International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-8 Issue-12, October 2019

Restructuring with Moora and Measuring Code Smells

T.Pandiyavathi, T.Manochandar

Abstract*: The paper presents measuring various code smells by finding critical code smells and thereby concentration is increased in those parts through Structural Modeling for arranging those code smells. Arranging the code smells in the way that they will not produce a new smell on their detection and removal is very necessary. Structural modeling helps in clarifying Interrelationship among these code smells. The code smells that contains high driving effects are ordered as optimized code which resulted in the increase in the overall code maintenance of the software code which will be used afterwards for achieving the concept of re-usability. In addition to this we have added a technique for restructuring technology for the purpose to achieve high accuracy. It involves more objectives related to the performance and the code smells are implemented with the concept called as pairwise analysis based on the priority method. Pairwise analysis based on the weights attained by the bad smells provides a better optimized results since more problematic areas are neglected here. This work gives optimized results for the process of overall code maintainability by applying restructuring before the refactoring process with Fuzzy technique and it is followed by finding the code smells which results in high ripple effects and then removing them. Still more research ideologies are needed for removing the bad smells in the code.

Keywords* : *MOORA, FODA, Fuzzy

I. INTRODUCTION

Code smells are the flaws in the design that shortens the process of maintenance and reusability. It is important for detecting all the code smells before the end delivery of the product. Moreover the design and coding part should be taken care of in order to yield an efficient code for further usage and maintainability.

Refactoring is a technique to keep the code cleaner, simpler, extendable, reusable and maintainable to achieve the quality of the code. It changes the program code to improve its internal structure, understandability and other features without disturbing its external behavior. This paper reveals the effective usage of FODA Rule set and its implementations establish methods for performing Restructuring with a Fuzzy methodology.

Revised Manuscript Received on October 30, 2019.
***** Correspondence Author
T.Pandiyavathi*, Assistant Professor/Department of Computer Application, B.S.Abdur Rahman Crescent Institute of Science and Technology, Chennai, pandiyavathi@crescent.education
T.Manochandar, Assistant Professor/ Department of Electronics and Communication Engineering, CK College of Engineering and Technology, Cuddalore. manochandar.be@gmail.com

Refactoring and Reuse of software is one of the most promising solutions so it is called “software crisis”. They are used to construct different feasible configurations of refactoring, reusable architectures.

Domain Analysis is defined as the process of identifying, collecting, organizing and representing the relevant information. That is to meet a customer and collect the correct requirements of customer need. If any changes of customer need, the developer is to be modified.

MOTIVATION

Refactoring a code, results in introduction of new code smells. Thereby conflicting criteria should be identified through measuring and selecting most suitable data distribution strategy using an integrated approach of fuzzy based multi-objective on the calculated values based on ratio analysis.

IDEA

Structural modeling that identifies interrelationship between code smells using MOORA-Multi objective optimization (In Restructuring The Code) process has been introduced which is further followed by refactoring of the modeled code with FODA TOOL which is a Feature Oriented Domain Analysis [FODA], a Rule Engine and also called FODA utility tool used for implementing Refactoring the code and then further reusing them.

MOORA

(MOORA) is a technique to optimize two or more conflicting attributes. This multi objective approach is actually an improvement to the already existing traditional techniques based on the cost and performance measures, where the units of all the costs and benefits were required to be same. In MOORA, various attributes or criteria can have different units. The MOORA method was developed by Brauers and Zavadskas (2006), and the method is implemented in a tool for selection of best methods in the field of engineering and maintenance measures. The MOORA technique considers the priorities of the conflicting criteria, which were calculated using the analytic hierarchy process (AHP). The priority weights of the criteria were determined using pairwise comparison that was based on expert opinion.

NEED OF REFACTORING

Refactoring improve code quality, reliability and maintainability throughout an all software lifecycle models.

Refactoring improves the design of a software code.

Refactoring makes software easy to understand.

Refactoring helps find the errors Refactoring helps to program run faster. Software Reusability

II. LITERATURE REVIEW

Gupta et al. (2016) in their research paper presented a framework to measure the code smells in a real time environment where two methods are incorporated namely TISM and Two-way assessment where pairwise comparisons between the code smells are done with the concept of AHP and priority weights are determined efficiently using which code smell with maximum weight can be resolved. Reddy and Reddy (2012) proposed a very promising traditional approach where they used a cloud environment for achieving data distribution strategies, which results in high availability of data and thereby provides better reliable data with better economic values. Compared with all these methods, various other methods are also available for fragmenting the data where many authors provided idea for cloud computing users with flexible decision models.

MOORA is used where the methodology of data distribution is used to select the better and related data distribution strategy. Zhang et al. [4] done literature survey to study bad smells in the software code. He investigated more then 300 papers from 2000 to 2009 and different parameters are considered for this investigation. Results are analyzed based on different perspectives.

In the discipline of refactoring and detection of code smells authors like, Al Dallal[2] , Misbhauddin[3] and Alshayeb , Zhang et al. [4] have reached a milestone in achieving certain relationships between bad smells and existence of antipatters in the software code.

Kundakci 2016; Mazilu 2010 deals with the distributed environment and discussed on partitioning or simply fragmenting the data in large enterprises and documented the methods that they have discussed using different strategical approaches and tried implementing in different sources and levels of technical and business fields and also the problems faced by the data admins and business architects.

Making extensive feasibility analysis and study on data distribution and fragmentation strategies, different authors remarked that, few only done a particular amount of work in the field. The research community commonly used Multi objective model and AHP frameworks.

Rababaah and Hakimzadeh (2005) introduced the concepts and issues that are highlighted with distributed databases. Ozsu and Valduriez (1991) discussed on the issues of transaction models, impact of problems related to technical issues in distributed data environment that needs more research works to be done an implemented in those environments. The major problems are thereby thrown into picture like query processing with distributed data, analyzing the duplication of data and the pitfalls faced by them.

Various levels of literature has been done and studied in the field of distributed database and they were evaluated for doing all the goals required in their enterprise software or business models. Various related concepts has been put into effect and checked where all the commercial products are in the compulsion of achieving their own promises down during their requirement phases.

Apart from the above mentioned systematic literature surveys, the work done in last few years on the data distribution is very much remarkable. They have boosted the belief of handling the proposed work and doing the further extensions by the research community people. Al Dallal [2] has used in his literature work a remarkable amount of data sets which is high when compared to others. Still more data and information is needed in the literature study of bad smells detection and correction.

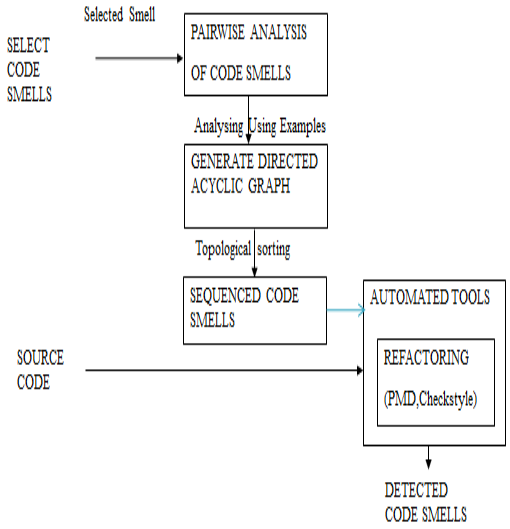
III. MODULES FOR RESTRUCURING

The various modules of the paper includes,

1. Platform Dependent Software
2. Impersonation Of Data In Software
3. Data Replication
4. Selection Of Master
5. Metrics Tool Usage

1. PLATFORM DEPENDENT SOFTWARE

Platform dependent database is a concept where the software resides in a single Operating system. As of discussed before any changes done in any part of the code will definitely affect other parts of the system. Thereby by using paramounted concept, changes to the software code affects in other part of the system. This positive approach of centralized software is better maintainability, processing, consistency, clarity, usability, reusability and traceability. Main measure of security and integrity also will be achieved through this approach. Although the pitfalls includes, lack of authorization to the customers by the developer. Sometimes integrity of the code cannot be attainted. This results in bad recovery measures.



2. IMPERSONATION OF DATA IN SOFTWARE

It is a fault tolerant technique which is used to initiate and also select and maintain copies of software code within different sets of tools available in usage.

In this step of our approach, two types of methodology is implemented. They are master and slave. Some ways of code, where the contents of the master to be saved to the slave should be used. Everything is related with the time constraint where based on the type of copying method, namely,

1. Synchronized replication
2. Non-Synchronized replication

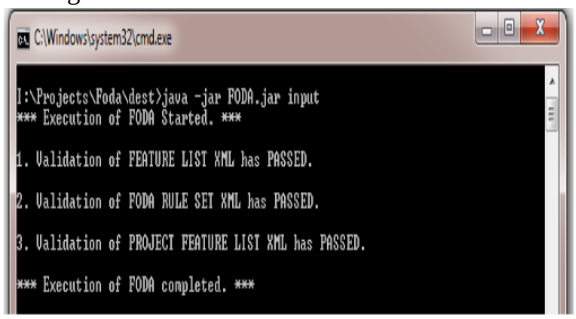
In the first method of code replication, soon after the update is done by the master, it is to be copied to the slave. This process should be done exactly after each and every update.
In the method of synchronized replication, data consistency is achieved in higher precision levels, and improves the efficiency of the system.

3. DATA REPLICATION

This step of better code smell detection and removal will automatically results in maintaining the high level of data consistency of data when applied to any kind of real time applications. Data replication may results in high storage needs however, data replication results in decreased level of loss of data and it also enhances the level of communication since we have master and a slave. In case of loss of data in master, data from slave can be retrieved through proper channel of communication. Cost of communication is also an important measure to be taken care of. Finally, overall cost is another important criterion that is a major factor in a distributed enterprise. It includes all the hardware and software costs. These criteria's are convicting as data architects targets to maximize reliability, expandability, manageability and data consistency while reducing the communication overhead and the overall costs.

4. SELECTION OF MASTER

From the coding that was categorized, the optimized master should be selected based on the Structural modeling tool. It should be build first and then the code should be checked. The optimized code should be further taken to the process of refactoring

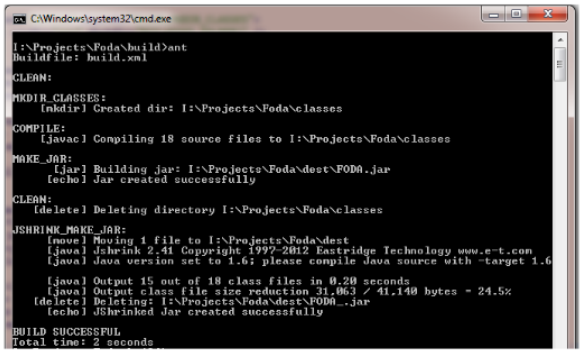


FODA EXECUTION
FODA BUILD

5. METRICS TOOL USAGE

As we know that the refactoring is processed by the FODA tool, Feature based technique has been used in there to select and change the code structures to yield the best software code with minimal amount of errors without changing the functional behavior. Finally metrics tool will be used to check the accuracy matrix and compared against the existing algorithms.

IV. MEASURES, OUTCOMES



This section illustrates the outcome of the methodology in a open source software code.
As the first step, performance measures are evaluated based on particular or certain criteria's based on the nature of the method used. Most importantly the reliability measures are calculated based on data consistency, manageability and cost factors. A matrix is created based on the above criteria's and finally a resultant values are found and checked for accuracy. In general, the FODA provides a detailed overview of the problem solved by software in a given domain. The complexity of even a well understood application, such as that of real time software systems, establishes the need for tools to handle them and variety of information the tool can generate.

1. Precision

Precision is defined as the amount of documents retrieved that are relevant

P (relevant|retrieved)

Therefore, P = tp/(tp + fp)

And based on the matrix generated and the values evaluated based on the given criteria the value of precision is,

P=66.50

2. Recall:

It is defined as the fraction of relevant documents retrieved.

R (retrieved|relevant)

R = tp/(tp + fn)

Here the value of recall is,

R=75.71

3.Accuracy:

Accuracy and f-score is calculated using the values of precision and recall.

A=(tp+tn)/(tp+tn+fp+fn) = 74.428

F-score= 2*(P.R)/(P+R) = 80.1

Below given is the tabular column for the f-scored attained based on the pairwise analysis done within the code smells.

Prioritized bad smell	F-score
S1	77
S2	78.4
S3	80
S4	81.3
S5	79.3
S6	78
S7	79
S8	77
S9	80
S10	80

V. CONCLUSION

This paper thus provides an insight into the restructuring concepts with the intrusion of the FODA tool support. Multi-objective approach along with the pairwise analysis of the available code smells proves that the resultant of the approach gives better results than the other existing methodologies discussed. However we develop these approaches for optimizing the restructuring algorithms, human effort is needed for the purpose of achieving better results. This is due to the fact that monitoring and maintaining the code is done with great efficiency by the human personnel comparatively with the software or other tools used.

VI. IMPROVEMENTS AND FURTHER STUDY

MIC MAC can been used later for(for identifying code smells having high driving power and dependency power for furthermore improvements in software refactoring and reusability of the software code. The FODA method must also extended to provide automatic support for the application to support the user decisions. Applying the method in new application will support the separation of the method and give validation to the approach.

REFERENCES

1. Martin Fowler, Kent Beck (Contributor), John Brant (Contributor), William Opdyke, don Roberts(2002), "Refactoring: Improving the Design of Existing Code".
2. Al Dallal Jehad. Identifying refactoring opportunities in object-oriented code: a systematic literature review. Inf Softw Technol 2015;58:231–49.
3. Misbhauddin Mohammed, Alshayeb Mohammad. UML model refactoring: a systematic literature review. Empirical Softw Eng 2015;20(1):206–51.,Springer.
4. Zhang Min, Hall Tracy, Baddoo Nathan. Code bad smells: a eview of current knowledge. J Softw Maint Evol Res Pract 2011;23:179–202.
5. Ali ouni, MarouaneKessentini, HouariSahraoui, Mohamed Salah Hamdi, "The Use of Development History in Software Refactoring Using a Multi-Objective Evolutionary Algorithm", proc. GECCO'13 pages 1461-1468 and ACM 2013.
6. Hui Liu, XueGuo and Weizhong Shao, "Monitor-Based Instant Software Refactoring", IEEE Transactions of Software Engineering 2013.
7. Hui Liu, ZhendongNiu, Zhiyi Ma, Weizhong Shao, "Identification of generalization refactoring opportunities", Automated Software Engineering: Volume 20, Issue 1 (2013), Page 81-110
8. A. Lozano, M. Wermelinger, and B. Nuseibeh, "Assessing the Impact of Bad Smells Using Historical Information," Proc. Ninth Int'l Workshop Principles of Software Evolution: In Conjunction with the Sixth ESEC/FSE Joint Meeting, pp. 31-34, 2007.
9. E. van Emden and L. Moonen, "Java Quality Assurance by Detecting Code Smells," Proc. Ninth Working Conf. Reverse Eng., pp. 97-106, 2002.
10. N. Tsantalis, T. Chaikalis, and A. Chatzigeorgiou, "Jdeodorant: Identification and Removal of Type-Checking Bad Smells," Proc. 12th European Conf. Software Maintenance and Reeng., pp. 329-331, Apr. 2008.
11. M. Fokaefs, N. Tsantalis, and A. Chatzigeorgiou, "Jdeodorant: Identification and Removal of Feature Envy Bad Smells," Proc. IEEE Int'l Conf. Software Maintenance, pp. 519-520, Oct. 2007.
12. M. Mantyla, J. Vanhanen, and C. Lassenius, "Bad Smells— Humans as Code Critics," Proc. IEEE 20th Int'l Conf. Software Maintenance, pp. 399-408, Sept. 2004.
13. G. Meszaros, xUnit Test Patterns: Refactoring Test Code. Addison-Wesley, 2007.
14. M. Mantyla, J. Vanhanen, and C. Lassenius, "A Taxonomy and an Initial Empirical Study of Bad Smells in Code," Proc. Int'l Conf. Software Maintenance, pp. 381-384, 2003.
15. B. Pietrzak and B. Walter, "Leveraging Code Smell Detection with Inter-Smell Relations," Proc. Seventh Int'l Conf. Extreme Programming and Agile Processes in Software Eng., pp. 75-84, June 2007.
16. T. Mens, G. Taentzer, and O. Runge, "Analysing Refactoring Dependencies Using Graph Transformation," Software and Systems Modeling, vol. 6, no. 3, pp. 269-285, Sept. 2009.
17. H. Liu, L. Yang, Z. Niu, Z. Ma, and W. Shao, "Facilitating Software Refactoring with Appropriate Resolution Order of Bad Smells," Proc. Seventh Joint Meeting of the European Software Eng. Conf. and the ACM SIGSOFT Symp. the Foundations of Software Eng., pp. 265- 268, 2009.
18. Core Java vol-1 Fundamenals,Eighth edition by Cay S.Horstmann and Gary Cornell
19. Hui Liu, Zhiyi Ma, Weizhong Shao and ZhendongNiu, "Schedule of Bad Smell Detection and Resolution: A New Way to Save Effort", IEEE Transactions on Software Engineering, vol.38, no.1, pp.220-235, Jan.-Feb. 2012
20. http://www.myreaders.info/html/artificial_intelligence.html
21. Integrating Code Smells" Detection with Refactoring Tool, a report submitted by KwankamolNongpon August 2012
22. Tools - http://www.Sourceforgenet.com- pmd, checkstyle
23. Automated refactoring: a step towards enhancing the comprehensibility of legacy software systems by Isaac D. Griffith

AUTHORS PROFILE

T. Pandiyavathi,completed her B.E and M.E in software engineering from Anna University, Chennai. She has three years of teaching experience and published papers on refactoring in scopus journals and been a member in ISTE, IAENG and IACSIT. Her research work covers the topics of software engineering, testing and genetic algorithm.

Manochandar Thenralmanoharan,was born in Tamilnadu, India in 1988. He received B.E. in Electronics and Communication Engineering from Kamban Engineering College (A constituent college of Anna University, Chennai) in 2009 and M.E. in Communication Systems from Prathyusha Institute of Technology and Management, Tiruvallur (A constituent college of Anna University, Chennai)in 2012.He has seven years experience in the teaching field. He is interested in the fields of Wireless Communication, Neural Networks, Image Processing and Embedded Systems. He is Assistant Professor of Electronics and Communication Engineering in CK College of Engineering and Technology, Cuddalore (A constituent college of Anna University, Chennai). He has been a consistent active member of ISTE, IAENG, IACSIT and acted as a reviewer in many journals