DETECTION OF BAD SMELLS IN SOURCE CODE ACCORDING TO THEIR OBJECT ORIENTED METRICS

Anshu Rani[1] (Research Scholar), Harpreet Kaur[2] (Assi Prof)
Department of Computer Engineering, UCOE
Punjabi University, Patiala

Abstract: Refactoring is a technique for improving software structure without changing its behavior which can be used toremove bad smells and increases software maintainability 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. There are many automatic detection tools to help humans in finding smells but these tools are platform dependent. For e.g. in eclipse we can execute the only java code. Automation detection tools are limited for detecting some bad smells. So to detect the more number of bad smells, we have to work on many detection tools so the window base GUI application is developed in the visual studio tool which detects more bad smells according to their Object Oriented Metrics. This paper reviews the window base GUI application developed in visual studio tool for code smell detection. This paper describes detection of bad smells and used software metrics to identify the characteristics of bad smells "lazy class" , "long method" ," comment lines" and "large class"

Keywords: Bad Smell, bad smell detection window base GUI application, Software Metrics

I. INTRODUCTION

Software need to be changed by time to time for different reasons such as requirement change, technology change, cost-benefits change. Developers must modify software timely. Sometimes, little modified code in software loses good design of software and leads to degrade maintainability. One of the techniques that upgrade maintainability is refactoring. Refactoring modifies the internal code structure of any project system without affecting the external behavior of the system to improve the quality of the design.The process of refactoring has three distinct stages to its application: identify location where to apply a refactoring, choose an appropriate refactoring technique as a solution and apply therefactoring. [2]

A. Bad Smells in Code

If we implement refactoring alone on code, then will not benefits of doing it, until we do not find the correct location where we should apply refactoring. For the easiness of developer to find the correct location for applying refactoring ,fowler give a idea of bad smells .Bad smells is the symptoms of bad design .Bad smells does not effect on code physically , it only degrade the quality of software by timely.Here we focus our attention on code smells and on window base GUI application developed for their detection. Code smells are structural characteristics of software that may specify a code or design problem but they do not produce run time error and can make software hard to understand and maintain.

Theconcept of bad smells was given by Fowler, who defined many kinds of smells. As programmer detectbad smells in code, they should examine whether their presence hints at some relevant degradation in the design of the code, and if it found then decide which refactoring should be applied on code.

B. Some bad smells are summarized below:

- *Long Method:* when method is too long means more number of lines of code.
- *Large Class:* Classes that have large numbers of instance variables and large number of lines of code. Sometimes they are only used occasionally large classes can also suffer from code duplication.
- *Long Parameter List:* Long parameter lists are hard to understand. Long parameter list means that a method takes too many parameters.
- *Comments:*If the comments are present in the code more than the lines of code.
- *Switch Statements:* Switch statements may produce duplication. You can find similar switch statements scattered in the program in several places. . Maybe classes and polymorphism would be more appropriate
- *Lazy Class:*Classes that are not doing much work and number of method is null.
- *Temporary Field:* when some of the instance variables in a class are only used occasionally.[5]

II. LITERATURER SURVEY

It presents about the previous studies of evaluating detect code smells, refactoring and object oriented metrics: what the other researchers have done regarding code smells detection.

Jan Verelstet al. discusses the guidelines the improvement of coupling and cohesion for specific refactoring. This elaborates on a validation of these guidelines regarding their improvement and applicability. Unfortunately, refactoringconcentrate on the treatment of symptoms (the so called code-smells), thus improvements depend a lot on theskills of the maintainer .Therefore, this paper analyzeshow refactoring manipulate coupling/cohesion characteristics,and how to identify refactoring opportunities that improve characteristics [1]**KarnamSreenu**et al.

 Copyright 2014.All rights reserved.

discussesthe Merge Class Refactoring method and Replace Temp Refactoring method will be identified by providing metric values based on Number of Methods (NOM), Instance Variable per Method in Class (IVMC) and some existing metrics like Lines of Code (LOC), Depth of Inheritance (DIT) . By identifying these metric values we can apply the above two refactoring methods directly on the source code to reduce the total number of lines of code (LOC) and to improve the structure of existing code. [2]**Marco Zanonia**et al. This paper presented a comparison of four code smell detection tools on six versions of a medium-size softwareproject. We observed that we have the best agreement in the results for the God Class detection and then for the Large Class and Long Parameter List smells.In this paper we focus our attention on code smells and on automatic tools developed for their detection. [3]**AmjanShaik**et al. (2010) discusses that the about designing and development of Object Oriented (OO) have become popular in today's software development environment. Design metrics play a vital role in helping developers to appreciate design aspects of software i.e. improve software quality and developer productivity. This paper, is trying to edify about the OOD, metrics, quality and the relationship between these. In this paper, the empirical evidence underneath the role of OOD Metrics specifically a subset of the CK Metric suite ids provided. [4].

Ganesh B. Regulwar et al. 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 (Fowler & Beck 2000) give a list of bad code smells. 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. . In addition, for each bad code smell Fowler (Fowler 2000) introduces a set of refactoring (move methods, inline temp, etc.) with step wise instructions on how each smell can be removed. Therefore, the reader should realize that the refactoring concept also includes detailed instructions on how to actually improve the source code [5]

III. PROPOSED WORK

Window base GUI application has been developed to detect bad smells.It detects more bad smells according to their Object Oriented Metrics like Coupling, Depth of inheritance, Weighted Methods, Number of Methods (NOM) and Instance Variable in a Class. Long method, Lazy class, Comment lines and large class bad smells are detected using GUI application developed. This application detects bad smells on both java source code and .net source code. Also provides a bad smell description framework and bad smell interpretation framework to collect the information regarding bad smells. These frameworks mainly contain two parts.

A. Bad Smell Description Framework:

- *Bad Smell Name*: It is the description of the bad smell which is going to detect.
- Identifying main characteristics from description of the bad smell.

B. Bad Smell Interpretation Framework:

- *Bad Smell Name*: It is the description of the bad smell which is going to detect.
- *Measurement Process*: Describe possible measurement metrics that when applied to source-code can help identify the problem.
- *Interpretation Rules*: The interpretation indicates a set of rules on how the metrics can be used to identify possible problem. We are using existing metrics and new metrics to identify bad smells "lazy class" ,"long method"," comment lines " and "large class"

IV. EXPERIMENTATION

The experimentation done is as following:

Tool used to create window base GUI application: visual studio ultimateA window base GUI application is developed through thetool: visual studio 2010 .this application is created in the c#.net. This application is developed for detecting the various code smells according to their metrics rules. Source code of a project is taken in dotnet language The project of BANKING system is taken in dotnet language. The BANKING system contains 22 classes. The source code of all the classes is in dotnet language.

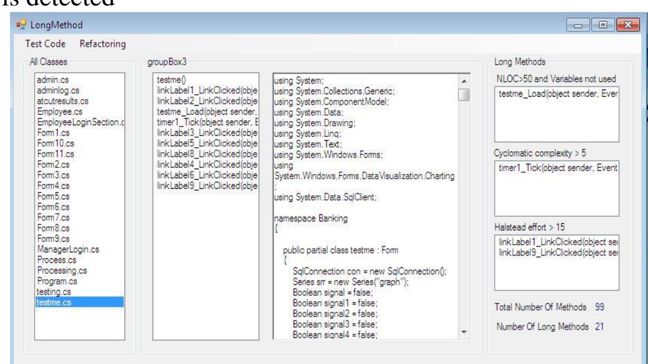
Tool used detects the code smells of the existing code: window base application developed in visual studio tool This window base GUI application is used to detect the code smells of a source code. In this, different types of metrics are measured according to code smell. Each bad smells has different metrics rules.

A.Long Method

The following metrics for the detection of long method smell: [3]

- *Rule 1*: If Number of line of code (NLOC) is greater than 50 and variables declared are not used.
- *Rule 2*: If Cyclomatic complexity is greater than 5(if else).
- *Rule 3*: Halstead effort E=D*V should be lower. (operators and operands)

If anyof the above rules is/ are true, Long method bad smell is detected



 Copyright 2014.All rights reserved.

Fig. 1 Detection of long method bad smell

B. Large Class

The following metrics for the detection of large class smells: [3]

- *Rule1*: If number of lines of code are greater than 300 and has more than 5 long methods.
- *Rule 2*: If Number of instance variables and methods are greater than 15 and 10 respectively.
- *Rule 3*: if depth of inheritance tree (DIT)>3
- *Rule 4*: coupling, if number of operation calls and number of called classed are high.

If anyof the above rules is/ are true, Large class bad smell is detected.

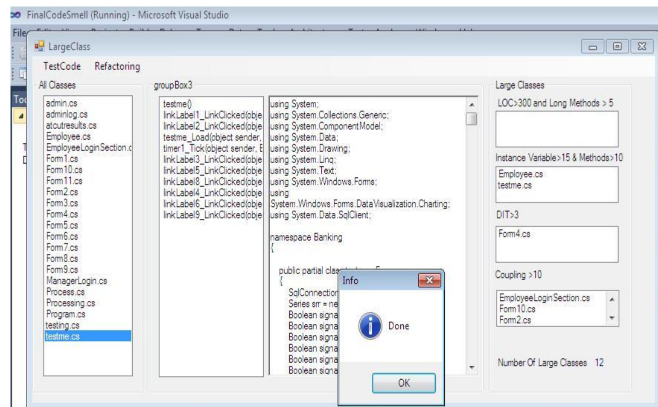


Fig. 2 Detectionof large class bad smell

C. Lazy Class

The following metrics for the detection of lazy class smells: [3]

- *Rule 1*: If number of method=0.
- *Rule2*: if LOC=100 and WMC/NOM<=2

If anyof the above rules is/ are true, Lazy class bad smell is detected.

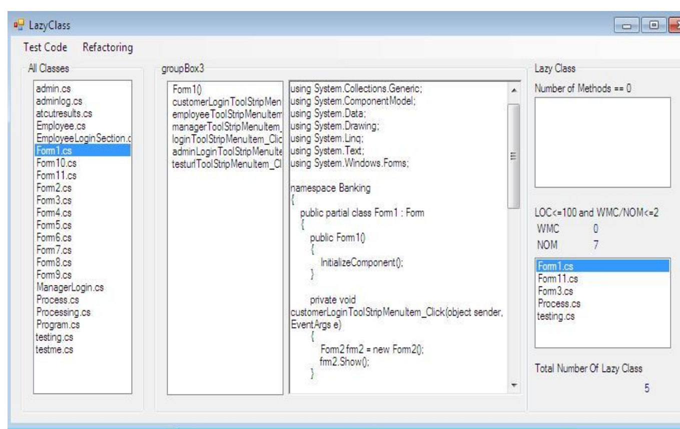


Fig. 3 detection of lazy class bad smell

C. Comment Lines

Rule: If comment lines are greater than or equal to Average lines of code

Average lines of code= (lines of code/3 in C)

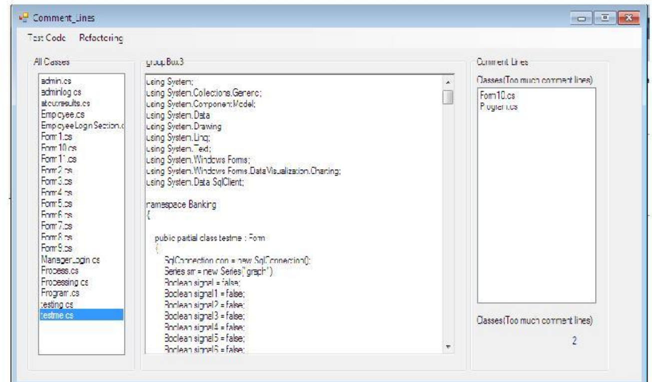


Fig. 4 Detection of comment lines bad smell

V. RESULTS AND DISSCUSSIONS

A. Case Study:

The case study taken for detection of bad smells is the BANKING system project in dotnet language. The four bad smells are detected in the banking system source code using GUI application developed. The following metrics (CC, NLOC, DIT, coupling, WMC and Halstead effort) in dotnet are implemented tofind out the four bad smells in the source code.

B. Various metrics that are being used are :

- Depth of inheritance tree: The depth of inheritance hierarchy is defined as the maximum length from the class node to the root of the tree and is measured by the number of ancestor classes.
  The class form4.cs has DIT >3 (rule of bad smell "large class")
- Coupling: The measure of the module's independence, fewer parameter flowing into or out from a module imply looser coupling.
  The classesform7.cs, form9.cs, form10.cs, form2.cs and managerlogin.cs have coupling> 10 (rule of bad smell" large class")
- Weighted Methods: WCM calculate the addition of complexities of all class methods and some of instance variables are not accessed in class. The WCM basically measure by calculating cyclomatic complexity. It tells maintainability of class also.The classes form1.cs ,form11.cs ,form3.cs ,testing.cs and process.cs have WMC/NOM <=2 (rule of bad smell" lazy class")
  NOM =number of methods
- Halstead measures: which measure the number of operators and operands, and can provide necessary information on method complexity.
  The classes testme.cs, form9.cs, form4.cs and admin.cs have halstead efforts >15(rule of bad smell "long method")
- Cyclomatic complexity: It represents the total number of regions present in the flow graph of a program. Cyclomatic Complexity is a metric of complexity that counts the number of independent paths.

 Copyright 2014.All rights reserved.

The classes Admin.cs, employe.cs, form6.cs and testme.cs have cyclomatic complexity >5 (rule of bad smell “long method”)

- Lines of source code: Lines of Code, usually referring to non-comment lines, meaning pure whitespace and lines containing only comments are not included in the metric.

  The classes form7.cs, form5.cs,form4.cs, form2.cs, employe.cs and testme.cs have NLOC >50 (rule of bad smell “long method”)

- Lines of comment: the lines of comment are used to describe the meaning of the statements which is specified.

  The classes form10.cs and program.cshave comment lines which are greater than or equal to Average lines of code.

TABLE 1 value of detected bad smells

Name Of Bad Smell	Description Of Bad Smell	Value Of Bad Smell
Long Method	Too long method that is difficult to understand	21
Large Class	classes have too many instance variables, methods	12
Lazy Class	A class having little functions	5
Comment Lines	More comment lines are present in code	2

VI. CONCLUSION

The four bad smells are detected in the banking system source code using GUI application developed. The measured object oriented metrics shows the value of each metric in their respective code smells detected on the coding. Calculated metric values will help in applying the refactoring methods directly on the source code to eliminate the bad smells and to improve the structure of existing code. Like Coupling factor will be helpful to decide whether we can apply "MOVE" method of refactoring or not .The value of NLOC, Cyclomatic complexity and Halstead effort will be helpful in applying Extract Method and Replace Temp with Query methods of refactoring.

VII. FUTURE SCOPE

- In the future the comparison will be performed between developed window base GUI application and Eclipse tool.
- Refactoring methods will be applied on the basis of calculated metrics on source code to refactor bad smells.

REFERENCES

[1] Jan Verelst , Bart Du Bois and Serge Demeyer, “Refactoring - Improving Coupling and Cohesion of Existing Code,”

[2] KarnamSreenu 1, D. B. Jagannadha Rao2 “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

[3] Francesca ArcelliFontanaaPietroBraionea Marco Zanonia, “Automatic detection of bad smells in code: An experimental assessment ,” Journal of Object Technology Published by AITO Association Internationale pour les Technologies Objets, c JOT 2011

[4] AmjanShaik, C. R. K. Reddy, BalaManda, Prakashini. C, Deepthi. K, “An Empirical Validation of Object Oriented Design Metrics in Object Oriented Systems,” Journal of Emerging Trends in Engineering and Applied Sciences (JETEAS), 2010.

[5] Ganesh B. Regulwar and Raju M. Tugnayat “Bad Smelling Concept in Software Refactoring ”, Jawaharlal Darda Institute of Engineering & Technology, MIDC, Lohara, Yavaymal (MS), INDIA.

[6] FOWLER, MARTIN: A list of refactoring tools http://www.refactoring.com/tools.html

 Copyright 2014.All rights reserved.