A Study on Bad Code Smell

I M Umesh[1], Dr. G N Srinivasan[2]

[1]*Research Scholar, Bharathiar University, Coimbatore, Tamilnadu*
[2]*Department of Information Science & Engineering, R V College of Engineering, Bengaluru, Karnataka*

***Abstract*- Software quality degenerates over time due to various reasons like software ageing, inconsistent design and improper requirement analysis during early stages of software development. Bad code smell is an indication of the persistent deeper problem that may exist. Bad Code smells are neither bugs nor technically incorrect and hence do not prevent the software from normal functioning. Refactoring is the term used to describe the process of removing the bad code. This paper throws light on bad code smell and detection techniques available. The paper is divided in three sections: first we introduce code smells and methods to detect them, and then we review of various studies conducted on these bad code smells. Finally we will describe the results and discuss them.**

Key words - Bad code smell, software maintainability, software metrics, refactoring.

I. INTRODUCTION

Design issues are nothing but Bad Smells at the code level. Improvising the software structure without losing any functionality is Refactoring. The awareness on the system design will throw light on probable errors and predict the possible failures.

A set of Software metrics is used to identify the bad smell in code that may cause frequent failures and associated costs. If left unattended, bad smells can consume lots of resources in terms of maintenance costs, testing etc., Removing bad smells from the code makes software more maintainable as bad code smells are a new measure of software maintainability.

Removal of code smell is identified as a way to improve design standard of software. Detection of bad code in huge systems remains time consuming and prone to error. This may be due to the lack of adequate tool support. Many researchers have done lot of work on detection of bad code smells.

Bad code smells indicate the trouble and they are only guidelines and can not be used as directives. There are various tools available for detection of bad code smells. An eclipse plug-in called JDeodorant traces bad smell and applies some refactoring to resolve them.

Checkstyle is a tool that can help programmers writing java code that adheres to a coding standard that is capable of detecting bad smells like Large Class, Long Method..

CodeNose was developed in 2005 as a prototype for automated code smell detection due to the lack of tools at that time. It is implemented as a plug-in for eclipse which detects and presents code smells similar to how compilation errors and warnings are displayed.

Following table depicts the various bad smells and its description as given by Fowler etal. [1].

Type	Description
Duplicated code	Repeated appearance of code structure
Long method	Too long method
Large class	The Classes with too many instances, variables and methods
God class	Class that tends to centralize the intelligence of the system
Long parameter list	A long list of parameters in a procedure or function
Feature envy	More tightly coupled class in wrongplace
Contrived complexity	Complicated design pattern
Complex conditionals	Checks for unrelated conditions
Primitive obsession	Primitives are used instead of small classes
Switch statement	Instead of polymorphism, runtime class are used.
Data clumps	Data items that often appear Together
Temporary fields	Class having very rarely used variable.
Refused bequest	Child class does not fully support all the methods or data it inherits
Lazy class	A class which does nothing enough and needs removal.
Data class	A class that contain data without any logic.
Middle man	A class that delegates most of its tasks.
Divergent change	The class that needs frequent changes for different reasons.

II. BAD CODE SMELLS

Bryton et al. [2] demonstrated in their work that the Long Method code smell can be detected automatically and objectively. But this model is restricted and cannot be generalized as the calibration was performed on a particular project and its detection capability was limited to that project. The authors suggested automatable process validity for detection of code smells.
The statistical techniques were used to obtain a mathematical model which is capable of detecting Long Method instances upon source code analysis.

An Eclipse plug-in was developed to detect and asses he code smells in java code by Tiago Pessoa [3]. The statistical detection algorithm was built on Binary Logistic Regression Model. The SmellChecker is a prototype version of the tool which was developed as an Eclipse plug-in to detect code smells in java code which allows smell tagging and visualization.

FoutseKhomh [4] in their research identified code smells in almost 9 releases of Azureus and 13 releases of Eclipse. The study was conducted to understand the relationship between bad code smells and change proneness. It was proved that in all Azureus and Eclipse, code smells are more correlated to change proneness than others. The empirical evidence of negative impact of code smells on change proneness was revealed by their study. The classes with smells are considerably subject to changes than others. Some specific code smells, are more likely to be cause of concern during evolution.

The study on Code Smell effects on Maintenance were conducted by Dag I.K. Sjøberg [5]. The study involved in quantifying the relationship between code smells and the effort of maintenance in the industrial environment.

The study was conducted on java systems which are developed independently and are functionally equivalent. Maintenance tasks were performed on these systems that include platform adjustment and functionality for tailored reports. The time spent on each of the file by the developers were recorded and analyzed whether the number of smells in the file affected effort.

The findings of the study include some inconsistently maintained duplicated code lead to more change effort than that of the copied code. However, there exist no strong evidences of associated duplicated code with defective code. The presence of bad code smells alone won't affect comprehension but their combination. The study indicated that combination of bad code smells increases the effort of the developers on comprehension tasks.

In an Industrial-strength Open Source system, an investigation was carried out to ascertain the relationship between the bad smells and class error probability in three error severity levels by Preet Kamal Dhilonet [6]. The Research showed that the Bad Code Smells are associated with the class error probability in the context of post release system evolution process.

III. DETECTION METHODS

The tools for detection of bad code smells employ different methods to detect bad code smells. Some of them are discussed here.

- JDeodorant is an eclipse plug in tool that can recognize opportunities for extracting cohesive classes from "God Classes" and automatically apply the refactoring chosen by the developer.
- inCode works in the background of eclipse. During programming, if programmer writes any bad structure, than it shows these smells as, "eclipse show error" and warnings in the shape of red color blocks along with code.
- PMD traces dead code, empty catch or switch statements, the variables that are not used or duplicated code in the java source code. This toll is capable of detecting bad smells like Long Parameter List, Duplicated Code Smells.
- FindBugs can be used to detect java program bugs. It can detect common coding mistakes like thread synchronization issues and also misuse of API methods.

IV. DISCUSSION AND CONCLUSION

It is required to understand what code smells are and why they are bad, then one can better judge whether source code should be refactored. Software refactoring is widely used to delay the degradation effects of software aging and facilitate software maintenance.

The studies are focused more on developing tools to detect bad code smells. Very few studies report the impact of bad code smell on software performance. This indicates an important gap in the current knowledge of Bad Code Smells.

The percentage of share of detection tools is shown in the graph below.

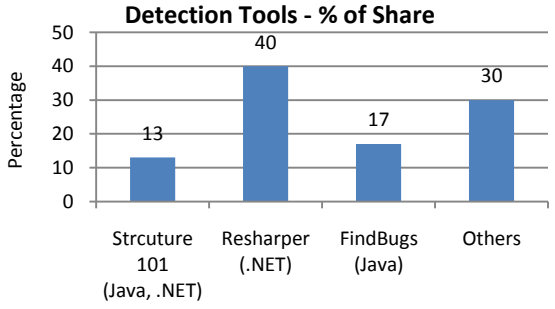

Figure 1 Detection tools: Percentage of share

Several studies are conducted on refactoring of the object oriented programming. Many software tools are available for the automated detection of bad smells. The tools differ in their approach and their capability. Some of the tools available are shown in Table.

Table 1 Bad code detection tools

Tool	Capability
Structure101	Helps in influencing the architecture when the code is edited wherein the architecture can be changed without disrupting the code.
ReSharper	Visual Studio plug in that can analyze several thousands of lines of code quickly
FindBugs	Used with eclipse IDE and it is the software used to find bugs in Java programs.
JDeodorant	Can detect Feature Envy, Type Checking, Long Method and God class.
inCode	Can detect Feature envy, God class, Duplicate code and Data class.
JDEvAn (Java Design Evaluation and Analysis)	Evaluates a design evaluation history of software system and provides the information about the system history

A feature that is missing amongst most of the smell detectors is to perform refactoring directly as a solution to a detected smell. Determining whether some piece of code contains bad smell(s) is somewhat subjective and still there is a lack of standards.

REFERENCES

[1]. FOWLER, M., BECK, K., BRANT, J., OPDYKE, W. & ROBERTS, D. (1999) Refactoring: Improving the Design of Existing Code, Addison Wesley

[2]. Bryton, S., Abreu, F.B., and Monteiro, M., 2010. ―Reducing Subjectivity in Code Smells Detection: Experimenting with the Long Method , Seventh International Conference on the Quality of Information and Communications Technology, Vol., pp. 337-343.

[3]. Tiago Pessoa, "An Eclipse Plugin to Support Code Smells Detection", INFORUM'2011 conference proceedings, Luis Caires e Raul Barbosa (eds.), 8-9 September, Coimbra, Portugal, 2011

[4]. FoutseKhomh, An Exploratory Study of the Impact of Code Smells on Software Change-proneness , Proceeding WCRE '09 Proceedings of the 2009 16th Working Conference on Reverse Engineering Pages 75-84 IEEE Computer Society Washington, DC, USA 2009

[5]. Dag I.K. Sjøberg, "Quantifying the Effect of Code Smells on Maintenance Effort", IEEE Transactions on Software Engineering, Volume 3, Issue 36

[6]. Preet Kamal Dhillon, GurleenSidhu, "Can Software Faults be Analyzed using Bad Code Smells? : An Empirical Study", International Journal of Scientific and Research Publications, Volume 2, Issue 10, October 2012

[7]. A. Deursen, L. Moonen, A. Bergh, and G. Kok, ―Refactoring test code,Proceedings of the 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2001), M. Marchesi and G. Succi, Eds., May 2001