Automated Approach for Anti-Pattern Detection

Neha Nanda
Computer Science and Engineering
Lovely Professional University, Phagwara, Punjab, India
nehananda50@gmail.com

Rohitt Sharma
Computer Science and Engineering
Lovely Professional University, Phagwara, Punjab, India
rohit.17458@lpu.co.in

www.ijcaonline.org

I. ABSTARCT

Poor design choices that tell us how to move from a problem to a bad solution are known as anti-patterns. It can also be described as a common design pitfall. In our work we investigate the impact of anti-patterns on classes in object-oriented systems by studying the relation between the presence of anti-patterns and the change and fault proneness of the classes, devise a tool for automatic detection of anti-patterns and refactoring the code post detection. Due to increased complexities in the software development and increasing of anti-patterns in the software development, there is a huge need of testing process to be carried out in a better and effective way.

II. INTRODUCTION

Design patterns are known as the good guys focussing on successes which are well defined and problem based unlike anti-patterns which focus on failures which are poorly defined and solution based. Anti-patterns are known as the bad guys which are the refactored solutions that seem to be effective but may lead to bad consequences .The term anti-pattern came into existence in 1995, introduced by Andrew Koenig [1]. When a problem arises during coding, sometimes due to lack of understanding, lack of time or lack of experience we devise a solution to the problem that seems easy and effective but in turn leads to more serious problems this clearly describes the concept of anti-pattens.it may not necessarily hinder the execution of the program but may lead to other unnoticed issues one tends to ignore thus making such anti-patterns difficult to detect. These days' anti-

patterns are an active research area that extends the study of design patterns into more extensive fields. Therefore it is a rising need to detect these unsuccessful behaviors and refactor (changing the internal behavior of the code without changing the external behavior) the code to a better desirable solution thus in turn enhancing the performance and software quality may it be in terms of cost ,memory consumption or time taking for execution.

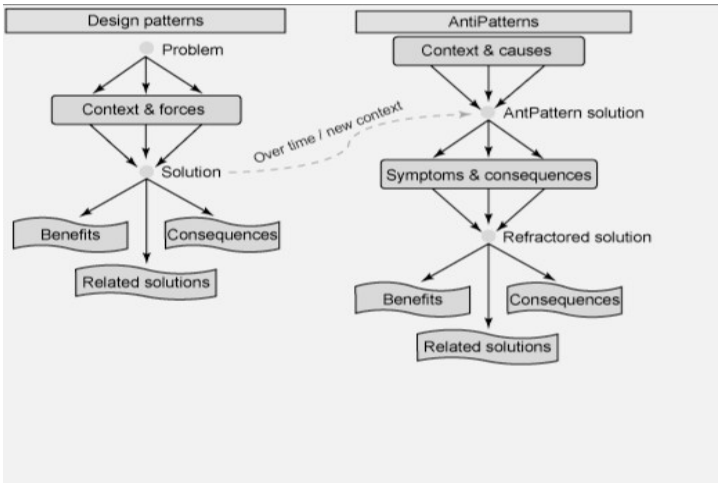

Fig 1. Design Pattern and Anti-Pattern Concept [2]

III. PROPOSED WORK

Our research will be focusing on automatic detection of three kinds of anti-patterns namely unused data, blob and cryptic code using eclipse, a java based open source platform and ArgoUML, an open source UML modeling tool which includes support for all UML diagrams. In this research we divide the process into two phases. First phase consists of reading the XMI file i.e. XML Metadata interchange, a standard for exchanging metadata information

via extensible markup language XML. Our purpose is to read the XMI generated which consists of class, methods and variables and second phase will use this collected information and serve as an input for eclipse testing module to test the anti-patterns in the given code. First we will create UML diagram for the program which is created using ArgoUML. Generated UML will be converted into XMI, this XMI will be input to eclipse testing module. We will give XMI a file path then we have to find nodes that contain class name and have to get the information about the field of these classes. Once we have provided the required input we will run our program. Program will read the xmi file from the specified location. To read xmi, first we will create a DOM Parser object that will parse the file.DOM parser reads XML; before an XML file can be accessed it must be loaded into an XML DOM parser. After reading the XML, XML parser converts it into an XML DOM object which in turn can be accessed with JavaScript. After parsing the file we will store the file in a Document object in the form of document. A Document Traversal object will be used to convert Document (xmi file is converted into document) into nodes. Each node has a name and some attributes. These node names and attributes will be used to get required information. Now we will pick one node at a time till all nodes are picked and we will get Node name and node attributes. After that it will be checked if node is specifying a class then all the following nodes that specify class attributes will be stored in a list corresponding to that class. These lists of attribute will be used to search attribute which is specifying the information about a field in the program. After getting the required information from the XMI, our java based eclipse testing module will analyze the main code and find the anti-pattern in it.

There are different types of anti-pattern, and in our research we will try to find some most common anti-patterns in the code i.e. unused data, blob and cryptic code anti-pattern that affects the quality of the code.

Field defined but not used during the execution of the code can be considered as unused code. These types of fields increase program execution time and consume unnecessary memory space which may reduce the quality of the code. Data gathered from XMI file will provide us information about the fields of the class. Our program will search these fields to find their values, if field is not used during the execution of program, that field is marked as unused. After searching through the whole program to be tested it will provide the list of all unused fields in the last.

Blob is a type of anti-pattern in which a single function defined in an application performs multiple functions. It is also known as God object. To find blob anti-pattern our tool will print method call stack. Method call stack will display the calling scenario in the program. Mostly it is expected that one method should perform one job/function. If some method is called over and over again, it may be a blob design. So by inspecting the call stack we can detect blob in our program to be tested.

Cryptic code is another type of anti-pattern which defines the abbreviations used for fields instead of proper naming. These cryptic fields make it hard to understand what type of values the field is simulating. This makes it difficult and cumbersome to reuse the code or modify the code in future. To find cryptic code our tool will define the minimum length required for a perfect name of field. The fields having character less than the specified length will be considered as cryptic code and will be added to the list which will print them after searching through the whole program.

After finding the anti-patterns code will be refactored to remove the anti-patterns from the code. This will make the execution of the code faster and will enhance quality of the code. Refactoring of the code will be carried out in three different phase. In each phase one of the anti-patterns will be removed from the code. After the completion of refactoring process, refactored code will be passed to the testing tool to check if still there is any anti-pattern left. In the end we will compare the execution time and memory consumption of refactored and non-refactored code.

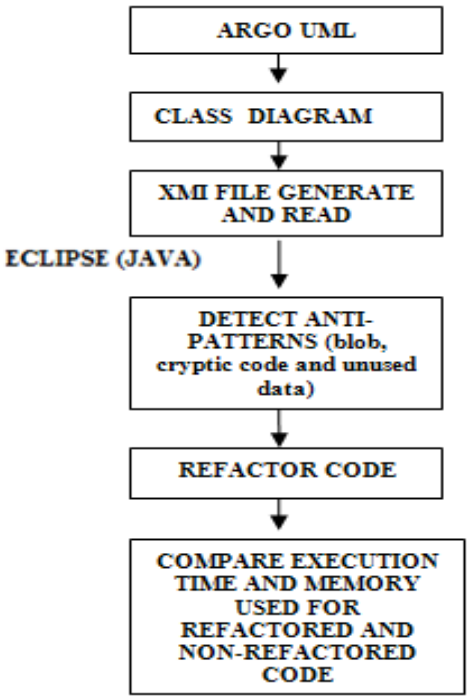

Fig 2.Flowchart to depict the methodology

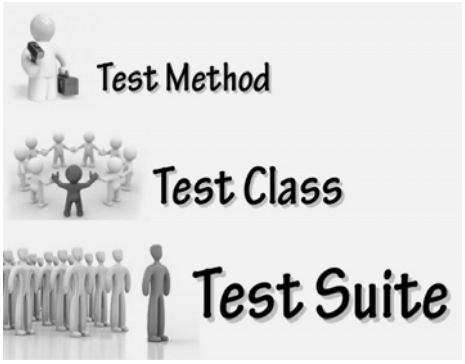

Fig 3.Test Procedure [3]

IV. ADVANTAGES OF REFACTORING

Advantages of refactoring the code that is improving the design of code without making any changes in the external behavior would include:

- Code can be highly reusable
- Code is easily maintained and upgraded
- Saves time by replacing a common problem with a definitive solution
- Maintaining consisting code standards
- Developing more trusted code
- Low defect counts
- Better understandability and readability
- Ease for automated testing
- Generating efficient and effective code
- Reduces the overall costs
- Reduces the overall memory consumption

Fig 4.Before Refactoring [4]

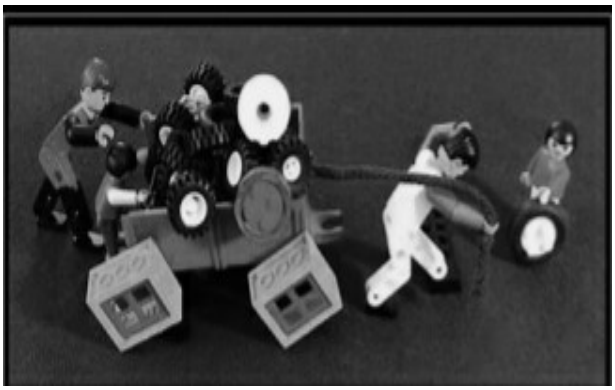

Fig 5.After Refactoring [5]

V. CONCLUSION AND EXPECTED OUTCOMES

Anti-patterns are poor design choices that are conjectured to make object oriented systems harder to maintain, increase the programming cost, increase the execution time and reduce the understanding of any program. It makes the source code difficult to understand and obstructs the development and the maintenance activities. Our approach aims at removal of anti-patterns which in turn would enhance the software quality and would also decrease the chances of any bug in the software and focuses on automatic detection and providing solution for mentioned anti-patterns through testing approaches which is the primary concern of our study. Our study aims at formulating a tool for automatic detection of anti-patterns.in short we expect program should read the xmi file from the specified location, code should be refactored by removing the found anti- patterns, The execution time of refactored code should be less than non-refactored code, memory consumed after the removal of anti-patterns should be less than the memory consumed before the removal of the anti-patterns and the anti-patterns namely blob, cryptic code and unused data should be nonexistent after refactoring of the code.

REFERENCES

[1] http://en.wikibooks.org/wiki/Introduction_to_Software_Engineering/Architecture/Anti-Patterns

[2] http://www.antipatterns.com/EdJs_Paper/Antipatterns.html

[3] http://www.slideshare.net/eduardomg23/refactoring-test-code

[4] http://www.thinkandgrowentrepreneur.com/2014/09/refactoring-and-building-next-big-thing.html

[5] http://performancemanagementcompanyblog.com/tag/facilitating-workplace-improvement/

[6] http://blog.codeclimate.com/blog/2014/04/01/launching-today-automated-refactoring/

[7] http://rigor.com/blog/2014/01/benerefactor

[8] https://medium.com/things-developers-care-about/what-is-quality-code-4c07a0a3653

[9] Li Bao-Lin, Li Zhi-shu, Li Qing, Chen Yan Hong , ” Test Case automate Generation from UML Sequence diagram and OCL Expression”, International Conference on Computational Intelligence and Security 2007, pp 1048-52

[10] http://java.dzone.com/articles/design-patterns-pattern-or

[11] http://c2.com/cgi/wiki?AntiPattern

[12] M. Fowler,” Refactoring – Improving the Design of Existing Code”, 1st ed. Addison-Wesley, June 1999.

[13] Abdou Maiga, Nasir Ali, Neelesh Bhattacharya, Aminata Sabané,Yann-Gaël Guéhéneuc,Giuliano Antoniol, and Esma Aimeur (2012), "Support Vector Machine For Anti Pattern Detection", pp 1-4.

[14] Aminata Sabane," A Study on the Relation Between Antipatterns and the Cost of Class Unit Testing", European Conference on Software Maintenance and Reengineering, July, 2013.

AUTHORS PROFILE

Neha Nanda is pursuing Post Graduation in Computer Science from Lovely Professional University. She is undergoing B.tech-M.tech Dual degree in Computer Science and Engineering.

Rohitt Sharma is Post Graduate in Computer Science and is currently working as Assistant Professor at Lovely Professional University. He has a teaching experience of around 3 years. He has 3 publications in his name including journals and conferences.