ISSN 2395-695X (Print)
ISSN 2395-695X (Online)
International Journal of Advanced Research in Biology Engineering Science and Technology (IJARBEST)
Vol. 2, Special Issue 10, March 2016

A METHODOLOGY FOR DETECTING CODE SMELLS ACCORDING TO OBJECT ORIENTED METRICS

S.Nandhini[1], Dr.E.Baby Anitha[2]
1,2Department of Computer Science & Engineering
1,2K.S.R College of Engineering, Tamilnadu, India

ABSTRACT:

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 “Data class”, “long method”, “Too Many Parameters”, “Shotgun surgery”, “Feature Envy”.

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

I. INTRODUCTION

In computer programming, code smell is any symptom in the source code of a program that possibly indicates a deeper problem. Another way to look at smells is with respect to principles and quality "smells are certain structures in the code that indicate violation of fundamental design principles and negatively impact design quality". Code smells are usually not bugs they are not technically incorrect and do not currently prevent the program from functioning. Instead, they indicate weaknesses in design that may be slowing down development or increasing the risk of bugs or failures in the future.

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

All Rights Reserved © 2016 IJARBEST

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.

B. Some Code smells are summarized below

- Long Method: when method is too long means more number of lines of code.
- Long Parameter List: Long parameter lists are hard to understand. Long parameter list means that a method takes too many parameters.
- Data Class: Classes with fields which may get and set methods for fields and nothing else.
- Shotgun surgery: A single Change in a class requires a lot of little changes in several classes.
- Feature envy: A method seems to be more interested in another class than its actual residing class.

II. RELATED WORK

Gabriele Bavota et al. [4] focuses on the system analyzes the impact of Application Program Interface (API) changes and faults in Android environment. The goal of this study is survey Android developers, with the purpose of understanding to what extent they experience problems when using APIs and how much they consider these problems to be related with negative user ratings comments. Hence, the study quality mainly focuses on perception of the developer ability for the impact change- and fault-prone APIs found on the apps' user ratings. Design factors are not focused in the system.

Abdelilah Sakti et al. [5] address a new automated SBST (search-based software test-data generation) approach for unit-class testing to achieve high code coverage. The proposed instance generator allows our approach to better explore the search space, reaching more test targets, thus increasing code coverage in less time. To improve over random testing, global and local search algorithms have been implemented in several ways. Despite of all the features, code smell discovery was not performed.

All Rights Reserved © 2016 IJARBEST

David Notkin et al. [3] presented a technique called codebase replication which simplifies the implementation of continuous analysis tools by converting an existing offline analysis into an IDE-integrated. Isolation and currency are considered as the two desirable properties of a continuous tool. Codebase Replication creates and maintains a copy of the developer's codebase. The analysis runs on the copy codebase without being disturbed by the changes made by the developer. Version based history level analysis was not performed.

Wael Kessentini et al. [7] consider detection of code-smells as a distributed optimization problem. During the process of optimization dissimilar methods are combined in parallel to find a consent regarding the detection of code smells. In order to perform the above stated basic idea Parallel Evolutionary algorithms (P-EA) is used. In P-EA many evolutionary algorithms with different adaptations such as fitness functions, solution representations, and change operators are executed, in a parallel cooperative manner, to serve a common goal called code-smell detection.

Fabio Palomba et al. [1] proposed a Historical Information for Smell detection(HIST), an approach exploiting change history information to detect instances of five different code smells, namely Divergent Change, Long Method, Data Class, Too Many Parameters, Parallel Inheritance, Blob, and Feature Envy. Code smells are detected using structural information identified from version histories. Structural and lexical information are analyzed with rule definitions to discover the code smells. Rules are represented with metric factors with threshold values. Detection algorithm uses the code components such s methods and classes for code smell detection process. Apriori algorithm is used to extract code smells from change history data values. Minimum support and confidence values are used to filter out the code smells.

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 Weighted Methods, Number of Methods (NOM) and Instance Variable in a Class. Long method, Data class, Too Many Parameters, Divergent Change, Parallel Inheritance, Feature Envy and Blob 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

All Rights Reserved © 2016 IJARBEST

ISSN 2395-695X (Print)
ISSN 2395-695X (Online)
International Journal of Advanced Research in Biology Engineering Science and Technology (IJARBEST)
Vol. 2, Special Issue 10, March 2016

smell description framework and bad smell interpretation framework to collect the information regarding bad smells.

A. Code Smell Description Framework:

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

B. Code Smell Interpretation Framework:

- Code 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.

IV. EXPERIMENTATION

The experimentation done is as following:

Tool used to create window base GUI application: visual studio ultimate window base GUI application is developed through the tool: 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. Each bad smells has different metrics rules.

A. Lazy Class

The following metrics are used for the detection of lazy class:

- Rule 1: If number of method=0.
- Rule2: if Class has instance variables, getters, and setters.

If any of the above rules is/ are true, Lazy class code smell is detected.

B. Long Parameter List

The following metrics are used for the detection of long parameter:

- Rule 1: If a method contains parameters >3.
- Rule 2: If consists of Unused parameters(declared in a method but never used)

If any of the above rules is/ are true, Long Parameter List code smell is detected.

All Rights Reserved © 2016 IJARBEST

ISSN 2395-695X (Print)
ISSN 2395-695X (Online)
International Journal of Advanced Research in Biology Engineering Science and Technology (IJARBEST)
Vol. 2, Special Issue 10, March 2016

C. Long Method

The following metrics are used for the detection of long method:

- Rule 1: If Number of line of code (NLOC) is greater than 50 and variables declared are not used.
- Rule 2: If a method in a code contains more than 3 looping statements (if else).

If any of the above rules is/ are true, Long method bad smell is detected.

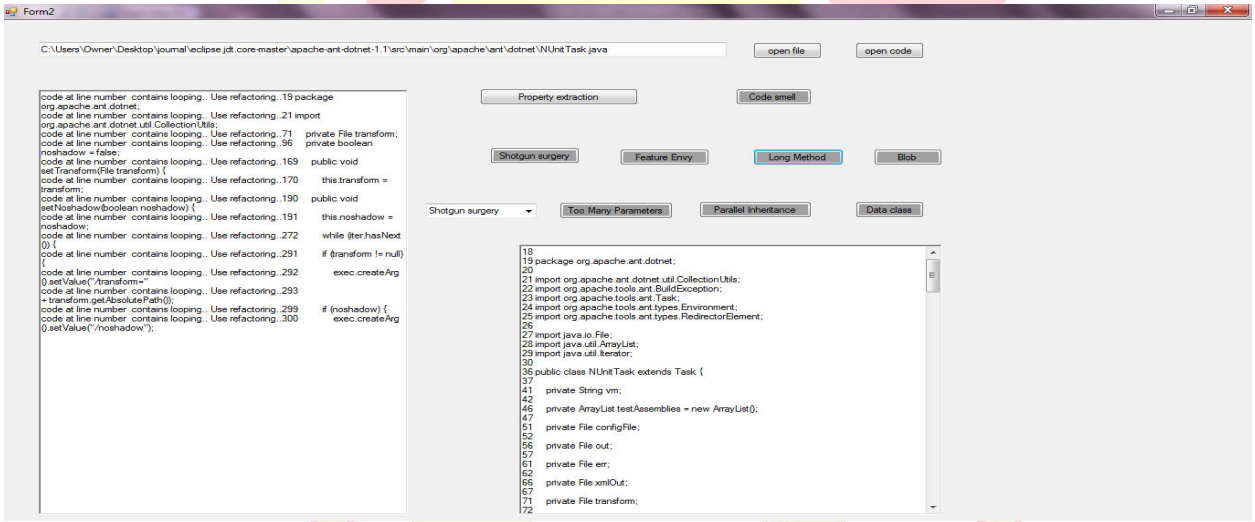

Fig. 1 detection of long method code smell

D. Shotgun Surgery

The following metrics are used for the detection of shotgun surgery class smell:

- Rule 1: If making a single Change in a class requires a lot of little changes in several places.
- Rule 2: If co-changes observed is found to be more than 2.

If any of the above rules is/ are true, Shotgun surgery code smell is detected.

E. Feature Envy

The following metric is used for the detection of feature envy class smells:

- Rule 1: If a method uses the properties of another class than its own.

All Rights Reserved © 2016 IJARBEST

ISSN 2395-695X (Print)
ISSN 2395-695X (Online)
International Journal of Advanced Research in Biology Engineering Science and Technology (IJARBEST)
Vol. 2, Special Issue 10, March 2016

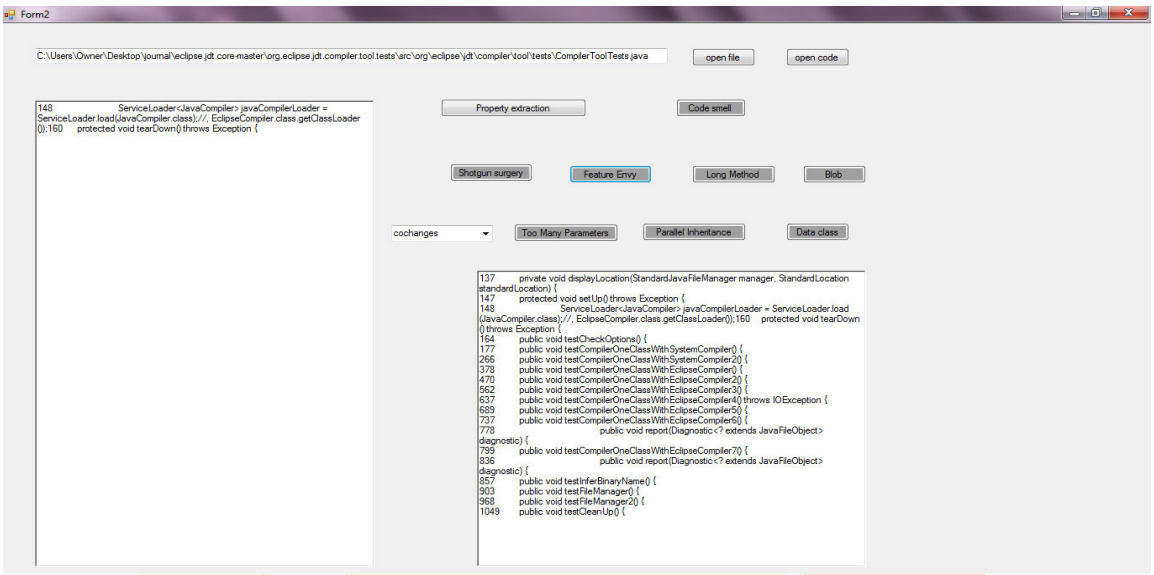

Fig. 2 Detection of Feature Envy code smell

V. CONCLUSION

The code smells are detected in the 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 code smells and to improve the structure of existing code. Refactoring methods will be applied on the basis of calculated metrics on source code to refactor code smells.

REFERENCES

[1] Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Denys Poshyvanyk and Andrea De Lucia, "Mining Version Histories for Detecting Code Smells", IEEE Transactions On Software Engineering, Vol. 41, No. 5, May 2015, pp.462-489.

[2] Seonah Lee, Sungwon Kang, Sunghun Kim and Matt Staats,"The Impact of View Histories on Edit

Recommendations", IEEE Transactions On Software Engineering, Vol. 41, No.3, March 2015, pp.314-330.

All Rights Reserved © 2016 IJARBEST

[3] Kivancs Mu slu,Yuriy Brun,Michael D. Ernst and David Notkin, "Reducing Feedback Delay of Software Development Tools via Continuous Analysis", IEEE Transactions On Software Engineering, Vol. 41, No. 8, August 2015, pp.745-763.

[4] Gabriele Bavota , Mario Linares-V_asquez, Carlos Eduardo Bernal-C_ardenas, Massimiliano Di Penta,Rocco Oliveto and Denys Poshyvanyk," The Impact of API Change and Fault-Proneness on the User Ratings of Android Apps", IEEE Transactions On Software Engineering, Vol. 41, No. 4, April 2015, pp.384-407.

[5] Abdelilah Sakti, Gilles Pesant and Yann-Gael Gu_eh_eneuc, "Instance Generator and Problem representation to Improve Object Oriented Code Coverage", IEEE Transactions On Software Engineering, Vol. 41, No. 3, March 2015, pp.294-313.

[6] Theodore Chaikalis and Alexander Chatzigeorgiou,"Forecasting Java Software Evolution Trends Employing Network Models", IEEE Transactions On Software Engineering, Vol. 41, No. 6, June 2015, pp.582-602.

[7] Wael Kessentini, Marouane Kessentini, Houari Sahraoui, Slim Bechikh and Ali Ouni, "A Cooperative Parallel Search-Based Software Engineering Approach for Code-Smells Detection", IEEE Transactions On Software Engineering, Vol. 40, No. 9, September 2014, pp.841-861.

All Rights Reserved © 2016 IJARBEST