suggested refactoring on the source code.

After implementing the move method refactoring technique to the source code it results the following non-functional attributes:

- Maintainability: Easy changes to functionalities.
- Readability: Ease of reading code.
- Testability: Ease of testing and error reporting.
- Understandability: Understandability of code.

IV. Results

This section represents the proposed approach in practical view. The proposed approach is completely implemented in Eclipse IDE and code transformation using semi automated tools is successfully achieved. The following screenshot shows the renaming method in a java code

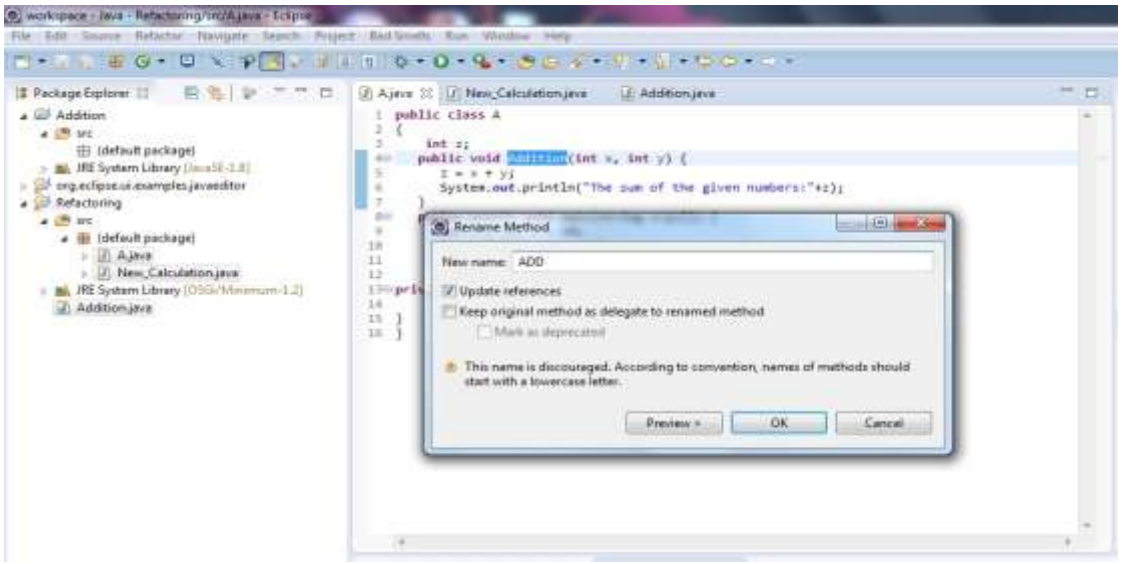

Figure 2: Screenshot of Rename method Refactoring

The proposed approach detects the similar classes in different java files and suggests to refactor the class names. Before applying move refactoring technique to the classes we can rename the class names as shown in the figure 2 and after renaming the class names, applying type constraints can be easy and then refactor the code by applying move method as shown in the figures 3 and figure 4.

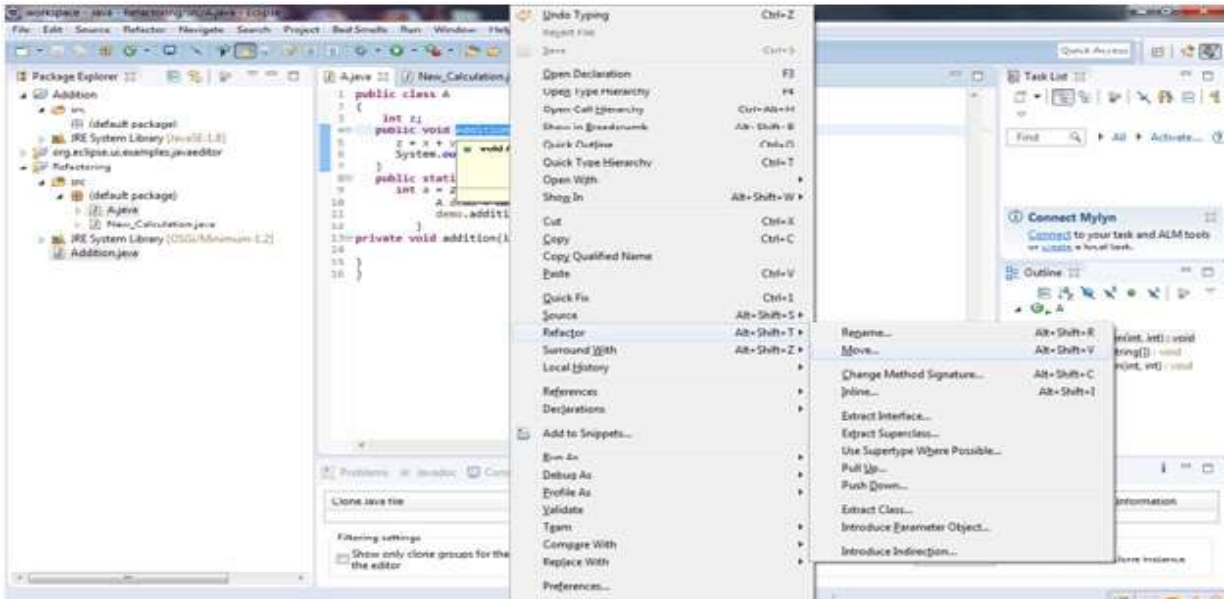

Figure 3: Screenshot of selecting Refactor- Move

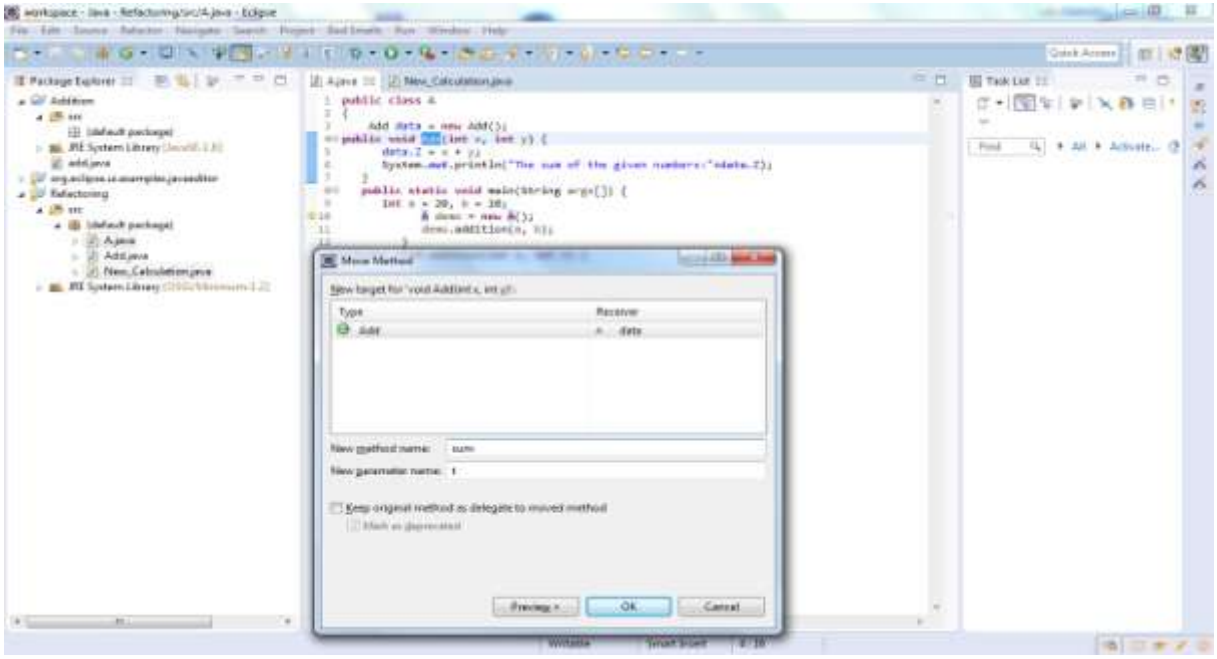

Figure 4: Screenshot of Move method

This technique analyzes method traces to detect refactoring candidates. This approach enables us to detect refactoring candidates based on the relationship between methods in program executions. It means that refactoring similar elements those are identified smells, in a program execution. Compared to normal refactorings move method refactoring removes future envy code smell with the help of Eclipse IDE [8], and by adding plug-in called JDeodrant which identifies bad smells in the code. Using this semiautomatic approach the productivity (in terms of coding and debugging time) can be improved, when compared to refactoring by hand. Similarly, one can expect developer productivity to improve after the software has been refactored because the software generally is more understandable, maintainable and evolvable.

V. Conclusion and future work

In this paper an approach of detecting code smells by using refactoring tools, implementing move method refactoring by using type constraints is recommended. Refactoring is an important part of software development cycle and refactoring tools are faster and used to reduce the bug occurrence from the code to avoid design problems and incomplete restructuring. An important category of refactoring is concerned with modifying types and class hierarchies which leads to quality in the source code. For this refactoring, type constraints are an excellent basis for checking preconditions and computing source code modifications.

Plans for future work include implementing various other refactorings such as replace method, inline method using type constraints in open source applications would be useful for developers and researchers.

REFERENCES

[1]. HuiLiu, QiurongLiu, Yang Liu and ZhoudingWang,"Identifying Renaming oppertunities by Expanding Conducted Rename Refactorings", IEEE Transactions on Software Engineering,Vol. 41,N0. 9, Sep. 2015.

[2]. F. Tip, A. Kiezun, and D. Baeumer, "Refactoring for generalization using type constraints", in proc. 18th Annu.Conf. Object-Oriented Program. Systems, Languages, Appl., Anaheim, CA, USA, Oct. 2003, pp. 13-26.

[3]. T. Mens and T. Tourwe, "A Survey of software refactoring", IEEE Transactions on Software Engineering, Vol 30, no.2,pp. 126-139,Feb,2004.

[4]. W. F. Opdyke, "Refactoring Object-oriented frameworks", PhD dissertation, Dept. of Computer Science, Univ. Illinois at Urbana-Champaign, IL, USA,1992.

[5]. M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, "Refactoring: Improving the Design of Existing Code". Pearson Education,1999.

[6]. G. Snelting and F.Tip "Reengineering Class Hierarchies Using Concept Analysis"1998.

[7]. Palsberg, Schartzbatch, "Object-Oriented Type Systems", John Wiley & Sons,1993.

[8]. Eclipse.Org. Online at www.eclipse.org.

S. L. Hemanth Chandra has obtained B. Tech degree in Computer Science from Jawaharlal Nehru Technological University, Ananthapuramu, India. Currently pursuing M. Tech in Software Engineering from Jawaharlal Nehru Technological University, Ananthapuramu.

Mr. G. Ramesh has obtained M. Tech degree in Software Engineering from Jawaharlal Nehru Technological University, Ananthapuramu, India. He is currently working as a lecturer in Department of CSE, Jawaharlal Nehru Technological University, Ananthapuramu. His interests include Software Engineering, Model driven software development and Agile Methodology.

Mrs. Geetha Sreeram has obtained M. Tech in Computer Science and engineering from Jawaharlal Nehru Technological University, Ananthapuramu, India. She is currently working as a lecturer in Department of CSE, Jawaharlal Nehru Technological University, Ananthapuramu. Her interests include Software Engineering and Bigdata.