Identification and Refactoring of Bad Smells to Improve Code Quality

Sandeep Kaur[1], Harpreet Kaur[2]

[1]Research Scholar, Department of Computer Engineering, UCOE Punjabi University, Patiala, 2015, India

[2]Assistant Professor, Department of Computer Engineering, UCOE Punjabi University, Patiala, 2015, India

Abstract: ***Bad Smells are design flaws in the code that make an architectural design weaken and bad. Bad smells does not prevent our source code to show any input. There is no any effect of the bad smells on output of the source code. But due to the bad smells our source code becomes hard to modify and understand. Presentation of bad smells in the code prioritized for refactoring. Therefore detection and refactoring of these bad smells is must. But with the assistance of the refactoring we can eliminate these design flaws and convert a suboptimal into optimal code. Refactoring is used to expose the bugs from the source code. Refactoring is a technique that makes the source code of the software easier, more readable, more efficient, more extendable and more understandable by eliminating the bad smells from the source code. Even the traditional software development methods that also start with a better design, but when we want to apply some changes to that software it may lead to a suboptimal design. Software whose requirements is changed or is under specification is a sub optimal design. Although a source code also has a bad smells in it that make the source code irrelevant and difficult for the programmers.***

Keyword: Refactoring, Bad smells, Detection, Window based GUI.

1. Introduction

Software systems need to change by time to time. There may be several reasons to change it. Some of them are changing requirements of the user, advanced or change in technology, cost benefits changes. Source code of the software is timely changed by the developer to make maintenance easy. But sometime a little change in the software source code degraded the quality of the software and loses its good design. To make the changes possible to the source code without changing the functionality of the software there is one technique, which is known as refactoring technique. But before applying refactoring to the source code we to find out where the code is to be refactor.

2. Bad Smells in the Code

The term bad smell was coined by Fowler and Beck. If there is any bad smell in the code it means there is some deeper problem in the code. Bad smells are structural problems that make a source code difficult to understand and maintain. Bad smells in source code is neither a bug nor an error. Bad smell does not mean that any technical problem. These bad smells do not prevent the source code from execution. However bad smells indicates the weakness in the source code which can create problems in future and due to the presents of these bad smells the working of the software getting slower down or risk of error is increasing. So if there is any bad smells in the source code it should have to be refactored.

Large Class	In large class there are too many functionalities are gathered in one class.Some developer make a large class for their convenience but it may lead to confusion when the code is analyzed or read by any another programmer. It's really hard to understand the functionality of large class.
Feature envy	It is a smell in which a class is interested to use the data or functions of another class in the source code.Feature envy means violation of principle of class.
Duplicate code	Duplicate code is the code in which the same copy of the code or expression is placed many places in the same source code.If we applying manual refactoring on the source code then he\she have to refactor the same duplicate code at all places which is a difficult task.
Switch Statement	Switch Statement does not necessarily mean bad smell.But it may lead to duplication in the source code.Often we see a same switch statement is scatteredat various places in the same source code.It is better to use the concept of polymorphism rather than switch statement

2.1 Refactoring

Refactoring is a process which can be applied to the source code of the software to removes bad smells from it. Refactoring only changes the internal structure of the code but there is no any effect of the refactoring on the external behavior of the code. External interface of the software remains the same. Refactoring makes the source code more reliable and efficient by removing the bad smells from it.

2.2 Refactoring loop

Refactoring becomes a very important technique these days. But before applying the applying the refactoring we have to understand the various facts about the refactoring such that:-

1) Analyze the source code.
2) Find out where the code should have to be refactored.
3) When the code should be refactor.
4) Why the code should be refactor.
5) Type of bad smells.
6) Which refactoring technique is more suitable to refactor the particular type of bad smells?
7) Effect of refactoring on the code.

2.3 Refactoring techniques

Move Method	Move method means moving a method to another class when classes have too many functionalities to do.
Extract Method	Extract method means extract a piece of code that is appear at many places in the source code and make a new method or class.
Replace Temp with query	Replace temporary variables with the method calls.It is same as extract method.
Rename method	Rename the method according to the functionality of the method.
Replace array with object	An array has certain elements with different things.Replace that array with an object in which there is a field for each element.
Inline Class	When a class does have much work to do then it is better to move its entire feature to another class and remove it.
Push Down	Some functionalities of the super class are valid for some subclasses. Move this functionality to those subclasses.

3. Literature Survey

It presents about the previous studies of evaluating what the other researchers have done regarding code smells detection.

Author	Description
Karnam Sreenu	Software refactoring is a technique that transforms the various types of software artifacts to improve the software internal structure without affecting the external behavior. Various types of object oriented metrics can be calculated to detect the bad smells.
Almas Hamid, Muhammad Ilyas, Muhammad Hummayun and Asad Nawaz	Refactoring is a technique to make a computer program more readable and maintainable. A bad smell is an indication of some setback in the code, which requires refactoring to deal with. Many tools are available for detection and removal of these code smells.In this work, we studied different code smell detection tools minutely and try to comprehend our analysis by forming a comparative of their features and working scenario.
Francesca Arcelli,Fontana Pietro Braione,Marco Zanonia	Code smells are structural characteristics of software that may indicate a code or design problem that makes software hard to evolve and maintain, and may trigger refactoring of code. Recent research is active in defining automatic detection tools to help humans in finding smells when code size becomes unmanageable for manual review
Martin Fowler	Bad smells are signs of potential problems in code. Detecting bad smells, however, remains time consuming for software engineers. Large Class is a kind of bad smells caused by large scale, and the detection is hard to achieve automatically.
Marija Katic	The main definitions and terms concerning software redesign is closely connected with the testing. This paper briefly presents the software redesign process and methods that are used in that process. Although one can say that for example a source code redesign belongs to the implementation phase, tests are needed to ensure that the behavior is not changed.
Wei Liu	This paper discusses how to detect and eliminate the lazy class. An automatic syntax tree (AST) is proposed in this work. Firstly source code file is converted into ASTs and then three types of relationship are considered between the classes and extracted syntax tree. After carrying out several operations on these ASTs lazy class is obtained and removes it automatically. This approach has good efficiency, and its execution time has a linear relationship to the size of a system.
Raju M. Tugnayat	This paper discusses refactoring which is one of the techniques to keep software maintainable. However, refactoring itself will not bring the full benefits, if we do not understand when refactoring needs to be applied. To make it easier for a software developer to decide whether certain software needs refactoring or not, Fowler & Beck's idea was that bad code smells are a more concrete indication for the refactoring need than some vague idea of programming aesthetics.
Mohamed Eladawyl	In this paper a novel assessment criterion based on including the inherited attributes and methods has been proposed. Additionally, the effect of including the inherited attributes and methods in measuring class cohesion has been extensively discussed.

4. Proposed Work

A Window based GUI application has been developed to detect bad smells. It detects the bad smells according to the Object Oriented Metrics. Large Class, Switch Statements, Long Parameter List, Dead code, Conditional Statement, Duplicate Code, Comments are the bad smells that are detected by the GUI. This application detects the bad smells from the source code of java. Also refactor the detected bad smells by using appropriate refactoring techniques. It focuses on improving the quality or performance by decreasing the complexity of the source code.

4.1 Experimentation

The experimentation is done as follow:

The GUI interface is creates in VB.Net language and support detection and refactoring of bad smells. This application is created in Visual Studio 2010. It detects the bad smells according to the object oriented metrics and refactors them from the source code. To detect these bad smells different types of object oriented metrics are measured. For different kind of bad smell the object oriented metrics is also different.

4.2 Metrics tocalculate the bad smells:

1) **Lines of source code** Lines of code usually refer to non-commentary lines meaning pure white spaces and lines containing only comments are not included in the metric.
2) **Lines of comment** Lines of comment are used to describe the meaning of the statement in the code.

4.3 Detection of bad smells by using Window Based GUI Application

1. Conditional statements
Firstly we calculate the metrics to detect the conditional statements.

The following metrics are used for the detection of conditional statements.

Rule:
If the number of if-else conditions or else-if condition is less than or equal to 10 then it is a low risk program and not considered as bad smell.

If the number of if-else conditions or else-if condition is less than or equal to 20 then it is a moderate risk program.

If the number of if-else conditions or else-if condition is more than or equal to 50 then it is a high risk program and considered as bad smell.

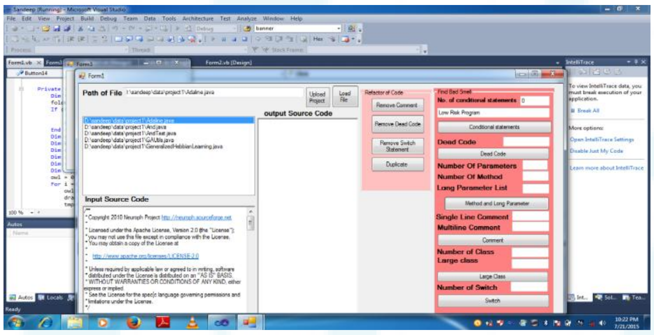

Figure 1: Detection of Conditional Statements

2. Dead Code To detect the Dead Code Bad smell. **Rule** If code is never processed at run time.

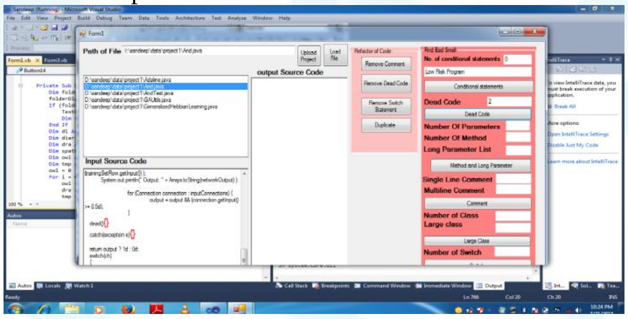

Figure 2: Detection of Dead Code

Long Parameter List
To identify the long parameter list we have calculate number of parameters in the code and number of methods in the source code.

Rule If the number of parameters more than 15 then we consider it as a bad smell. If the value of number of parameter list is less than 15 then it is a low risk program.

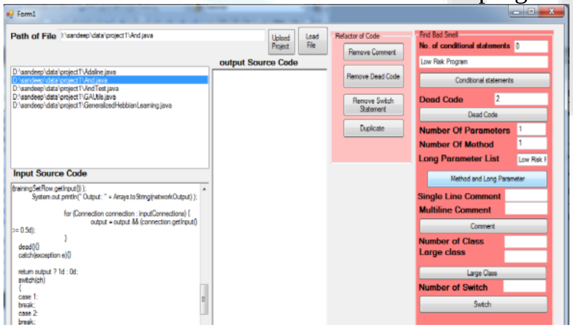

Figure 3: Detection of Long Parameter List

Comments Rule: If number of comment lines that are present in the code are more than the average lines of code.

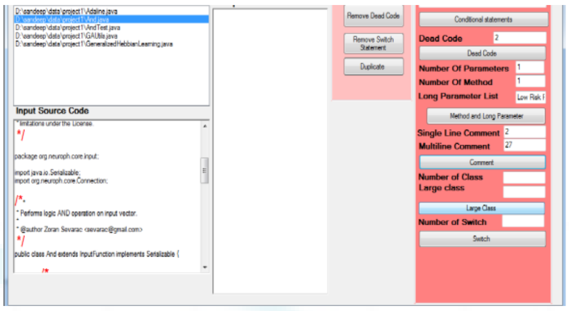

Figure 4: Detection of comments

Large Class To detect the large class from the source code we calculate line of code metrics and number of methods.**Rule** If the LOC (line of codes)are greater than 150 then this class is considered as large class.If number of method is greater than or equal to 10.

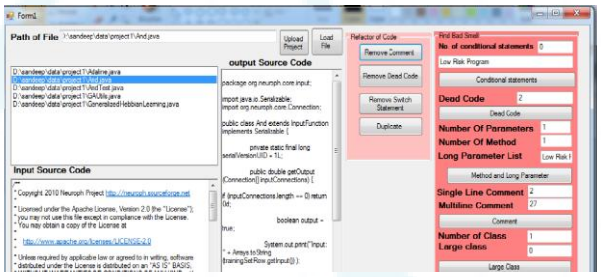

Figure 5: Detection of Large Class

Switch Statements: For the detection of Switch statement the rule is : **Rule** If the number if switch statement is greater than 5 then we consider it as a bad smell.

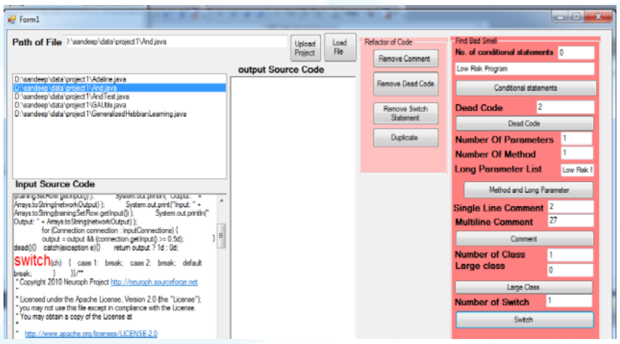

Figure 6:.Detection of Switch Statements

4.4 Applying the Refactoring Techniques

The results are shown below**:**
1. Refactoring of Comments:

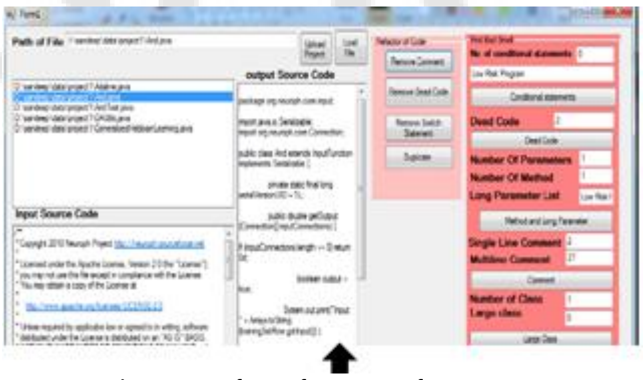

Figure7: After refactoring of comments

2. Refactoring of Dead Code

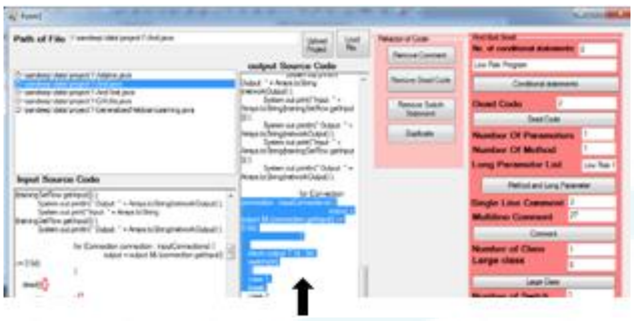

Figure 8: After Refactoring of Dead Code

3. Refactoring of Switch Statements

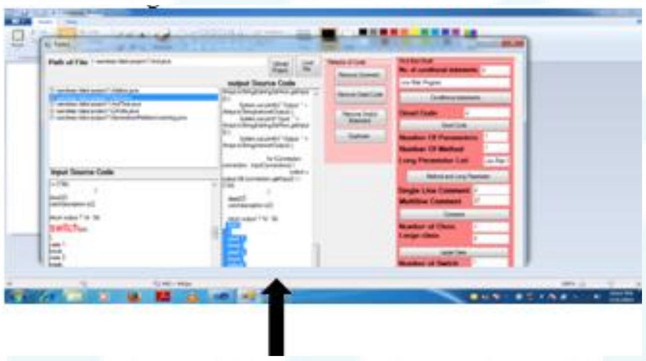

Figure 9: After refactoringof switch statements

5. Conclusion

Six types of bad smells are detected through this GUI. From these six bad smells our project contains three types of bad smells which are refactored by using this GUI. This work conclude that removing of bad code smells by using the refactoring makes the software more reliable, more efficient and more readable also decrease the complexity of the source code then its original source and the external working of the software remains same. In this work the main focus is to develop the window based GUI application to detect and refactor the bad smells from the source code. In the detection of bad smell we identify the code which degrades the quality of the source code. Although these bad smells does not produce any error at the runtime but the presence of bad smells in the code makes the source code difficult for maintenance.So the use of refactoring to change the internal structure of the code in such way that no any change occurs to the external interface of the software. The calculations of values of the bad smells are on the basis of object oriented metrics.

6. Future Scope

1) In the future the comparison will be performed between developed window base GUI and Eclipse.
2) Calculate more number of metrics to detect more bad smells.
3) Refactoring methods will be applied on the basis of the bad smells to refactor them.

References

[1] KarnamSreenu,D.B.JagannadhaRao2 ”Performance Detection of bad Smells In Code for Refactoring Methods”International Journal of Modern Engineering Research (IJMER).Vol.2,Issue,5,sep-oct,2012 pp-3727-3729.
[2] Almas Hamid, Muhammad Ilyas, Muhammad Hummayun and Asad Nawaz “ A Comparative Study on Code Smell Detection Tools” International Journal of Advanced Science and Technology Vol.60, (2013)
[3] Francesca Arceli Fontana Pietro Braionea Macro Zaninia”Automatic detection of bad smellsin code”Journal of object Technology Published byAitocjot2011.
[4] Martin Fowler “A list of Refactoring tools”Http://www.refactoring.com
[5] Stefan Slinger “Code Smell detection in Eclipse” Delft University of Technology.
[6] Marjia Katic “Software Redesign Methods” Departments of Applied Computing Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia.
[7] Ganesh B. Regular and Raju M. Tugnayat “Bad smelling concept in Software refactoring”, Jawaharlal Darda institute of Engineering and Technology, MIDC, Lohara, Yavaymal,India.
[8] Safwat M.Ibrahim,Sameh A.Salem,ManalAand Mohamed Eladawyl” Identification of Nominated Classes for Software Refactoring Using Oject-Oriented Cohesion Metrics” IJCSI International Journal of Computer Science Issues,Vol.9,Issue 2, No.2, March 2012 ISSN(Online):1694-0814