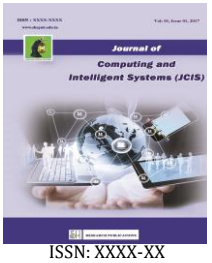
Original Research Article

J. Computing & Int. Systems (2017) 10–14

SACRED HEART RESEARCH PUBLICATIONS

Journal of Computing and Intelligent Systems

Journal homepage: www.shcpub.edu.in



ISSN: XXXX-XX

A Novel Cognitive Complexity Metric for Inheritance Factor

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

Received on 31st Mar 2017, Accepted on 05th May 2017

Abstract — *The complexity of Object Oriented (OO) software is often measured with the intra-inter relationship within the classes of a module. The more the classes are related, the more the occurrence of complexity is. Inheritance concepts are one of the most widely used and inevitable feature of OO programming that often enhances the possibility of method reuse and module extensibility. However, a module designed with high quotient of inherited classes increases the coupling factor of the module which lets a modification and understanding of one class pre-requisites the knowledge of other related classes. The assessment of coupling complexity exists in the inherited classes is very much useful in the evaluation of the overall software. Hence, in this paper, a Cognitive Complexity Inheritance Metric (CCIM) is proposed to define the complexity of inherited classes in a module in terms of understandability and modifiability. The metric is validated against both empirical and experimental evaluations where the results clearly highlight the cognitive complexity of inherited classes in software modules.*

Keywords: *OO metric, coupling, CCIM, Complexity metrics*

1 INTRODUCTION

The basic notion of software metrics is to assess the quality of software products. Software metrics aids as a tool to control software complexity. The metrics help the developer to observe the weaknesses of developed software there by valuing the quality of the software. Hence, software metrics are considered to be an indispensable task in the course of software development life cycle for achieving high qualitative software. In the present decades, almost all software projects are developed using Object-Oriented (OO) programming languages because of their salient features like as modularity, reusability and extendibility [1]. Thus, the analysis on the quality of OO programming is also an inevitable process and is highly achievable through OO software metrics.

Software complexity metrics are yet another branch of software metrics that attempt to measure the effort or degree of criticality evolved in understanding or comprehending the software code based on the three fundamental factors of input, process and output. Though the importance of software complexity metrics in software maintainability and understandability is high, the majorityof software complexity metrics that have been proposed for procedural programming is still being used for assessing the complexity OO programming [2]. Therefore, the present situation is in need of the proposal of newer cognitive complexity metric suite for OO languages.

This paper is proposed with an intention of introducing a novel cognitive complexity inheritance metric CCIM for understanding and comprehending the complexity exists in the inheritance concepts. CCIM elucidates the cognitive complexity value of inherited classes where the high CCIM value denotes lower complexity and in contrast the lower CCIM value denotes that the module consists of high complexity. The remaining section of the paper is organized as follows: Section II consists of the review of literature, section III entails of the methodology and section IV contains the empirical validation of the metric, section V explicates the experimental validation of CCIM and finally section VI concludes the findings of the paper.

2 RELATED WORK

A. Cognitive Weighted Response for a Class (CWRFC)

CWRFC metric is used for measuring the complexity involved in message passing [3]. Supposing if a class holds 'n' number of response sets CWRFC calculates the complexity of the class using the response set complexity as shown in equation 1.

$$CWRFC = \sum_{i=1}^{n} RSC_i \quad \ldots (1)$$

Where RSC denotes the response set complexity, which is calculated by summing the set of all m methods in a class and set of R methods called by any of those methods.

$$RSC = \forall_i R_i + M \quad \ldots (2)$$

As per message passing, the methods of the classes are segmented into two as, Methods With Arguments (MWA) and methods without arguments (MOA). MOA is also referred as Default Function (DF). The arguments of MWA can either be passed through Pass By Value (PBV) or Pass By Reference (PBR). Hence, R can be computed using the formula shown in equation 3.

$$R = DF \times (CW_f + WF_d) + PBV \times (CW_f + WF_v) + PB \times (CW_f + WF_r) \quad \ldots (3)$$

* Corresponding author: E-mail: vijay_sjctni@yahoo.co.in, proftnravi@gmail.com

[1] Assistant Professor, Dept. of Computer Science, Srimad Andavan Science College, Tamilnadu, India

[2] Assistant Professor, Dept. of Computer Science, E.V.R College, Tamilnadu , India

where, DF is the total number of default functions
PBV is the total number of Pass By Value Function Call Statements
PBR is the total number of Pass By Reference Function Call Statements
CW_f is the CWs of the Function Call Statement
WF_d is the Weighting Factor of the DFCS
WF_v is the Weighting Factor of the PBV statements
WF_r is the Weighting Factor of the PBR statements

B. Cognitive Weighted Coupling Between Objects (CWCBO)

The motivation for defining CWCBO metric is to elucidate the complexity involved with coupling of classes by considering the different types of coupling such as control, data, interface and global couplings [4]. The unnecessary object coupling increases the complexity the chances of system exploitation. CWCBO can be calculated using the equation 4.

$$CWCBO = ((CC \times WFCC) + (GDC \times WFGDC) + (IDC \times WFIDC) + (DC \times WFDC) + (LCC \times WFLCC) \quad \ldots (4)$$

Where
CC is the total number of modules that contains Control Coupling
WFCC is the Weighting Factor of Control Coupling
GDC is the count of Global Data Coupling
WFGDC is the Weighting Factor of Global Data Coupling and its weight is given as 1
IDC is the count of Internal Data Coupling
WFIDC is the Weighting Factor of Internal Data Coupling and its weight is given as 2
DC is the count of Data Coupling
WFDC is the Weighting Factor of Data Coupling and its weight is given as 3
LCC is count of Lexical Content Coupling
WFLCC is the Weighting Factor of Lexical Content Coupling and its weight is given as 4

C. Cognitive Weighted Polymorphism Factor (CWPF)

Thamburaj et al. [5] have proposed CWPF. The goal of CWPF metric is to evaluate the complexity of software with respect to three types of polymorphisms such as pure, static and dynamic polymorphisms. The metric calculates the cognitive complexity arising from the efforts needed to comprehend the different types of polymorphism involved in the software rather than calculating only the architectural complexity of the polymorphism which is shown in equation 5

$$CWPF = \frac{\sum_{i=1}^{TC} CWM_o(C_i)}{\sum_{i=1}^{TC}[M_n(C_i) \times DC(C_i)] \times ACW} \quad \ldots (5)$$

Where, $CWM_o(C_i)$ is the number of overriding methods in class C_i
$DC(C_i)$ is the number of children of class C_i
TC is the total number of classes. The calculation of *ACW* is done by the equation 6.

$$ACW = (CW_{PP} + CW_{SP} + CW_{DP}) \quad \ldots (6)$$

Where
CW_{PP} is the cognitive weight of pure polymorphism
CW_{SP} is the cognitive weight of static polymorphism
CW_{DP} is the cognitive weight of dynamic polymorphism

D. Cognitive Weighted Attribute Hiding Factor (CWAHF)

CWAHF metric enhances cognitive perspective on the visibility of different types of attributes which are commonly divided into private, protected and public [6]. Private arguments are the arguments that are fully invisible, protected means partially visible and public means fully visible. The default visibility comes under the package private scope and does not have any keyword. The equations 7, 8 and 9 denote the calculation of CWAHF

$$CWAHF = \frac{\sum_{i=1}^{TC} A_h(C_i)}{\sum_{i=1}^{TC} A_h(C_i) + \sum_{i=1}^{TC} A_v(C_i)} \quad \ldots (7)$$

$$\sum_{i=1}^{TC} A_h(C_i) = \sum_{i=1}^{TC} A_p(C_i) \times CW_{pa} + A_d(C_i) \times CW_{da} + A_t(C_i) \times CW_{ta} \quad \ldots (8)$$

$$\sum_{i=1}^{TC} A_v(C_i) = \sum_{i=1}^{TC} A_u(C_i) \times CW_{ua} \ldots (9)$$

Ap(Ci) is the number of private arguments
CWpa is the cognitive weight of private arguments
Ad(Ci) is the number of default arguments
CWda is the cognitive weight of default arguments
At(Ci) is the number of protected arguments
CWta is the cognitive weight of protected arguments
Au(Ci) is the number of public arguments
CWua is the cognitive weight public argument

3 METHODOLOGY

Inheritance plays a major role in the complexity of a module. Though, the concepts of inheritance enhance reusability and extendibility, the module with high quotient of inherited classes in fact increases the complexity. So far there is no such metric that assesses the complexity involved with inheritance which is a motivating factor of the paper. To start with the cognitive complexity weight of the leaf node is initially set to 1 and has been increased sequentially when the tree is traversed from leaf to root nodes as the conceptualization of successor class requires the knowledge of the predecessor classes. The accumulation of the cognitive weights of all leaf nodes depicts the total cognitive complexity of the inheritance tree which is then divides the total number inherited classes in the module. The cognitive complexity weight of the leaf nodes can be derived using equation 10 as follows

$$CCW_{LF} = \sum_{i=1}^{m} NNR \quad \ldots (10)$$

Where
i represents the current leaf node
m represents the total number of leaf nodes
NNR is the Number of nodes to the root
$CCW_{LF}$is the accumulation of cognitive weights of the leaf nodes

The cognitive complexity weight obtained using equation is used to identify the overall cognitive complexity of the inherited classes in the module using the following equation 11.

$$CCIM = \frac{n}{CCW_{LF}} \ldots (11)$$

Where
n is the total number of inherited classes

CCW_{LF} is the accumulation of cognitive weights of the leaf nodes
CCIM is the cognitive complexity inheritance metric

4 ILLUSTRATION

As an illustration two modules with different pattern of inherited modules are defined to calibrate the complexity of inheritance and shown in Figure 1.

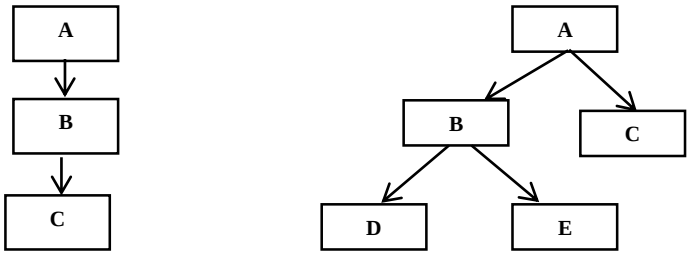


Figure 1.
a. Multilevel Inheritance Module *b. Hybrid Inheritance Module*

The leaf node of Multilevel Inheritance Module is c. Hence, the CCW_{LF} of Multilevel Inheritance Module is calculated by traversing the tree from C to A which requires the understanding of class A requires the understanding of class B and C which is 3.

$$CCW_{LF}(Module\ 1) = \sum_{i=1}^{1} NNR = 3$$

Since, there is only one leaf node in Multilevel Inheritance Module and the number of nodes from the leaf C to root A is 3, the CCWLF of Multilevel Inheritance Module is 3. The CCIM value of module is calculated as

$$CCIM = \frac{n}{CCW_{LF}(Module\ 1)} = \frac{3}{3} = 1$$

Likewise the CCIM calibration of Hybrid Inheritance Module is calculated below. The total number of leaf nodes in Hybrid Inheritance Module is 3 such as D, E and C, Hence the value of m is 3. The number of nodes from D to root A is 3, E to A is 3 and C to A is 2. Hence, the cognitive complexity weight of the leaf nodes of Hybrid Inheritance Module is 8.

$$CCW_{LF}(Module\ 2) = \sum_{i=1}^{3} NNR = 8$$

CCIM of Hybrid Inheritance Module is the fraction of the number of inherited classes with the cognitive complexity weight of the leaf nodes as described below

$$CCIM = \frac{n}{CCW_{LF}(Module\ 1)} = \frac{5}{8} = 0.625$$

5 COMPARISON OF DIT WITH CCIM

The proposed CCIM metric is compared with its base Depth Inheritance Tree (DIT) metric to highlight the cognitive complexity of inheritance concepts in object oriented metrics. DIT measures only the number of ancestors classes that affect the measured classes [9]. In case of multiple inheritance, the metric gives the longest path from the class to the root class which does not indicate the number of classes involved where the complexity of the class actually lies.

DIT of Multilevel Inheritance Module is 3 and Hybrid Inheritance Module is also 3, whereas the cognitive complexity of Multilevel Inheritance Module and Hybrid Inheritance Module is 1 and 0.625. Table 1 depicts the metric values of DIT and CCIM respectively.

TABLE 1
Comparison of DIT vs CCIM

Program	DIT	CCIM
Multilevel Inheritance Module	3	1
Hybrid Inheritance Module	3	0.625

When comparing to Multilevel Inheritance Module, Hybrid Inheritance Module has the highest complexity with more number of classes. But, the DIT metric value shows the same value for both the modules. On the other hand, the metric values of CCIM differ from each other where the value 1 of Multilevel Inheritance Module designates that the complexity is low and 0.625 in Hybrid Inheritance Module is comparatively complex than Module1. The pictorial representation of the comparisons of DIT and CCIM is depicted in Figure 2.

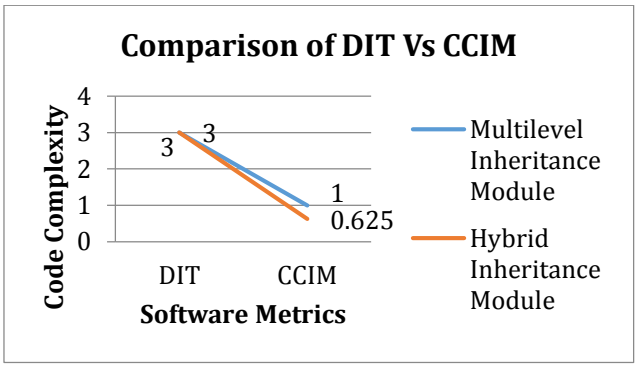


Figure 2. Comparison of DIT Vs CCIM

6 THEORETICAL VALIDATION

Software metric has to satisfy certain validation properties for proving their usefulness in real time implementations. Among the various software metric validations, the properties suggested by Basili and Reiter is very much sensitive to the evaluation of software [7] metric, as it observes the external differences in software development. The properties also exploit the intuitive notions of the inherent characteristics of the software metric and the differences between the artifacts.

Weyuker has also presented a formal list of nine properties that evaluates the characteristics of any novel or existing software metrics [8]. The properties of Weyuker denote the important notions that the software metric should possess such as non-coarseness, granularity, interactions, permutations, monotonicity, and uniqueness etc. The objective of this section is to validate the proposed CCIM metric against the Weyuker's nine properties to prove the validity of the metric.

Several properties of Weyuker's may even significantly helpful in classifying the complexity of the software. The properties of the Weyuker's are discussed below:

Property1

Non-coarseness
Not all modules have the same metric value. If the software consists of 'n' number of modules, CCIM does not result the same metric value for all 'n' modules. Hence property 1 is satisfied by CCIM.

Property 2

Granularity
Let 'c' be the finite number of modules having the same metric value. If the software consists of 'n' number of modules, the metric value provided CCIM is for 'c' finite modules. Thus, property 2 is satisfied by CCIM.

Property 3

Non-uniqueness
It is acceptable if the number of modules in the software has the same metric value. CCIM satisfies this property, if the inheritance tree between classes within the modules is same.

Property 4

Design details are important
This property states that if two modules perform the same functionality, but vary in terms of implementation design, then the software metric may result different value for each module. CCIM affirms this property if the classes in the modules are represented with different inheritance models. Thus, CCIM satisfies property 4.

Property 5

Monotonicity
If two modules M and N are concatenated as M+N, then the complexity of the concatenated class must be larger than the complexity of the discrete modules M and N. CCIM affirms this property when the modules are concatenated with inheritance relationship. Thus property 5 is satisfied with CCIM.

Property 6

Non-equivalence of interaction
If a module O is added with two existing modules M and N having the same complexity, the complexity of the newly added modules O+M may different from the complexity of O+N. CCIM for sure produces a different complexity values for both modules M+O and N+O since O is dependent on the fitness of inheritance with the existing modules M and N. Thus CCIM affirms property 6.

Property 7

Permutation
If the program bodies M and N are permutated in such a way that N is formed by changing the order of statements of N then (|M|=|N|). CCIM does not abide property 7 it is not suitable for object oriented metrics.

Property 8

Renaming of modules does not affect the complexity. If the name of module M is changed as N then the complexity of M and N must be same as |M|=|N|. CCIM does not have any influence over the complexity of renaming the modules, Thus, CCIM satisfies property 8.

Property 9

Interaction increases complexity
Let O be the new class combined from two classes M and N, then the property states the complexity of the new class may be greater than the sum of complexity of two individual classes M+N. This property does not satisfied with CCIM as the complexity of combined modules could be possibly equal than the individual complexity but not greater. Summary of the CCIM validation is described in Table 2.

TABLE 2.
Summary of CCIM validation with Weyuker's Properties

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

7 CONCLUSION

Software cognitive complexity metrics on inheritance factor in object-oriented concepts is one of the essential factors to identify the quality of software in terms of maintainability. This paper proposes new software metric called CCIM for assessing the level of cognitive complexity involved in the inherited classes in the module. The results of the CCIM show that the low CCIM value of increases to higher cognitive complexity of the module and vice-versa. 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 CCIM metric will be helpful for the developers to identify the flaws in their program in the development stage itself.

REFERENCES

[1] Misra, Sanjay, Murat Koyuncu, Marco Crasso, Cristian Mateos, and Alejandro Zunino. "A suite of cognitive complexity metrics." In International Conference on Computational Science and Its Applications, pp. 234- 247. Springer Berlin Heidelberg, 2012.

[2] Dr. L.Arockiam and A.Aloysius, "Attribute Weighted Class Complexity: A New Metric for Measuring Cognitive Complexity of OO Systems", International Journal of Computer, Electrical, Automation, Control and Information Engineering Vol:5, No:10, 2011,

[3] Dr. L.Arockiam and A.Aloysius, "Maintenance Effort Prediction Model Using Cognitive Complexity Metrics", International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 11, November 2013.

[4] Dr. L.Arockiam and A.Aloysius, "Coupling Complexity Metric: A Cognitive Approach", I.J. Information Technology and Computer Science, 2012

[5] T. Francis Thamburaj, A. Aloysius, "Cognitive Weighted Polymorphism Factor:A Comprehension Augmented Complexity Metric" , International Journalof Computer, Electrical, Automation, Control and Information Engineering Vol:9, No:11, 2015.

[6] T. Francis Thamburaj, A. Aloysius," Cognitive Perspective Of Attribute Hiding Factor Complexity Metric", International Journal Of Engineering And Computer Science ISSN: 2319-7242 Volume 4 Issue 11 Nov 2015
[7] 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.
[8] 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.
[9] Sheldon, Frederick T., Kshamta Jerath, and Hong Chung. "Metrics for maintainability of class inheritance hierarchies." Journal of Software Maintenance and Evolution: Research and Practice 14, no. 3 (2002): 147-160.