IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-6, Issue-9, Sept. 2018 E-ISSN: 2347-2693

An Advanced Coupling Complexity Metric for Evaluating the Quality of OO Software Modules

N. Vijayaraj[1], T.N. Ravi[2*]

[1]Department of Computer Science, Srimad Andavan Arts and Science College, India
[2]Department of Computer Science, Periyar E.V.R. College, India

[*]*Corresponding Author: proftnravi@gmail.com*

Available online at: www.ijcseonline.org

Accepted: 18/Aug/2018, Published: 30/Sept./2018

Abstract: Software metrics plays a major role in assessing the quality of software in testing process. Software metrics elucidates the complexity, reusability, maintainability and understandability of the software code. Software complexity metrics are one of the emerging types of software metrics that focuses on the cognitive analysis of software in terms of understandability and maintainability. In other words, it can be rephrased as the effort taken to comprehend the program code for future enhancements. Complexity metrics has a direct impact with the analysis of complexity in software through an intrinsic study on the object oriented features. This paper proposes a novel Coupling Complexity Metric (IMFC), to highlight the complexity that incurs with coupling and weighs the complexity of a class.

Keywords: Coupling, CBO, reusability, maintainability, modifiability

I. INTRODUCTION

Software metric is the process of continuous evaluation of software. Software metrics employ techniques and algorithms with the primary goal of delivering quality software product. The term metric denotes a specific set of evaluation measurements that are to be applied on particular product or item so as to prove its credibility or usability in its real time applications. Thus, the goal of software metrics are to assess, predict and validate the software in terms of quality. Software metrics are usually incorporated with testing phase of software development life cycle. Software testing not only verifies the requirements, design, and functionalities of code but also to ensure the qualitative writing of program.

At present, Object-Oriented Programming Language (OOPL) is the most popular and widely used software programming paradigm in IT industries since because of its various advantages of software reuse, ease of maintenance and extensibility and lets a paradigm shift from procedure oriented programming (POP) to Object Oriented Programming (OOP) almost in all computing domains. With this increased complexity and the multidimensionality of OO systems, it is inevitable for the programmers to set out the quality parameters for automatically measuring the quality of OO software in the development stage itself. Therefore, the present system necessitates more researches from different perspective on the assessment of complexity in software code. There have been continuous efforts made from numerous researchers for evaluating the complexity of software code since 1994. The very most foundation of software metric suite is proposed by Chidamber and Kemerer (CK) metrics and the other is Metric for Object Oriented Design (MOOD) from Li and Hendry metric. From then on, the research on software metric has been emerging and has been found as a useful practice to be implemented in software development.

Software complexity has been proven to be one of the major contributing factors of the cost of developing and maintaining software. Software complexity is also a key quality indictor and has an impact on many software qualities attributes such as efficiency, reliability and testability. Complex software often challenges the programmers for future modification or change in the functionalities of software code. Moreover, the extensibility of complex software is nightmare and takes much of programmer's time for adding new modules in the existing system. There are lots of complexity metrics proposed for traditional procedure oriented programming. But, the proposal of software complexity metrics for the intrinsic characteristics of object oriented programming is still diminutive and yet to be proposed for the better development of OO software.

A software complexity metric is defined as a metric that specifically measures the complexity involved for

comprehending, understanding and modifying the software code. The verification of complexity of code depends on how well the modularization of the software program is constructed. The two important factors that can effectively assess the complexity in modularization of the program code are coupling and cohesion. Coupling is the measure of the degree of relationship between modules. The measurement of coupling over the structured development context was first defined by Stevens et al. during the year 1974 [1]. Coupling measures the interdependencies between one or more objects. For example, objects A and B are said to be coupled if a method of object B accesses or calls a method or variable in object A. A classic design of the object-oriented programming necessitates the modules to be designed with low coupling [2]. As low coupling has a direct impact with the quality of good program code, it may be obligatory for the software to be assessed with the identification of types of coupling in object oriented programming. The types of coupling called, subclass coupling and temporal coupling [3] are the two streams of object oriented coupling where the prior describes the relationship between a parent and its children and the posterior bundles two actions into one module as they just happen to occur at the same time. Cohesion refers to the degree to which the elements within the module are integrated within the methods of the identical module. There are seven probable types of cohesion exists in a module such as co-incidental, logical, temporal, communicational, sequential, procedural and functional with which co-incidental refers to a poor representation of module and functional cohesion refers to a module design with high integrity.

Coupling in software has been linked with maintainability and existing metrics are used as predictors of outside software quality attributes such as fault-proneness, impact analysis, ripple effects of changes, changeability, etc. Many coupling measures for object-oriented (OO) software have been planned each of them capturing precise dimensions of coupling.

II. REVIEW OF LITERATURE

A). Coupling Between Objects (CBO)

CBO for a class is a count of the number of other classes to which it is coupled. CBO relates to the notion that an object is coupled to another object if one of them acts on the other, i.e., methods of one use methods or instance variables of another. As stated earlier, since objects of the same class have the same properties, two classes are coupled when methods declared in one class use methods or instance variables defined by the other class. Excessive coupling between object classes is detrimental to modular design and prevents reuse [4]. The more independent a class is, the easier it is to reuse it in another application. In order to improve modularity and promote encapsulation, inter-object class couples should be kept to a minimum. The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult. A measure of coupling is useful to determine how complex the testing's of various parts of a design are likely to be. The higher the inter-object class coupling, the more rigorous the testing needs to be.

B). Message Passing Coupling (MPC)

The Coupling through Message Passing (CTM) defined as the number of different messages sent out from a class to other classes excluding the messages sent to the objects created as local objects in the local methods of the class. Two classes can be coupled because one class sends a message to an object of another class, without involving the two classes through inheritance or abstract data type. Theoretical view given was that the CTM metric relates to the notion of message passing in object-oriented programming. The metric gives an indication of how numerous methods of other classes are desirable to fulfill the class' own functionality [5].

C). Coupling Factor (CF)

Coupling Factor (CF) Coupling can be due to message passing (dynamic coupling) or due to semantic association links (static coupling) among class instances. It has been known that it is desirable that classes communicate with as few other classes and even when they communicate; they exchange as little information as possible [6]. It is formally defined as:

$$CF = \frac{\sum_{i=1}^{TC}\sum_{j=1}^{TC}[is_{child}(C_i, C_j)]}{TC^2 - TC} \quad \ldots (1)$$

Where, TC is the total number of classes

$$is_{client}(C_c, C_s) = \begin{cases} 1 iff C_c = C^C C_s \ldots \\ 0 otherwise \end{cases}^c \quad (2)$$

Couplings due to the use of the inheritance are not included in CF, because a class is heavily coupled to its ancestors via inheritance. If no classes are coupled, CF = 0%. If all classes are coupled with all other classes, CF = 100%.

D). Response for class (RFC)

The response set of a class (RFC) is defined as set of methods that can be executed in response and messages received a message by the object of that class. Larger value also complicated the testing and debugging of the object through which, it requires the tester to have more knowledge of the functionality. The larger RFC value takes more complex is class is a worst case scenario value for RFC also helps the estimating the time needed for time needed for testing the class.

III. METHODOLOGY

This section explains the proposed software metric called Inverse Module Frequency Coupling (IMFC). The objective of the metric is to exhibit the quotient of coupling in inter-module attributes in both conventional and OO software. The method computes the frequency of each attributes within software modules and calculates the incidence of those attributes using the method inverse of frequency of variables. The idea has been acquired from the concepts of clustering in data mining. When the documents are clustered, the term frequency and inverse document frequency have been computed to cluster similar documents. The documents that share more number of terms with the same frequency are grouped in a cluster. Similarly, the modules that share the common attributes between the modules are identified and divided by the total number of modules for measuring the level of coupling in the software. The following Equation Equ.1 denotes the formulae for the measuring of inter-module coupling of attributes.

$$IMF = \frac{\sum_{i=1}^{n} log_e\left(\frac{m}{NM(i)}\right)}{log_e(m)} \quad \ldots (3)$$

$$IMFC = \frac{IMF}{m} \quad \ldots (4)$$

Where,
iis represents an attribute from 1….n
m denotes the total number of modules in the software
NM (i) is the number of modules with attributes ‘i’ in it

For instance, table 1 denotes the description of software with modules that performs arithmetic operations. Each module performs a specific arithmetic operation such as addition, subtraction, multiplication and division. Assume that there is a parent module that initiates variables v1,v2 and v3. Each module shown in table 1 is the client of parent module.

Table 1. Description of Modules with High Coupling Program

Module Name	***Attribute Names***	***Method Name***
Addition	v1,v2,v3	Add()
Subtraction	v1,v2,v3	Sub()
Multiplication	v1,v2,v3	Mul ()
Division	v1,v2,v3	Div ()

The coupling value of this software can be calculated as follows:

Variable ‘v1’ has been used in almost all modules in the software. Thus, the inverse attribute frequency of variable ‘v1’ is computed as the fraction of total number of modules by the number of modules that uses variable ‘v1’. Total number of modules in the software is 4. The number of modules uses the variable ‘v1’ is also 4, hence the fraction is 1. Log (1) is 0. In this way, the inverse frequency of all variables is calculated. Since, all three variables ‘v1’,’v2’, and ‘v3’ are shared in all four modules the inverse frequency is ‘0’. The IMFC of the software is calculated as the sum of inverse frequency of all variables by the total number of modules. Thus, the sum is ‘0’ and the fraction of ‘0’ divided by ‘4’ is ‘0’. The IMFC value is ‘0’ for the software, which depicts the degree of relationship between the modules are high.

Another example of software with low coupling is denoted in Table 2 proposed for the same type of software.

Table 2. Description of Modules with Low Coupling Program

Module Name	***Attribute Names***	***Number of Methods***
Addition	v1	Add()
Subtraction	v2	Sub()
Multiplication	v3	Mul()
Division	v4	Div()

The variable ‘v1’ has been used only in Addition module. Thus, the inverse attribute frequency of variable ‘v1’ is log (4/1) which is 0.60205999132. The sum of inverse module frequency of all variables is 2.40823996531. IMF of software is calculated as the fraction of 2.40823996531with log (4) which is 4. The IMF value is again divided by total number of modules to compute IMFC which is 1. Thus, the coupling factor of inverse module attribute frequency of the given software is 1. Since, the modules do not share any variable the coupling factor of the software depicted in table 2 is 1, which depicts the degree of relationship between the modules are low. Table 3 denotes the comparison of the proposed IMFC metric with the traditional Coupling CBO metric.

Table 3. Comparative Analysis of IMFC with CBO

Program Name	***IMFC***	***CBO***
High Coupling Program	1	4
Low Coupling Program	0	0

IV. RESULTS AND DISCUSSION

The result of CBO does not depict the severity of coupling in the software. However, the empirical studies show that the software with CBO value more than 50 is

complex. But, the value 1 in IMFC depicts that the software is highly complex, means, when the modules share all the variables among them and 0 if no variable is shared.

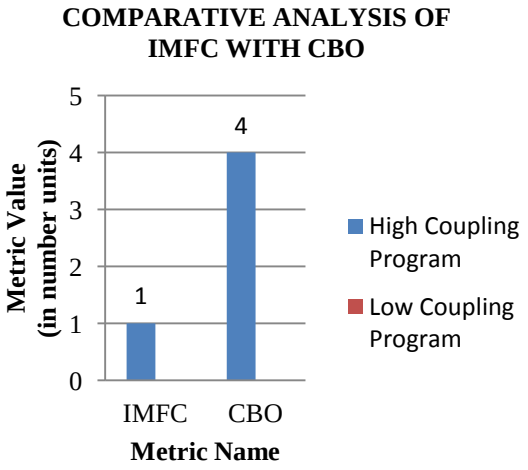

Fig.1. Comparative Analysis of IMFC with CBO

The details of the samples were collected through a questionnaire. There were two lower and higher complexity versions of stock market program, where each pair depicts a perfective and corrective maintenance tasks. The samples were split into two groups consists of members each and were asked to yield the results of the programs by understanding the flow of classes in a module (Perfective), and also instructed to modify the segment of code (Corrective). The starting and ending time to understand and modification of the samples was noted. The proposed metric makes a clear distinction between high and low couplings. Thus, has proven to be better than the traditional ones. The pictorial representation of the comparative study is shown in Figure 1.

V. THEORETICAL VALIDATION OF IMFC

Many inventions have suggested that the software metric should satisfy certain properties for their real time usability in software testing. Basili and Reiter [8] suggested that software metrics should be sensitive to external observable differences in development process, and should correspond to intuitive notions about the characteristic differences between the software artifacts being measured. Weyuker's has also proposed an authorized list of properties for software metrics that could be evaluated on the existing software metrics [9]. The notions of the Weyuker's properties include permutation, interaction, monotonicity, non- coarseness. Many researchers have recommended various properties uniqueness and so on. The challenge in this section is to evaluate the proposed IMFC against the nine properties of Weyuker's to prove its usefulness.

Though, several Weyuker's properties are considered to be most significant to classify the complexity of a measure. Weyuker's properties state that

Property1
Non-coarseness:

Not all class can have the same complexity. If there are 'n' numbers of modules in the software, IMFC does not rank all 'n' modules as equally complex.

Property 2
Granularity:

Let 'r' be a non-negative number and there could be only finite number of modules have the complexity r. If the number of modules in large scale system is finite, the complexity value of IMFC is also finite. Hence this property is satisfied.

Property 3
Non-uniqueness:

This property implies that there may be number of modules have the same complexity. IMFC abides this property, if the hierarchies of class in the modules are similar.

Property 4
Design details are important:

The property affirms that though if two classes have the same functionality, they may differ in terms of details of implementation. If the design implementation of two modules is different, IMFC produces different complexity values for each module.

Property 5
Monotonicity:

Let the concatenation of two modules R and S be R+S. Hence, this property states that complexity value of the combined class may be larger than the complexity of the individual classes R or S. IMFC abides this property if there is a possibility of inheritance between the modules R and S while concatenation.

Property 6
Non-equivalence of interaction:

This property states that if a new module is added to the two existing modules R and S which has the same module complexity, if a new module T is added with both modules, the module complexities of the two new combined modules may be different or the interaction between R and T may be different than the interaction between S and T resulting in different complexity values for R + T and S + T. IMFC for sure yields different complexity values for both modules R and S since T is dependent on the fitness of inheritance with the existing modules R and S.

Property 7
Permutation:

There are program bodies I and J such that J is formed by permuting the order of the statements of I and (|I| = |J|). This property is not taken into the consideration of object oriented metrics.

Property 8
Renaming:

If module R is renamed as S then |R| = |S|. This property requires that renaming a module should not affect the complexity of the module. IMFC does not have any impact over the change of name of module, hence IMFC satisfies property 8.

Property 9
Interaction increases complexity:

The property says that the class complexity measure of a new class combined from two classes may be greater than the sum of two individual class complexity measures. This property is not satisfied with IMFC as the complexity of combined modules could be possibly equal to the individual complexity but not greater. Summary of the IMFC validation is described in Table 3.

Table 3. Evaluation of IMFC Metric with Weyuker's Properties

Metric	***P1***	***P2***	***P3***	***P4***	***P5***	***P6***	***P7***	***P8***	***P9***
IMFC	Y	Y	Y	Y	Y	Y	N	Y	Y

CONCLUSION

Coupling plays a vital role in assessing the quality of software modules. Coupling also reflects in the cognitive analysis of program since a module with low coupling is easily interpretable and understandable than the module with high coupling. This paper proposed a new software coupling metric called IMFC for assessing level of coupling by assigning cognitive weights of variables inside a class. . Moreover, the higher complexity in software leads to more cost expensive and less maintainability of software. The assurance of less complexity software is of been great interest to researchers since the early days of development. Hence, the proposed IMCF metric will be helpful for the developers to identify the flaws in their program in the development stage itself.

REFERENCES

[1]. S.R. Chidamber and C.F. Kemerer. Towards a metrics suite for object-oriented design. In Object Oriented Programming Systems Languages and Applications, pages 197-211, Phoenix, Arizona, USA, November 1991.

[2]. Christodoulou. D and Qi.X,"Difficulties in Software Measurement",http://www.dcs.shef.ac.uk/~m3xq/om6660/diff_sm .pdf.

[3]. Henderson-Sellers. B, "Object-Oriented Metrics: Measures of Complexity", Prentice Hall, New Jersey, 1996.

[4]. Chidamber, Shyam and Kemerer, Chris, "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, June, 1994, pp. 476-492.

[5]. W.Li,"Another metric suite for object-oriented programming, "Journal of Systems and Software, vol. 44, no. 2, pp. 155–162, 1998.

[6]. Abreu F.B. and R.Carapuca "Object-Oriented Software Engineering: Measuring and Controlling the Development Process "Proceedings of the 4th International Conference on Software Quality, McLean, Virginia, USA, and October, 1994.

[7]. Cheolhyun Park, Junhee Kim, and Eunseok Lee. Using Page Rank Algorithm to Improve Coupling Metrics. ACEEE Int. J. on Information Technology, Vol. 02, No. 01, March 2012

[8]. Basili, Victor R., and Robert W. Reiter Jr. "Evaluating automatable measures of software development" In Proceedings on Workshop on Quantitative Software Models, pp. 107-116. 1979.

[9]. Michael, James Bret, Bernard J. Bossuyt, and Byron B. Snyder. "Metrics for measuring the effectiveness of software-testing tools." In Software Reliability Engineering, 2002. ISSRE 2003. Proceedings. 13th International Symposium on, pp. 117-128. IEEE, 2002.

