Journal of Applied Physics and Engineering Vol.1, No.1 (2016) 30-33

Monitor Based Instant Refactoring Framework for Detecting and Resolving Code Smells

Konanki Dinesh [1]

[1]*P.G.Scholar, Computer Science & Engineering, JNTUACE, A.P., India*
[1]konankidinesh@gmail.com

***Abstract*—Software refactoring is a process of enhancing quality of a software system in such a manner it should not affect the external behavior of the code, it improves its internal structure. Refactoring is used to improve code quality, reliability, and maintainability throughout the software life cycle. The automated tools can be used to detect various kinds of code smells. This may cause another problem of extended time and effort because the smell being refactored may have impact on resolving or increasing some other types of code smells. That is a smell being refactored may have impact on presence of an existing smell or brought some more problems into the system. The previous methods lead to lots of human effort and huge extent of maintenance time. Hence to minimize the manual workload to get the quality source code for easy maintenance the clamant refactoring technique is proposed in this work to enrich detection and sequencing of bad smells.**

I. INTRODUCTION

Software refactoring [1], [2] is to reform the code in a series of small internal structure of objects oriented software that to improve the software quality in term of maintainability, reusability and extensibility of such software, while software external output remains unchanged. The term Refactoring was first proposed to Opdyke [2] after it became popular with the book written by Fowler et.al that published in the year 1999. Refactoring was tracked down 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 what refactoring is visible. The critical thing in software refactoring is tool support. For this, researchers have proposed tools to provide software refactoring. The most predictable IDE's such as Microsoft Visual Studio, Eclipse 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 a 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.

We proposed a Monitor based clamant refactoring framework. Finally, we apply and evaluate the proposed framework and the result might help inexperienced software engineers in removing more code smells quickly.

II. RELATEDWORK

The various software refactoring tools are available in the market for example Eclipse and Microsoft visual studio, IntelliJ IDEA. Extant refactoring tools cannot be invoked until refactoring opportunities are identified by the software engineers with the help of code smells detection tools it may be the automatic or semi-automatic. Researchers are seeking to enhance the usability of software refactoring tools. Murphy-hill and Black [9] introduce the five values to improve the usability of refactoring tools. The extent of automation refactoring tools varies depending upon the refactoring activities. The reliability of a refactoring tool mostly depends upon the ability to guarantee that is provided for refactoring transformation is truly behavior preserving contemporary software development tool only supports primitive refactoring.

According to Beck, bad smells are "Structure of a code that suggests the possibility of refactoring". Bad smells are the signs of potential problems with the code that might require refactoring. A bunch of code smells detection tools for both automatic and semiautomatic 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

Selected paper: International Conference On Computing (NECICC-2k15)

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 programs. Van Rompaey ET .al extend the Munro algorithm with the feature of detection of two kinds of smells general fixtures and eager test. Tsantalis and Chatzigeorgiou proposed a genetic based algorithm which has to find out the bad smells Feature envy and it can refactor by move method.

III. FRAMEWORK

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 contains the monitor, smells detection, smells view, refactoring tools, and feedback controller. Each 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. With this framework the programmer can analyze the smells instantly whenever changes occurred in source code and results in potential code smells. The proposed framework shows the detection and removal of the various code smells. When a programmer makes a change in the source code, then the changes are getting analyzed by the monitor.

A. Monitor & Smell Detection

Monitor to analyze the changes and those changes are getting forwarded to Smell Detection. Smell detection encloses the Code Smell Detector and Refactoring Manager. The various code smell detection algorithm is 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 invoke appropriate refactoring methods to resolve those smells. The developer can be able to view the suggestion of code smell in the smell view. Smell view is the small message box which shows the explanation and suggestions. The framework is composed up of a monitor, collection of smells detectors and a smells view, a feedback controller. The explanation for monitor, smell detectors and refactoring tools and feedback controller are provided in the following paragraphs.

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

B. Smell Detectors & Refactoring Tools

This may contain a collection of code smells detectors for detecting various code smells like a Large Parameter List, Lazy Classes, Large Class, Long Method, Switch Statements, 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 INS Refactor tool. We improve the performance of this tool for improving the tool tendency to detect more code smells. B. Biegel and S. Diehl [11] proposed JCCD is written in Java to detect clones in Java source code.

C. Feedback Controller

This may contain a collection of code smells detectors for detecting various code smells like a Large Parameter List, Lazy Classes, Large Class, Long Method, Switch Statements, 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 INS Refactor tool. We improve the performance of this tool for improving the tool tendency to detect more code smells. B. Biegel and S. Diehl [11] proposed JCCD is written in Java to detect clones in Java source code.

D. Smell 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 on 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 of the particular identified code smell.

Selected paper: International Conference On Computing (NECICC-2k15)

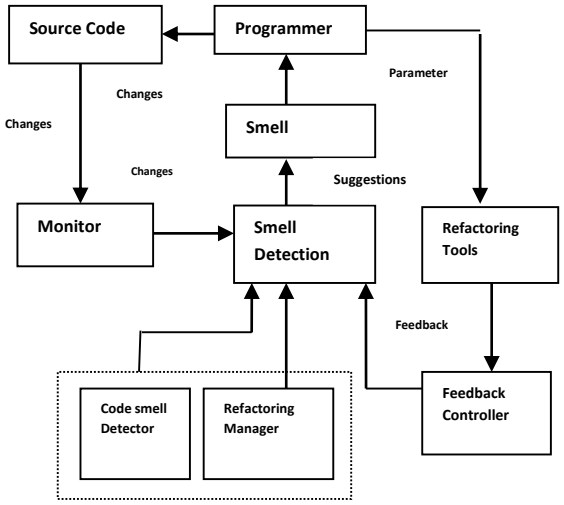

Figure1. Overview of the Framework

IV. DIFFERENT SMELLS IN CODE

A smell in source code is evidence that indicates something incorrect somewhere in the source code. If a bad smell occurs it denotes that the code should be rechecked. Identifying those places of bad design is a challenging job for inexperienced developers. These areas of bad scheme are known as Bad Smells. To operate mechanism of refactoring it is significant to decide when refactoring should start and when refactoring should stop. Unless if we don't understand when refactoring needs should be considered refactoring does not carry full benefits. If we do not understand when refactoring needs to be applied. Making it easier about a software developer in deciding whether or not particular software needs to be refactored, Fowler & Beck gave a series of bad code smells.

A. Descriptions of smells

*Duplicated Code:*Finding the same code structure more than one place. For example, this problem may have the two sibling subclasses.

Solution: The duplication is eliminated by using Extract Method and Form Template Method in both similar classes.

*Long Method:*Long Method is more difficult to understand, and then performance concerns with respect to lots of shot methods are largely obsolete. The Long Method smell is related to the Brain Method smell explain by Lanza et al. [LM06], which centralize the functionality of a class, in the same manner God Class centralizes the functionality of the whole subsystem, or sometimes even a complete system.

Solution: Decompose conditional, Extract Method, Replace Temp with a query.

*Large Class:*Class that has too many instance variables or methods, duplicated code.

Solution: Extract Class, Extract Interface, And Introduce Foreign Method.

Long Parameter List: Long parameter list is difficult to understand, because they become incompatible and difficult to use, and because you are forever changing them as we need more data.

Solution: Introduce Parameter Object, Replace parameter with Method.

*Divergent Change:*Occurs when one class are frequently changed from into different ways for different reasons.

Solutions: Extract Class.

*Feature Envy:*Feature Envy is a Code Smell, occurs to methods. A method has Feature Envy on another class, if it uses more features (i.e. Fields and methods) of another class than on its own.

*Data Clumps:*Data Clumps smell means that two or more data items possessed in number of places.

Solutions: Preserve Whole Object or Introduce Parameter Object and Extract Class.

V. EVALUATION

This clamant refactoring may facilitate more refactoring with leisure time for large number of resolved code smells. Clamant refactoring is to take out the inexperienced software engineers to make them to do more refactoring quickly. In earlier stages to detect the smells of manually driven, but it takes more time for detecting smells and lesser refactoring. The Ins Refactor tool 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 when the methods make too many calls to other classes to obtain data or functionality.

In the earlier Ins Refactor [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 theEclipse 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.

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

Where, F_{c--} the set of features used by *m* that belong to type *c*, Cm-- the class in which *m* is defined,

The Feature Envy code smells can be detected by using the above Detection Strategy. Then Feature envy smell detectors, the various code smells detector is getting integrated in our Ins Refactor tool that detects the various code smells and help the programmer to resolve the smells. This Ins Refactor helps the programmer to identify the code smells instantly and helps in doing the refactoring without delay in processing.

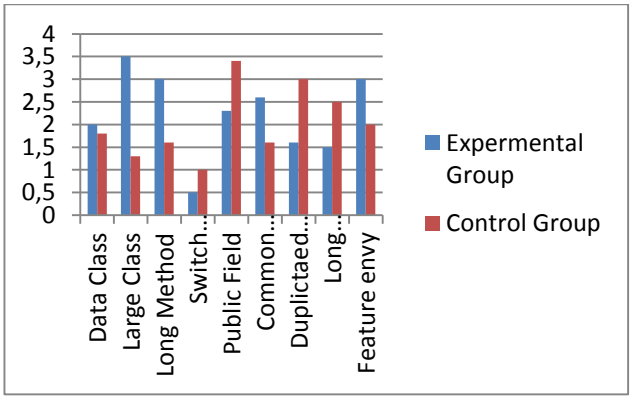

Figure 2. Evaluations of Code Smells

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

VI. CONCLUSION & FUTURE WORK

In this work, instant refactoring framework of clamant algorithm is proposed which makes the developer to identify the changes of that source code analyze results of the bad smells and to resolve smells it makes to conducting refactoring quicker for the inexperienced developers. This framework has to detect nine kinds of code smells and also improve the performance of framework to reduce software cost and improve quality. The feature work carried over on other kinds of Eclipse, Visual studio and intelliJ IDEs etc. Also the semi-automated change modification will also be fully automated under the system that facilitates the developer for clear refactoring process.

The future work is to evaluate the proposed framework, further based on more applications. The prototype implementation, INS refactor focuses on functionality rather than performance, and then future work is needed to improve the performance. The proposed framework could be improve code quality and reduce software cost.

REFERENCES

[1] T. Mens and T.Touwe, "A Survey of Software Refactoring," IEEE Transactions Software Eng., 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] R. Arnold, "An Introduction to Software Restructuring," Tutorial on Software Restructuring, R. Arnold ,ed. IEEE CS Press,1986.
[4] E. Mealy and P. Strooper, "Evaluating Software Refactoring Tool Support," Proc. Australian Software Eng. Conf., pp. 331-340, 2006.
[5] 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.
[6] 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.
[7] 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.
[8] E. Murphy-Hill and A.P Black, "Refactoring Tools: Fitness for Purpose," IEEE Software, vol. 25, no. 5, pp. 38-44, Sept/Oct. 2008.
[9] 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, Systems, Languages, and Applications, pp. 47-56, 1999.
[10] Hui Liu, XueGuo, and Weizhong Shao, "Monitor-Based Instant Software Refactoring," IEEE Transactions on Software Engineering, vol. 39, no. 8, August 2013.
[11] 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.