International Journal of Engineering Applied Sciences and Technology, 2018
Vol. 3, Issue 8, ISSN No. 2455-2143, Pages 55-60
Published Online December 2018 in IJEAST (http://www.ijeast.com)

ACO OPTIMIZED FACT AND RULE BASED BAD SMELL DETECTION TECHNIQUE FOR OOPS PROGRAMMING

Sunita Sharma
Department of CSE
SVIET, Banur, Punjab, India

Er. Manmeen Brar
Department of CSE
SVIET, Banur, Punjab, India

***Abstract*— Code smell (CS) is a sign that tells something has gone incorrect, somewhere in the code. Such problems are neither bugs nor they are technically wrong. Moreover, they do not prevent the program from its functioning. CS indicates the flaws in the design that may be a reason to slow down the development in the near future. From software engineer's perspective, detecting CS remains major concern so to enhance maintainability. However, it is a time consuming task. Current CS detection tools are not equipped with functionality to assess the parts of code where improvements are required. Hence, they are unable to re-factor the actual code. Further, no functionality is available to permanently remove the CSs from the actual code thereby increasing the Risk factor. In this research work we find the bad smells in the code form like unused empty- catch, unused variable, dead code ,switch statements and long method. In this approach implemented to detect the bad smell in two ways i.e., fact and rules and Ant colony optimization algorithm. Our proposed algorithm is an OOPs based concept which supports multiple languages (C,C++, Java and .Net). Bad smell means to code in a wrong way. It is not a standard form. Bad smell may lead to overall performance reduction in the software system leading to high time consumption, high space complexity, high software maintenance cost etc. Therefore, to detect the bad smells to reduce these kinds of parameters with false acceptance rate, false rejection rate and accuracy and compared with existing parameters.**

Keywords – **Code smell, object-oriented programming, optimization, software maintenance, OOPS Metrics.**

I. INTRODUCTION

Code Smells are the unwanted parts of code which may serve as fertile ground for errors. They can be because of lack of time, clear requirements , experience, proper testing etc. These smells can be detected manually or automatically with a tool. Manual detection is static and performed by the developer itself or by special team. Many formal or informal techniques are used for this purpose. Every detail can be noted down and alternates can be suggested for the code smells detected. There are various pros and cons of both manual and automatic detection. Manual detection is time consuming, tiresome and may miss some smells but it is considered to be more accurate as only the humans should take the final decision about refactoring the code. Automatic Detection using tools is easier, consumes less time, is cost effective but can be error prone as it may consider some code weaknesses falsely to be errors. Here we have developed a tool for the detection of some major Code Smells like Long Method, Dead Functions, Un-Used Variables, Un-used Catch, Switch Statements.

1.1 Overview of Detection Methods

Detection of code smells from the source code can be performed using various techniques. We achieve this using Facts and Rules. All the statements in the source code are traversed . The statements which are actually keywords in the language used like if,else,for,while etc are not included while calculation of FAR and FRR. Symbols are also excluded like }, {, ; etc. An Abstract Syntax Tree is build and studied or parsed to find the exactly matching smells. To do this facts and rules are applied along with Ant Colony Optimization algorithm . We use ant colony algorithm to optimize the results. As shown in the figure below when the code is queried for a matching syntax (which are actually code smells), facts and rules are applied and if a match happens it indicates that a code smell has been successfully detected.

International Journal of Engineering Applied Sciences and Technology, 2018
Vol. 3, Issue 8, ISSN No. 2455-2143, Pages 55-60
Published Online December 2018 in IJEAST (http://www.ijeast.com)

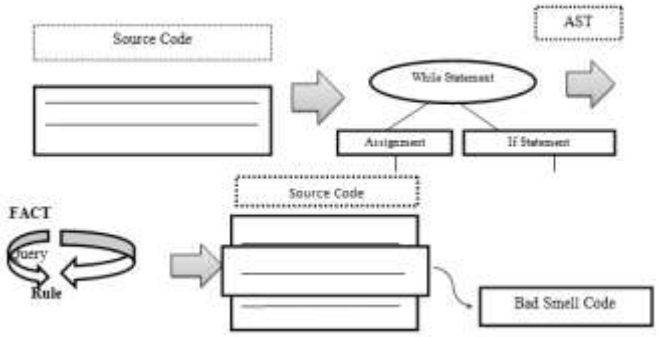

Fig. 1. Detection Method

1.2 Refactoring

Factoring is a term used in mathematics to divide or express any term in to its multiples. It expresses the same statement in a much cleaner manner. Same is the gist of Refactoring : Clean Code. Fowler's work inspired the IT industry to use refactoring technique on code. Fowler mentions 4 advantages of code Refactoring:

- **Improvisation in design** – Code starts to decay in the process of modifications as the structure is changed. Refactoring confirms that the process actually makes the code more portable, scalable and flexible.
- **Ease of understanding** – Simplicity or Ease of understanding the code is an important and reach of the product to the masses depends on the understanding of the product.
- **Detecting Defects** – Better code structure is helpful in detecting the defects that are currently latent or masked and are not causing any problem but may cause a potential threat in future.
- **Efficiency** – Performance of the product basically depends on the usage of standard and fast coding practices.

Refactoring is the most basic process used in industry. For better rationality, modularity, portability, readability and robustness refactoring is a must. Refactoring of code is now moving towards automatic detection but it still needs manual intervention.

1.6.1 Techniques of Refactoring

- Move Method
- Extract Class
- Pull Up Method
- Extract Method
- Replace Temp with Query
- Inline Method.
- Replace Array With Object
- Inline Class

1.3 Ant Colony Optimizaton Algorithm

Ant Colony Optimization (ACO) combines proposed promising solutions with already existing ones. We aim to achieve the best solution by merging the already available solutions with the most probable solutions and keep on rejecting until we achieve the best answer to our questionThe main characteristic of ACO algorithms is that they try to make the best possible use of the available information .

II. RELATED WORK

W. Abdelmoez et al., 2014 [40] considered the concept of Risk. The tool that he developed was based on risk to detect code smells. He uses the tool to study the problems encountered in c# code. He detects 4 code smells which are empty catch,Message chain, Long Parameter list and Long Method. Risk level increases with the increase in the frequency of ocuurence of bad smells. Risk level also increases if the smell is strong.Anshu Rani et al.,2014[41] Describes that refactoring improves performance of the code without making any changes to its behaviour. It removes the bad smells and thus contributes in increasing the maintainability of code. He also mentions that most of the tools available are platform dependent. Some tools work only on java code and some only on c# code and so on . Nonetheless they are more efficient than humans in detection of code smells. More work needs to be done on windows based GUI applications to make more developers use refactoring.Van Noije, et al., 2014 [42] He uses the term crowd smells to collectively find many code smells. It prefers collaborative enviornment for development of a product and detection of code smells in java code. It uses collective intelligence. Users from all over the world connect themselves to a cloud server to access information. It leads to accurate code without any major defects leading to a secure and robust product. PhongphanDanphitsanuphan, et al., 2012 [44] This paper propses OOP based metrics for the detection of bad smells in source code of software. Certain software metrics are used to detect bad smells. It makes use of an eclipse plug in. Many code smells were detected using the tool developed in this paper some of which are: Data class,Switch statement,Lazy Class, Large class,Long method, heirarchy of parallel inheritance. KarnamSreenu, 2012[46] This paper takes in to account 2 code smells which are Temporary Fields and Lazy Class. It describes some new methods of refactoring for the identification of smells. After identification of these bad smells we need to use refactoring methods which are most suitable for the smell. It uses Replace Temp and Merge Class methods to refactor code. These refactoring methods can be applied on the source code directly and reduce the length of code to make significant improvement in the source code. It also considers Depth of Inheritance for detection of smells.

International Journal of Engineering Applied Sciences and Technology, 2018
Vol. 3, Issue 8, ISSN No. 2455-2143, Pages 55-60
Published Online December 2018 in IJEAST (http://www.ijeast.com)

III. SIMULATION MODEL

Code Smells detection tool that we make in Microsoft Visual Studio Ultimate will be based upon the risk based strategy. The detection methodology is determined by feature selection using fact and rules and ACO algorithm.The software program made by us will look for dead code,long method,unused catch,switch and unused variable and informs the user about the detected smell on the GUI itself along with a message. It is a tool which bears a simple Graphic User Interface and easy to learn and operate. Before we start testing any source code for the code smells we need to upload the project. After the project upload is done user is notified with a message. After uploading the project user selects a code smell to be detected from the 5 code smells listed on the page. This takes the user to the specific page of that code smell. Here the user needs to upload all the files the user needs to test. Files are uploaded in sequence and after this user needs to select the "Start Testing" option. All the files are processed one by one and the smells present in them are highlighted. User can study and analyse these smells in the code and decide whether to refactor the smell or not. There are various methods for refactoring and user can select the one which is most suitable for his code and functionality. Refactoring does not have any impact on the behaviour of code. In this manner user can detect all the five code smells from his files. After all the files uploaded in the project have been tested for code smells user can check the Accuracy of detection process which has been stored under the Results tab. Results page displays the Fault Rejection Rate, Fault Acceptance Rate and Accuracy percentage. FAR,FRR and Accuracy values are stable in the project with a minor variation. Accuracy of the project is stable at 99 % whereas the FAR is stable at 0.006 and FRR is stable at 0. 004. The values of FAR and FRR represent the errors in calculations which is quite low and Accuracy of 99 % obviously indicates the results are exact and correct.

Graphic User Interface of the tool is quite simple and self explanatory. No special expertise is required to operate the tool. It is very beneficial for naïve developers who need expert guidance on coding practices. Code after completion can be uploaded to this tool for detection of smells. Code Smells are shown in a very clear manner as a list of methods in the message displayed on the screen. User can now analyse the code and refactor it if required to make it simpler, robust, secure, flexible and compact. This refactored code is easier to maintain and more scalable as compared to the longer version of it. Hence, Code Smell Detection tool has a simple and userful design and operation.

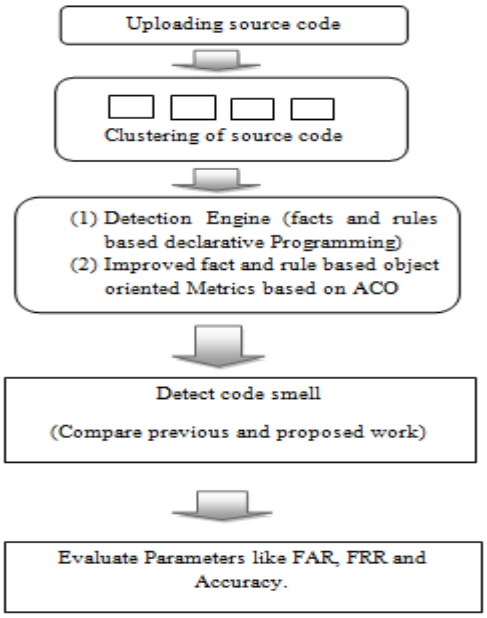

Fig.2. Flowchart of Proposed Work

IV. EXPERIMENT AND RESULT

Code Smell Detection Tool has a simple and user friendly Graphic User Interface. After launching the tool and uploading the project we are ready for detection of bad smells. We select among the 5 Bad Smells the tool detects. For Example we select the Long Method. Now the tool scans the project's code and names the methods which are longer than the specified limit. These methods are named in a message after which it is user's discretion to take appropriate action.

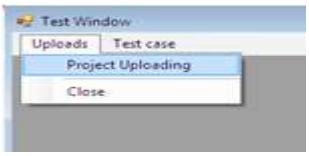

Fig.3. Project Upload Window

Fig.4. Upload Completion message

International Journal of Engineering Applied Sciences and Technology, 2018
Vol. 3, Issue 8, ISSN No. 2455-2143, Pages 55-60
Published Online December 2018 in IJEAST (http://www.ijeast.com)

After upload user selects the type of smell to be detected.For ex. we select Long Method and test the file.

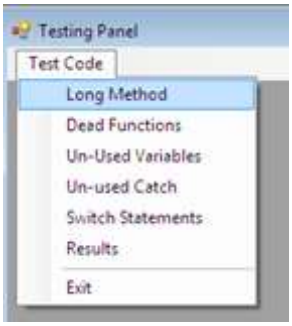

Fig.5. Selection of Code Smell

After testing the functions which are long are displayed along with message to the user.These methods can be now refactored.

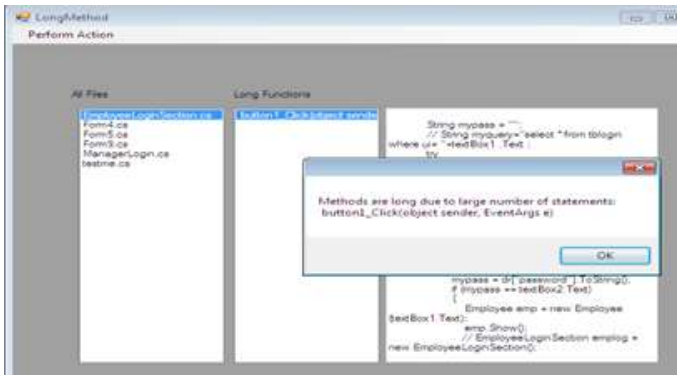

Fig.6.Detection of Long Methods

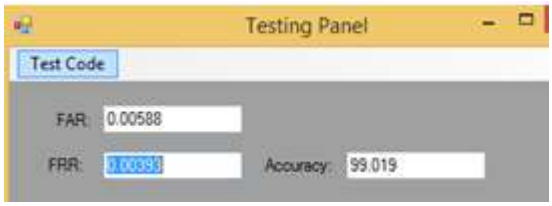

Fig.7. Result of Project

As per the execution of various files uploaded by the user for code smell detection the results are executed and displayed on the results page. Results page consists of three parameters which are FAR, FRR and Accuracy respectively. We have executed all the project files four times and shown the results obtained in the form tables and graphs.

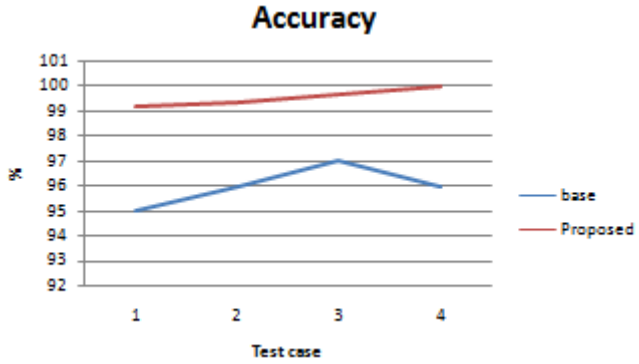

Fig.8. Accuracy

	1	2	3	4
Base	95.4	97	96	95
Proposed	99.18	99.33	99.76	99.9

Fig.9. Comparison of Proposed and Existing Accuracy Parameters

False Rejection Rate measures the likelihood that a system will reject the valid input as invalid one whereas False Acceptance Rate measures the likelihood that the system will accept the invalid input as valid one.

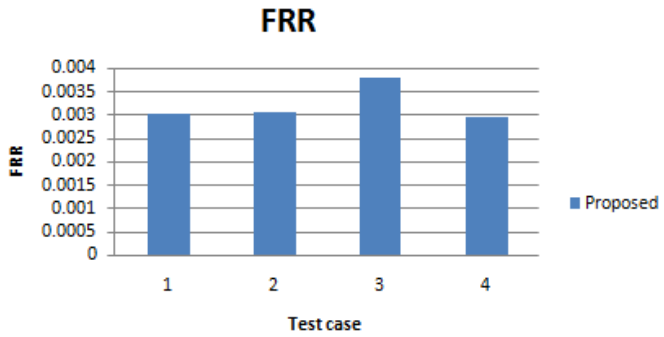

Fig.10. False Rejection Rate

The above figure shows the proposed FRR in graphical form.

	1	2	3	4
Proposed FRR	0.00309	0.00303	0.00373	0.00290

Table 1.Performance based False Rejection Rate

The above table displays the comparison of Base and Proposed FRR performance data of the tool.

International Journal of Engineering Applied Sciences and Technology, 2018
Vol. 3, Issue 8, ISSN No. 2455-2143, Pages 55-60
Published Online December 2018 in IJEAST (http://www.ijeast.com)

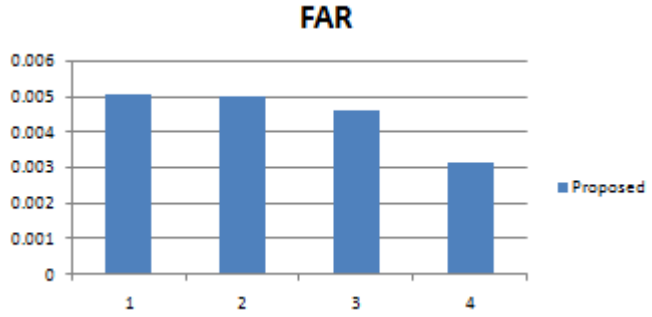

Fig. 11. False Acceptance Rate

	1	2	3	4
Proposed FAR	0.00503	0.006	0.00457	0.00317

Table 2. Performance based False Acceptance Rate

The above table displays the comparison of Base and Proposed FAR performance data of the tool. Based on these results the performance of Code Smell Detection Tool can be judged.

IV. CONCLUSION

Various techniques and tools are available in market for the study and analysis of bad smells from the software system in different languages (C, C++, Java and .net) as discussed in literature survey. Comparison of these detection tools is a complex task, and in some projects using them is not advised or necessary. Various code smells are detected in our source code using graphical user interface application. The calculated object oriented metrics show the value of each metric in their respective code smells detected in the coding. The purpose of this research work was not to evaluate the tools, but to explain our experience in using them and draw the difficulties in the comparison task. The first experiential study on the result of code smells on software conservation effort in a prohibited industrial setting. As a verification of concept, it developed an automatic risk based code smells detection tool. Our proposed approach uses optimized (Ant Colony Optimization) algorithm which is available to find various bad smells in various languages (C,C++,Java and .Net). The proposed algorithm perform better in terms of various parameters like false acceptance rate, false rejection rate and accuracy.
In the future scope, researchers can develop a refactoring approach which is able to refactor the code for various languages in our system. It can implement a hybrid approach (PSO+Firefly) and a designer based research to duplicate Mantyla's designer study and on an investigation of the difficult implication of smell suppression. The consequences accessible now will be the principle of many smell revisions and will receive additional searching in this area, to enhance the maintenance of software system and different fields. The performance can be optimized through some other optimization algorithms and refactoring of bad smells for some other parameters of code. The performance of proposed algorithm can also be enhanced through adding some other bad smells in the code.

V. REFERENCES

[1] Hazelwood, K., and Smith, M. D. ," Generational cache management of code traces in dynamic optimization systems", In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture* (p. 169). IEEE Computer Society,2003.'

[2] W. Humphrey, A discipline for software engineering. Boston [u.a.]: Addison-Wesley, 2003.

[3] Mens, Tom. "Introduction and roadmap: History and challenges of software evolution." In *Software evolution*, pp. 1-11. Springer, Berlin, Heidelberg, 2008.

[4] Fowler, Martin, and Kent Beck. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.

[5] Fontana, Francesca Arcelli, PietroBraione, and Marco Zanoni. "Automat

[6] ic detection of bad smells in code: An experimental assessment." *Journal of Object Technology* 11, no. 2 (2012): 5-1.

[7] Chatzigeorgiou, Alexander, and AnastasiosManakos. "Investigating the evolution of bad smells in object-oriented code." In *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*, pp. 106-115. IEEE, 2010.

[8] Singh, Gurpreet, and Vinay Chopra. "A study of bad smells in code." *Int J SciEmergTechnol Latest Trends* 7, no. 91 (2013): 16-20.

[9] Van Emden, Eva, and Leon Moonen. "Assuring software quality by code smell detection." In *Reverse Engineering (WCRE), 2012 19th Working Conference on. IEEE*. 2012.

[10] Counsell, Steve, Robert M. Hierons, HamzaHamza, Sue Black, and M. Durrand. "Exploring the eradication of code smells: An empirical and theoretical perspective." *Advances in Software Engineering* 2010 (2011).

[11] Mantyla, Mika V., JariVanhanen, and Casper Lassenius. "Bad smells-humans as code critics." In *Software Maintenance, 2004. Proceedings. 20th*

IEEE International Conference on, pp. 399-408. IEEE, 2004.

[12] Moha, Naouel, Yann-Gael Gueheneuc, Laurence Duchien, and Anne-Francoise Le Meur. "DECOR: A method for the specification and detection of code and design smells." *IEEE Transactions on Software Engineering* 36, no. 1 (2010): 20-36..

[13] Munro, Matthew James. "Product metrics for automatic identification of" bad smell" design problems in java source-code." In *Software Metrics, 2005. 11th IEEE International Symposium*, pp. 15-15. IEEE, 2005.

[14] Mathur, Neeraj, and Y. Raghu Reddy. "Correctness of Semantic Code Smell Detection Tools." In *QuASoQ/WAWSE/CMCE@ APSEC*, pp. 17-22. 2015.

[15] Fokaefs, Marios, NikolaosTsantalis, and Alexander Chatzigeorgiou. "Jdeodorant: Identification and removal of feature envy bad smells." In *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, pp. 519-520. IEEE, 2007.