Detection and Removal of Bad Smells instantly using a InsRefactor

Vignesh.P
PG Research Scholar
Department of Computer Science and Engineering
Sona College of Technology, Salem
jackvignesh@gmail.com

P. Ramya
Assistant Professor
Department of Computer Science and Engineering
Sona College of Technology, Salem
shriramyabe@gmail.com

ABSTRACT--Software refactoring is one of the essential techniques which are used to improve the software quality without affecting any of the external functionality of the software. There were numerous of software refactoring tools and code smells detection tools which are to be automatic or semi automatic. Most of these tools were human driven, so Software refactoring depends on developers' spontaneity. Due to little experience in software refactoring this may cause poor software quality and delays in refactoring. For this, we have proposed a framework called Instant Refactoring by Monitor framework using Ins Refactor which helps the inexperienced software engineers to do more refactoring quickly. Source code are instantaneously analyzed by the Monitor, if changes occur and it to be a code smells (symbols of possible problem in the code which requires refactoring) then framework invokes smells detection tools which helps the programmer to resolve the bad smells instantly without delay in time. The proposed framework has been evaluated, implemented and compared with the human-driven refactoring tools.

Keywords: Software Refactoring; Code Smells Detection; Monitor; clamant refactoring; Feature Envy.

I. INTRODUCTION

Software Refactoring [1] , [2] is to reframe the code in a series of small interior structure of objected oriented software that to improve the software quality based on the terms of maintainability, extensibility and reusability without changing the external behavior of the software. The term Refactoring was first proposed by Opdyke [2] after it became popular by the book written by Fowler et.al that published in the year 1999. Refactoring was tracked by the re-structuring [4] which was the extended history in the literature. Kim et .al assessed the value of software refactoring within Microsoft and suggested that refactoring is visible.Software refactoring tools are significant to support. For this, researchers have proposed tools to provide software refactoring. Most predictable Integrated Development Environment (IDE) such as Eclipse, Microsoft Visual Studio and IntelliJ IDEA provide tool support to conduct refactoring [5].

Developers have to identify the refactoring opportunities' if not they can't apply refactoring tools. Researchers have précised a number of typical situations which may need refactoring which Fowler calls bad smells. Experts proposed various Smells detection algorithms that to identify different kinds of code smells that may be automatic or semi-automatic [6][7].

Extant refactoring tools and smells detection tools are inactive and human driven. Murphy Hill et.al, [8] programmers fail to invoke refactoring tools and smells detection tools which may result in delay of refactoring and results in higher cost of refactoring. The reason for this is that unaware of extant tools, don't know where to invoke the tools and when to detect and resolve code smells.

This paper is to make the essential to take inexperienced software engineers to carry out more software refactoring quickly. Next we proposed an Monitor based clamant refactoring framework. Finally, we apply and evaluate the proposed framework and the result might help inexperienced software engineers in remove more code smells quickly.

This paper is ordered as follows, Section describe the related work, Section 3 present a framework of the clamant refactoring with Monitor, and its description Section 4 shows the evaluation of the framework, Section 5 shows the conclusion and future work.

II. RELATED WORK

A. Refactoring tools

To process software refactoring, various refactoring tools are available. The market is rapid and has a bunch of refactoring tools. For example: Refactoring Browser, JRefactory, IntelliJ IDEA, Eclipse and Visual studio. Extant refactoring tools can't be invoked until refactoring opportunities are identified by the software engineers with the help of code smells detection tools it may be automatic or semi automatic. The time and the occurrence for refactoring is depend upon the developer intent. The characteristics that affect the usability of a tool are automation, consistency, configurability, exposure and scalability of refactoring tools.

Researchers are attempting to improve the usability of refactoring tools Murphy-hill and Black [9] proposes the five values to improve the usability of refactoring tools. The extent of automation of a refactoring tool varies depending upon the refactoring activities. The reliability of a refactoring tool mostly depends upon the ability to guarantees that is provide refactoring transformation are truly behavior preserving contemporary software development tool only support primitive refactoring.

B. Smells detection

According to Beck, bad smells are "Structure in a code that suggests the possibility of refactoring". Bad smells are the signs of potential problems in the code that might require refactoring. Bunch of code smells detection tools for both automatic and semi automatic are available for various bad smells detection. Travassos et.al [10] proposed a technique called reading technique which makes the developer to identify the bad smells. Tourwe and Mens proposed an algorithm named smells detection algorithm with Logic rules in SOUL, Logic programming language that to identify the bad smells in the logic programs. Moha et al proposed a smells detection algorithm especially for Domain specific language (DSL) which is similar to that of Tourwe and Mens algorithm but slightly different in detection.

Munro proposed the Metric based approach which is a smells detection algorithm for java program. Van Rompaey et .al extent the Munro algorithm with the feature of detection of two kinds of smells general fixture and eager test. Tsantalis and Chatzigeorgiou proposed a genetic based algorithm which has to find out the bad smells Feature envy and it can refactored by move method.

III. FRAME WORK

This section shows the clamant refactoring with Monitor framework which takes out the developer to detect and resolve bad smells. Figure: 1, Show the overview framework of our proposed system which may contains the monitor, smells detection, smells view, refactoring tools, and feedback controller. Each may have their own role that to make a clamant refactoring if there may be any changes occur in the source code which leads to the need of refactoring technique. This frame work may help the programmer to analyze code smells instantly whenever source code changes occur and results in potential code smells. The Framework shows the detection and removal of the various code smells. When a programmer make a changes in the source code then the changes are get analyzed by the monitor. Monitor analyzes the changes and those changes are get forwarded to Smell Detection. Smell detection encloses the Code Smell Detector and Refactoring Manager. The various code smell detection algorithm are integrated and based upon the code smell the refactoring methods are suggested to the developer. So that the developer can easily identify the code smells and refactoring methods to resolve those smells. Developer can able to view the suggestion of code smell through the smell view. Smell view is the small message box that to show the details of code smell. Developer can close the smell view and can continue the programming

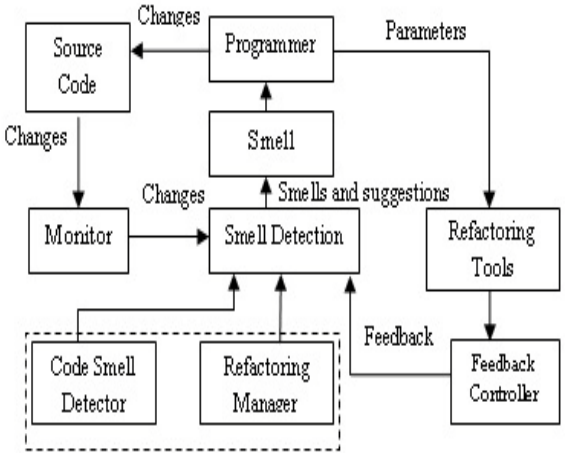

Figure 1.Overview of the Framework

The framework is composed up of an Monitor, collection of smells detectors and a smells view, a feedback controller. These will be explained individually explained by the following paragraphs.

C. Monitor

Monitor oversees the changes made on the source code .This may run in the background of the source code, if it analyzes some changes in source code then it calls for smells detectors. This has to perform instantly and take out the knowledge of smells to the developer. Monitor helps the developer to view the code smells with the help of Smells view. Monitor is meant to give a warning so that a mistake can be avoided by the developer.

D. Smells detectors & Refactoring tools

This may contains a collection of code smells detectors for various code smells like Data class, Large Class, Long Method, Switch Statements, Public Field, and Common Methods in Sibling Classes, Duplicated Code and Feature Envy. Refactoring tools are to be extant one but a detection algorithm is to be different from extant which may carry out by this InsRefactor tool. We improve the performance of this tool by improving the tool tendency to detect more code smells.

E. Feedback controller

Various control algorithms are defined for smells detection algorithm to improve readability of the framework, feedback control algorithms are get implemented in the prototype. These control algorithms are similar and simple template. This control algorithm improves the performance of the framework.

F. Smells View

If the changes occur in a source code it forwards to a code smells detectors which detect the bad smells and these smells are viewed with the help of smells view to the developer. This helps the developer to easily identify the location and invoke the refactoring technique. The developer can quit the smells view and continue coding. The smell view helps the programmer in a friendly way of displaying the code smell and the extracting method for the particular identified code smell.

IV. EVALUATION

Clamant refactoring is to take out the inexperienced software engineers to make them to do more refactoring quickly**.** In earlier refactoring the developer used to detect the smells by manual driven but it takes more time for lesser refactoring. This clamant refactoring may facilitate more refactoring with leisure time with large number of resolved code smells.

In the earlier InsRefactor [11] prototype implementation the eight kinds of smells were detected. The prototype has implemented for the detection of Data class, large class, Long Method, Switch Statement, Public Field, Sibling Class, Duplicated Code, and Long Parameter List that is based on JCCD [12]. This prototype is based on the Eclipse and Java. Modified source code is get compared with the all other source code which to make identification for similarities. Identified Smells are notified in the smells view where developer can easily notify the bad smells.

InsRefactor is the tool which has to identify the wrong method location called feature envy smells. This tool has been plugged in the eclipse through that the developer can detect the bad smells. Feature Envy refers to smells in which when the methods makes too many calls to other classes to obtain data or functionality. The detection strategy for Feature Envy can be given by,

$$\text{Feature Envy} = \max_{c \neq cm}(|F_c|) - |F_{cm}| \quad (1)$$

where,

m-the method, for which we want to calculate the feature envy,

F_{c}-- the set of features used by *m* that belong to type *c*

c_{m}-- the class in which *m* is defined

Using the above Detection Strategy the Feature envy code smells can be detected. This is Conventional based detection strategy which helps to detect the feature envy smells promptly. Not only the feature envy smells detector, the various code smells detector are get integrated in our InsRefactor tool that detects the various code smells and helps the programmer to resolve the smells.

This InsRefactor helps the programmer to identify the code smells instantly and helps in doing the refactoring without delay in process. This InsRefactor get processed with the one team of the developers and another team get processed without the InsRefactor. From the observation among the two developers it shows the team who carries out the InsRefactor makes the Refactoring promptly without any delay in their schedule whereas the team without the InsRefactor feels difficult to do refactoring and delays in their process, so they cause delay in schedule and makes the software cost to be more.

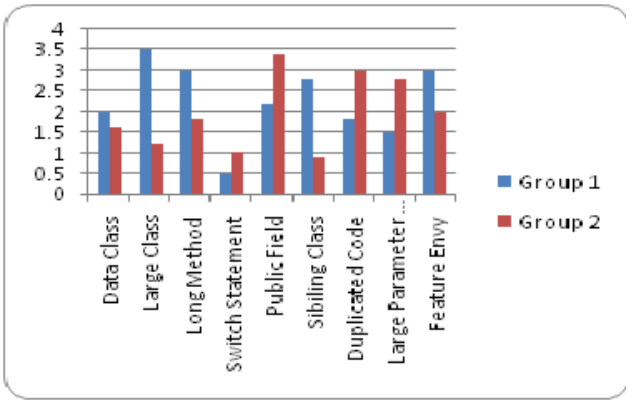

Figure: 2 Evaluations of Code Smells

The Figure:2 shows the graph which represents the variation of identification of nine kinds of bad smells like Data Class ,Large Class, Long Method, Switch Statements, Public Field, Sibling Class, Duplicated Code, Large Parameter List, Feature envy among the two group peoples. The Group 1 is the team of programmers who carries the InsRefactor and the Group 2 peoples works without InsRefactor. From the above chart we can able to identify that the variation between two groups. The different kinds of code smells charted above are most common code smells which occurred mostly during the programming. InsRefactor tool helps the programmer to identify the code smells promptly. The Lifespan and range of the various kinds of code smell differs at each level of programming.

V. CONCLUSION & FUTURE WORK

This paper is to make the inexperienced developer to do more refactoring quickly. We propose a Clamant refactoring with Monitor framework which makes the developer to identify the changes of the source code that results in the bad smells and to resolve bad smells.

This paper explains the various bad smells detection algorithms and refactoring tool of InsRefactor. This framework has to detect the nine kinds of bad smells. To improve the performance of the framework, improve the functionality of the framework by enhancing the more refactoring tools which to identify more bad smells with various bad smells detectors and to resolve it use various refactoring methods.

VI. REFERENCES

[1] T.Mens and T.Touwe, "A Survey of Software Refactoring,IEEETransactions.SoftwareEng" vol.30, no.2, pp.126-139, Feb.2004.
[2] W.F.Opdyke,"Refactoring Object-Oriented Frameworks," PhD dissertation, Univ. of Illinois at Urbana-Champaign, 1992.
[3] W.F.Opdyke,"Refactoring Object-Oriented Frameworks," PhD dissertation, Univ. of Illinois at Urbana-Champaign, 1992.
[4] R.Arnold,"An Introduction to Software Restructuring," Tutorial on Software Restructuring, R.Arnold,ed. IEEE CS Press,1986
[5] E.Mealy and P.Strooper, "Evaluating Software Refactoring Tool Support," Proc. Australian Software Eng. Conf., pp.331-340, 2006.
[6] T.Kamiya, S.Kusumoto, and K.Inoue, "CCFinder: A Multi Linguistic Token Based Clone Detection System for Large Scale Source Code," IEEE Trans. Software Eng., vol.28, no.6 ,pp.654-670, July 2002
[7] N.Tsantalis and A.Chatzigeorgiou,"Identification of Move Method Refactoring Opportunities," IEEE Trans. Software Eng., vol.35, no.3, pp.347-367, May/June 2009.
[8] E.Murphy-Hill, C.Parnin, and A.P. Black, "How WE Refactor, and How WE Know IT," IEEE Trans. Software Eng., vol. 38, no.1, pp.5-18, Jan/Feb. 2012.
[9] E.Murphy-Hill and A.P Black, "Refactoring Tools:Fitness for Purpose, "IEEE Software, vol. 25, no.5, pp.38-44, Sept./Oct. 2008.
[10] G.Travassos, F.Shull, M.Fredericks, and V.R Basili, "Detecting Defects in Object-Oriented Designs:Using Reading Techniques to Increase Software Quality,"Proc. 14th ACM SIGPLAN Conf. Object-Oriented Programming System,Languages, and Applications, pp. 47-56, http://doi.acm.org/10.1145/320384.320389, 1999.
[11] Hui Liu, Xue Guo, and Weizhong Shao, "Monitor-Based Instant Software Refactoring",IEEE Transactions on Software Engineering, vol. 39, no. 8, August 2013.
[12] B.Biegel and S.Diehl," JCCD:A Flexible and Extensible API for Implementing Custom Code Clone Detectors,"Proc. 25th IEEE/ACM Int'l Conf. Automated Software Eng.,pp.167-168,Sept. 2010.