Object Oriented Design Refactoring for Enhancing the Technical Debt

M. Vidhya, G. Zayaraz

Abstract: ***Code Refactoring is a process of changing the internal behavior without changing its external behavior or functionality. Manual refactoring is hard to modify changes, if we automate refactoring there are much more benefits possible. Software refactoring is a valuable process in software development and is often aimed at repaying technical debt. The automated refactoring techniques, software metrics and Metaheuristic Search and automated refactoring tool are combined to improve the quality of software without affecting its functionality. The four different refactoring approaches are compared using automated refactoring tool. The more number of metrics are added to improve the quality and reduce the complexity. Metrics are combined to measure Abstraction, coupling, inheritance and technical debt. This will improve the quality of software and also reduces technical debt by maintenance cost and time.***

Keywords: Search based techniques; refactoring; Software metrics; software quality; design level metric; Technical debt.

I. INTRODUCTION

In software engineering most of the problems are solved using search based software engineering (SBSE) techniques. Here it is based on source code refactoring, so it is considered as white-box problem. This SBSE is used for finding the nearest or optimal solution. Further, the Metric also called fitness function or quality measure is used to measure the quality of software and their potential solutions.

Metrics are used both implicitly and explicitly to measure and assess software [1]. Increasingly, object-oriented measurements are being used to evaluate and predict the quality of software. A growing body of empirical results supports the theoretical validity of these metrics. The validation of these metrics requires convincingly demonstrating that (1) the metric measures what it purports to measure (for example, a coupling metric really measures coupling) and (2) the metric is associated with an important external metric, such as reliability, maintainability and fault-proneness. The fitness function that guides the search is based on one or more software metrics.

The search based software maintenance (SBSM) is used to solve the problem easily. Then the automated refactoring is applied without the programmers need concentrating on the time basis. The technical debt (TD) is a metaphor, which is used to reduce the complexity. Technical debt complicates the decision making. If the TD is identified, measured and monitored then the manager can have informed decision and have higher quality maintenance. The TD can be regularly repayed in order to minimize the risk, meaning refactoring. The quality attributes like abstraction, inheritance, coupling and technical debt all give the weighted sum with fitness factors [2]. The important concepts of Object Oriented are inheritance (reusability), coupling (message passing, complexity), Cohesion (Maintainability, complexity) [3].

The individual quality attributes that are taken for refactoring are Technical Debt – A measure of reducing the code complexity by refactoring the classes, methods and fields.

Coupling - A measure of the dependencies between classes based on counts of usage of class, attributes and parameters by other classes.

Inheritance - A measure of the class structure of a project in terms of counts of interface implementations and of descendants and ancestors.

Abstraction - How easy it is for a software system to be extended and built upon. Estimated based on number of abstract classes present and the number of interfaces present and implemented.

II. RELATED WORK

Code refactoring is a type of cleanup can have a profound, positive effect on the maintainability, reusability and understanding of code. I don't have to test after the automated refactoring. The input for refactoring is java program as byte code. The individual quality attributes are calculated with the metric measurement as shown in table 1. There are various level of refactoring available like class level refactoring, method level refactoring and field level refactoring of that each concentrate on their own security characteristics. The refactoring patterns are composing methods, moving features between objects, organizing data, simplifying conditional expressions, making methods calls simpler, and dealing with generalization [4]. Extract method is an important refactoring. Quality Model for Object Oriented Design (QMOOD) has six metrics they are taken to calculate the value of fitness function and then for comparing the refactor metric value and fitness function and display that how much technical debt is reduced. The six metrics are Method Hiding Factor (MHF), Attribute Hiding Factor (AHF), Weight Method Per Class (WMPC), Response For Class (RFC), Lack of Cohesion of Methods (LCOM), Attribute Inheritance Factor (AIF). The six open source programs are taken to evaluate the software quality factors.

Revised Version Manuscript Received on May 15, 2017.

M. Vidhya, Department of Computer Science and Engineering, Pondicherry Engineering College, Pillaichavadi (Puducherry)-605014, India. E-mail: vidhya93m@gmail.com

Dr. G. Zayaraz, Professor, Department of Computer Science & Engineering, Pondicherry Engineering College, Pillaichavadi (Puducherry)-605014, India. E-mail: gzayaraz@pec.edu

Table 1: software metric weights with ACMA

Software Factor	Metric components and weights
Technical Debt	-0.1 * numField - 0.1 * avrgField - 0.1 * numOps - 0.06 * nesting + 0.1 * abstractness + 0.1 * numCls + 0.1 * numInterf + 0.1 * iFImpl + 0.06 * NOC + 0.06 * numDesc - 0.06 * Dep_In -0.06 * Dep_Out
Coupling	-0.125 * iC_Attr - 0.125 * eC_Attr - 0.125 * iC_Par - 0.125 * eC_Par - 0.25 * Dep_In - 0.25 * Dep_Out
Inheritance	0.25 * iFImpl + 0.25 * NOC + 0.25 * numDesc + 0.25 * numAnc
Abstraction	0.33 * abstractness + 0.33 * numInterf + 0.33 * iFImpl

A. Refactoring Tool

For QMOOD metrics refactoring they use a tool called Automotive Component Manufacturers Association of India (ACMA) Tool. QMOOD is the hierarchical model that defines relation between quality attributes and design properties with the help of equations [5]. It has their own feature to load the design and refactor automatically. It will refactor according to the parameters set in the java program and with certain conditions and rules like if condition, switch case and looping etc.

III. PROPOSED WORK

In the proposed system design, Chidamber and Kemerer metrics(C & K) is added in the existing work to enhance the software quality and reduce the TD. CK metric were designed to measure the unique aspects of OO approach, to measure complexity of the design, to improve the development of software. The design inspector determines whichever to exclude/include the inherited attributes and methods [6]. This is a design level metric. Design metrics are useful in measuring the complexity and integrity of a design. A large number of metrics have been proposed for Object oriented design. Some of these are validated, some little or no validated. The CK metric also called **M**aintain and **O**perate the IT **O**rganization, **S**ystems, and **E**quipment (MOOSE) metrics. The CK metric calculation has been viewed as straight forward and non-controversial. CK aim is to reuse. This metric compares the design and predict which is better. Significance of CK metrics to predict the number of faults. CK is a predicted commercial approach.

Here we have taken the six java programs like jhotdraw, beaver, apachexmlrpc, jflex, json and mango. They have the default classes in it. The CK has six metrics like Weighted Method Per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling between Objects (CBO), Response for Class (RFC), Lack of Cohesion in Methods (LOCOM), Afferent Coupling (Ca), Number of Public Methods of a Class (NPM). MOOD and CK are the two set of metrics that cover every aspect of Object Oriented Paradigm [7]. These CK metrics are added with the QMOOD metrics for maintainability and development effort. It gives the measure of the metric values that how much it is good and bad in case of refactoring the code. The individual quality factors is calculated with metric components and weights as shown in table 2.After refactoring, the metrics and the fitness function value is measured for improving the technical debt further and also for the software quality. For refactoring and reduction of TD using CK metric a tool is used called as Chidamber and Kemerer Java metrics (CKJM). It will calculate the TD with the classes of open source programs and measure for automate the refactoring.

Table 2: software metric weights with CKJM

Software Factor	Metric components and weights
Technical Debt	0.6 * NOC + 0.1 * WMC – 0.06 * NPM
Coupling	-0.125 * CBO – 0.125 * RFC
Inheritance	0.25 * DIT + 0.25 * NOC
Abstraction	0.01 * LCOM + 0.03 * CA

The developer takes java programming as input and their constraints are methods, fields and class. They find the possibility of refactoring the code. That will be measured using QMOOD and CK metrics value. Once it found the security issues in the code, it will refactor with the automated tool and shows the fitness function value and their relevant metric value. The fitness value of QMOOD and the fitness values of CK are compared against each other that software quality is enhanced and reduced the technical effort and maintainability and reuse of code or not. The below figure 1 explains clearly about the existing and proposed work.

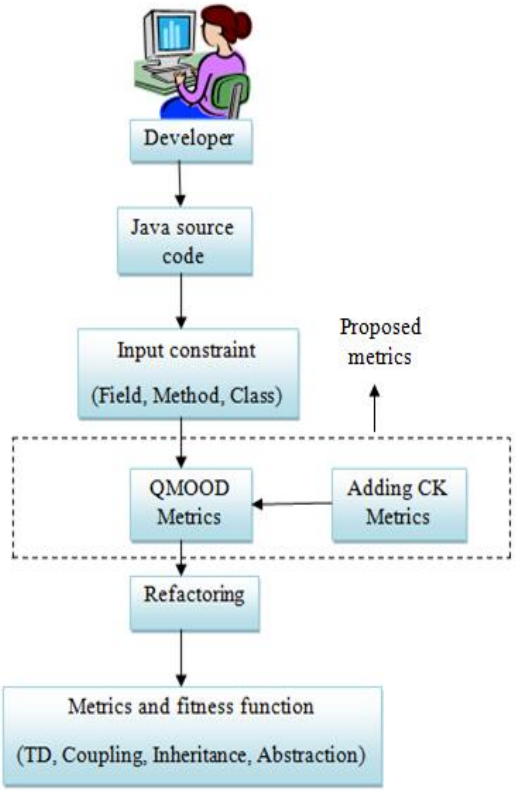

Fig 1:Architecture of Proposed System

Each metric is measured for specific reason one is for identify the complexity another is for counting the methods and fields,and some for calculating the hiding attributes or methods.That will be discussed neatly in below table 3.

Table 3: Metrics definition

Metrics	Description
Method Hidding Factor(MHF)	Ratio of the sum of the invisibilities of all methods defined in all classes to the total number of methods defined in the system under consideration.
Attribute Hidding Factor(AHF)	Ratio of the sum of the invisibilities of all attributes defined in all classes to the total number of attributes defined in the system under consideration.
Weighted Method per Class(WMC)	Sum of complexity of the methods in a class
Response for Class(RFC)	Defined as set of methods that can be executed in response and messages received a message by the object of that class.
Lack of Cohesion of Methods(LCOM)	Count the number of disjoints methods pairs minus the number of similar method pairs used.
Attribute Inheritance Factor(AIF)	Ratio of the sum of invisibilities of all attributes defined in all classes divided by the Total number of attributes defined in the project.
Depth of Inheritance Tree(DIT)	To find the length of the maximum path from the root node to the end node of the tree
Number of Children(NOC)	Immediate sub class coordinated by the class in the form of class hierarchy
Coupling between Objects(CBO)	To count the number of the class to which the specific class is coupled.
Afferent Coupling(Ca)	A class's afferent coupling is a measure of how many other classes use the specific class.
Number of Public Methods of a Class(NPM)	The NPM metric simply counts all the methods in a class that are declared as public.

These characteristics decides the quality factors and validate which increases and which decreases if we add new more metrics for automated refactoring.

IV. PERFORMANCE EVALUATION

The metric value describes the way that it is pursuing in the refactoring process. Refactorings, behavior preserving transformations, are claimed to make software easier to understand and to improve software design [8]. The value of NOM and WMC are similar as method complexity considered to be unity. WMC measures complexity of individual class. In WMC smaller number of methods is good for usability and reusability. Large number of methods, limiting the possibility of reuse. A lower WMC indicates to a class with better abstraction and polymorphism. High complexity is harder to test, reuse and maintain. This WMC is a good indicator of how much effort will be necessary to maintain and develop a particular class. If WMC value is 1 to 50 good. More than 50 are bad.

NOM is a subset of RFC and is simple to measure. The value of AIF is 0.25 suggesting low use of attribute inheritance. The MIF value is 0.54. It is observed that a very less methods in super classes, they contain only abstract methods that are overridden in subclasses. The MHF and AHF values are same. So all methods and attributes are private in this program. The MHF and AHF measures the variables and methods encapsulated in class. Their visibility is counted. Private method and attribute are fully hidden. A low MHF indicate insufficient abstract implementation. MHF methods are unprotected and probability of error is high. A high MHF indicate little functionality. Encapsulation measures are contained in mood suite. MHF method hiding increases reusability. MHF method hiding decreases complexity. MHF reduces modifications to the code.

A strong coupling between classes in an Object Oriented design can also increase the complexity of the system by introducing multiple kinds of inter-dependencies among the classes [9]. CBO and LCOM are coupling based metrics. If its value positive means original version performs better. If its value negative means refactored version performs better. Information hiding gives rise to encapsulation in object oriented languages. Large programs that use Information hiding have been found to be easier to modify. The CK metric suite examines LOCOM. High cohesion means good class. Low cohesion means increase complexity. Inheritance decrease complexity, but abstraction of objects can build safeguarding and design difficult. Coupling and cohesion -Code that exhibits low coupling and high cohesion is typically easier to read, understand and test. There are code analysis tool that can report the amount of coupling and cohesion in a given system. Couplings due to the use of the inheritance, because a class is greatly coupled to its ancestors by way of inheritance [10]. CBO value should be as low as possible. Increased coupling increases interclass dependencies, and working the code less suitable for reuse. More coupling means code is difficult to maintain. More classes link, more complex, difficult and test. If LCOM values is 0, no methods. 1, cohesive component. 2 or more, lack of cohesion.

Afferent Coupling (Ca) is that If I = 1 , unstable, possible to easy change in package. If I =0 , more difficult to modify due to greater responsibility. If it has value 0.0 has maximal stable package(0.0 to 0.3) and 1.0 has maximal unstable package(0.7 to 1.0).

Number of Public Methods (NPM) For this the Lower limit is 1 , it has single class. If it has upper limit class is harder to define, because it depends on the type of class, is it a simple DTO(data transfer object) with several getter and setter methods.

The original java code metric and refactored metric value are compared and then it showing the highest possibility of factoring. That is shown in below figure 2.

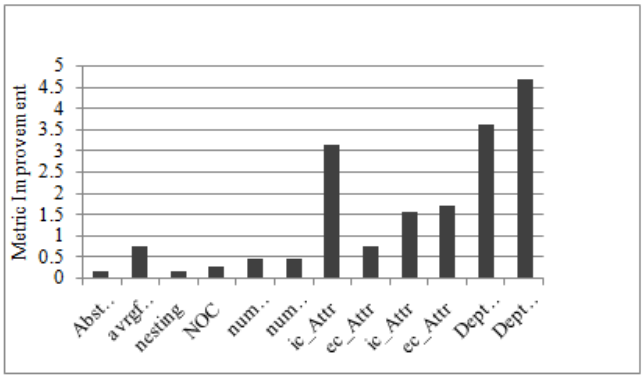

Fig 2: Mean quality gain for each metric

Once the refactoring is done with open source programs and fitness function. With their predicted values the graph is made and it clearly explains which attribute is increased and which attribute is decreased. While comparing to all attributes Technical debt shows greater improvement in reducing the effort of Developing and reducing maintenance cost and time. This is given in below figure 3.

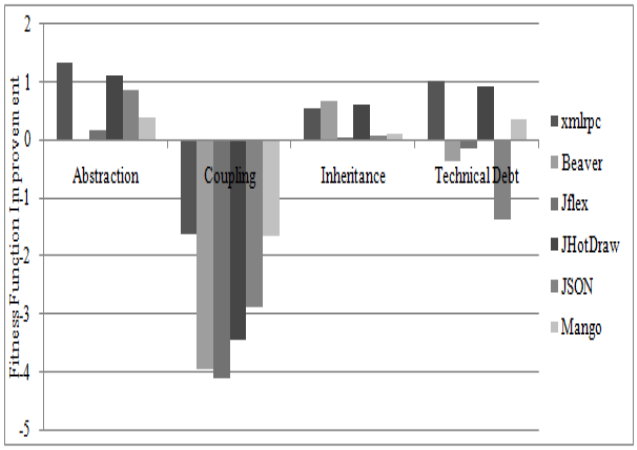

Fig 3: Fitness function of each program with quality attributes

A sample java programs taken as input and corresponding metric values for each program is defined. The value ranges from 0 to infinity, but Nan describes that its not a number because the input has no defined constant values or variables or it cannot find the given methods or variables. That values are given below in table 4. The metric calculation is done for both source code and detailed design level [11].

Table 4: Metric class value

JAVA CLASS	METRICS					
	MHF	AHF	WMPC	RPC	LOCOM	AIF
Apriori.java	0.11	0.2	5.3	5	1	0.0
Math.java	∞	1.0	1	1	0	Nan
Preprocess.java	0	0	0	0	0	0
Tempcode1.java	∞	0.5	1	2	0	Nan
Test.java	∞	0.33	2	3	0	Nan
Test1.java	∞	0.33	2	3	0	Nan
Type2.java	∞	1.0	1	1	0	Nan
Type2a.java	0.5	0.5	1	2	1	0.0

The open source program are automatically refactored with the refactoring tools and the fitness function are calculate with metric values. These values are given below in table 5.

Table 5: Refactored metrics value

OPEN SOURCE PROGRAMS	FITNESS FUNCTIONS			
	TECHNICAL DEBT	COUPLING	INHERITANCE	ABSTRACTION
ApachexmlRpc	1.02	-1.63	0.54	1.34
Beaver	-0.35	-3.96	0.68	0.02
JFlex	-0.13	-4.13	0.05	0.18
Jhotdraw	0.92	-3.47	0.62	1.11
Json	-1.36	-2.89	0.08	0.88
Mango	0.37	-1.66	0.11	0.40

Effort in developing a class is determined by the number of methods. Overall complexity of a class can be measured as a function of the complexity of its method.

V. CONCLUSION

The simulated annealing performs the better search when QMOOD metric is used for reducing technical debt. But this approach cannot fully improve the software quality and automate the process. Adding more metrics to automate refactoring, that is CK metric and martin metric are used. Using fitness function to TD gives a better improvement in metric values compared against fitness function that only aim to measure specific factors of software like coupling , inheritance, Technical debt and abstraction. This automated approach could ease the costly maintenance, saving time and effort for the developer and reduces the technical debt.

REFERENCES

1. Mel Ó Cinnéide, Laurence Tratt , Mark Harman, Steve Counsell , Iman Hemati Moghadam. Experimental Assessment of Software Metrics Using Automated Refactoring. ESEM'12, September 19–20, 2012, Lund, Sweden Copyright 2012 ACM 978-1-4503-1056-7/12/09.
2. Michael Mohan , Des Greer , Paul McMullan. Technical debt reduction using a search based automated refactoring, The Journal of Systems and Software 0 0 0 (2016) 1–12.
3. Gomathi. S and Edith Linda. P. An overview of Object Oriented Metrics A complete Survey. International Journal of Computer Science & Engineering Technology (IJCSET). Vol. 4 No. 09 Sep 2013. ISSN : 2229-3345.
4. Muktamyee Sarker. An overview of Object Oriented Design Metrics. Master Thesis Department of Computer Science, Umeå University, Sweden June 23, 2005.
5. Sonia Chawla. Review of MOOD and QMOOD metric sets. International Journal of Advanced Research in Computer Science and Software Engineering. Volume 3, Issue 3, March 2013 ISSN: 2277 128X .
6. Safwat M. Ibrahim, Sameh A. Salem, Manal A. Ismail, and Mohamed Eladawy. Identification of Nominated Classes for Software Refactoring Using Object-Oriented Cohesion Metrics. IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 2, No 2, March 2012. ISSN (Online): 1694-0814. www.IJCSI.org
7. Amandeep Kaur, Satwinder Singh, Dr. K. S. Kahlon and Dr. Parvinder S. Sandhu. Empirical Analysis of CK & MOOD Metric Suit. International Journal of Innovation, Management and Technology, Vol. 1, No. 5, December 2010. ISSN: 2010-0248.
8. Gauri Khurana and Sonika Jindal. A model to compare the degree of Refactoring opportunities of three Projects using a machine algorithm . Advanced Computing: An International Journal (ACIJ), Vol.4, No.3, May 2013. DOI : 10.5121/acij.2013.4302 17.
9. Ramanath Subramanyam and M.S. Krishnan. Empirical Analysis of CK Metrics for Object-Oriented Design Complexity:Implications for Software Defects. IEEE Transactions On Software Engineering, VOL. 29, NO. 4, APRIL 2003.

10. D.I. George Amalarethinam and P.H. Maitheen Shahul Hameed. Analysis of Object Oriented Metrics on a Java Application. International Journal of Computer Applications (0975 – 8887) Volume 123 – No.1, August 2015.
11. Elvira Maria Arvanitou, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Paris Avgeriou.,2015. Software metrics fluctuation: a property for assisting the metric selection process, Information and Software Technology 72 (2016) 110–124.

AUTHORS PROFILE

M.Vidhya, is currently pursuing master's degree programme in Computer Science and Engineering in the stream of Information Security in Pondicherry Engineering College, Pondicherry, India. She received her Bachelor`s, degree in Computer Science & Engineering from Pondicherry University. She is doing research on Secure Software Engineering. She can be reached by email at vidhya93m@gmail.com

Dr. G. Zayaraz, is currently working as Professor in Computer Science & Engineering Department at Pondicherry Engineering College, Puducherry, India. He received his Bachelor`s, Master's and Doctorate degree in Computer Science & Engineering from Pondicherry University. He has published more than twenty five research papers in reputed International Journals and Conferences. His areas of specialization include Software Architecture and Information Security. He is a reviewer for several reputed International Journals and Conferences and Life Member of CSE, and ISTE. He can be reached by email at gzayaraz@pec.edu