International Journal of Contemporary Research in Computer Science and Technology (IJCRCST) e-ISSN: 2395-5325
Volume 2, Issue 10 (October '2016)

AN ANALYSIS ON THE INITIATIVES OF COGNITIVE COMPLEXITY METRICS IN OBJECT – ORIENTED PROGRAMMING: A SURVEY

N. Vijayaraj,
Assistant Professor,
Department of Computer Science,
Srimad Andavan Arts and Science College,
Trichy, Tamilnadu, India.

T.N.Ravi,
Assistant Professor,
Department of Computer Science,
Periyar E.V.R. College,
Trichy,Tamilnadu,India.

Abstract: The primary goal of software metric is to evaluate the quality of built software. The conventional software metrics are majorly categorized into procedure and object oriented. Despite the emergence of newer technologies, object oriented (OO) software programming are said to be evolutionary as they are widely used in software design and development in the recent era. Though there have been many efforts made in developing software metrics for OO programming, the real time implementation of those metrics is still seem to be diminutive. Software Cognitive complexity metrics are yet another newer dimension of software metrics which are known to be simple and ready to implement in various domains. The core objective of this paper is to analyze the usability and maintainability of software cognitive complexity metrics in software industries.

Keywords: Software metrics, OO programming, Maintainability, Usability, Cognitive, Complexity

I.INTRODUCTION

Quantitative measures are indeed essential for all scientific inventions as it acts as a justifying factor of the novelty of any research work. Software metric is a quantitative measure of quality of software. There have been continuous efforts made by computer science practitioners and researchers to propose numerous software metrics that can be used in the development stage of software with the objective of assessing them with quantifiable scales. The result of software metrics ensures the hidden characteristics of software such as complexity, maintainability, modifiability and reusability. The benefits of software metrics are also extended to evaluate the cost required for testing, debugging and performance optimization.

Software complexity metrics are one of the vital classifications of software metrics that encompasses the measures for quantifying the complexity involved in the software code. Hence, the research on software complexity metrics has always been considered as thrust area in research. The three important attributes of software complexity metrics are essential, selection and incidental [1]. The essential software complexity metrics focuses on the problems that the software tries to solve, the selecting software complexity metrics focuses on the problems involved in the software models and design and finally, the incidental software complexity metrics focuses on the improvement of software quality.

OO programming is the most popular and widely used software development programming over the past three decades. The ease of implementation, modifiability, maintenance, understandability and the ability to reuse software code and modules makes the OO programming as standalone and let the developers to opt for. Software complexity metrics in OO programming refers to the complexity of software code with respect to understandability, modifiability, maintainability and reusability. According to Abbot the complexity of a function is the criticality of interactions between the classes, methods and attributes that increases the time taken for the above mentioned activities on the software. Software cognitive complexity metrics are another invention of software metrics that are proposed for the purpose of measuring the code complexity with cognitive characteristics approach. Software cognitive complexity metrics for methods are still under the scope of research which is yet to be improved in software engineering. The objective of this paper is to survey the researches that have been initiated on cognitive complexity metrics their merits and demerits there by identifying the different cognitive approaches that can be helpful in the betterment of software engineering.

The remaining section of the paper is organized as follows: Section 2 describes the existing cognitive complexity metrics and finally section 3 concludes the findings of the survey.

II.COGNITIVE COMPLEXITY METRICS

Misra et al. [2] have proposed a family of cognitive metrics for evaluating the method, message and attributes complexities involved in OO Codes. The authors have also proposed a code complexity metric by considering the complexity with inheritance. The authors have used the cognitive aspect of code in terms of assigning weight. The metrics are described as follows:

Method Complexity (MC):
MC is computed by assigning the cognitive weights of structures that exists in a method of a class. The weights of the cognitive complexity are measured by the logical and control structures that are resided in methods of software. The logical structures are weighed as one, two, three and two corresponding to sequence, branch, iteration and call respectively. The MC is calculated by associating the weight

with each method of a class and adds the overall weights to attain the cumulative results which can be formulated as shown in equation 1.

$$MC = \sum_{i=1}^{m} \left[\prod_{j=1}^{n} \sum_{k=1}^{o} W_c(i,j,k) \right] \quad \ldots (1)$$

Where W_c refers to the cognitive weights of control structures. The sum of cognitive weights of 'm' linear blocks within the individual control structures is defined as the method complexity of a class.

Message Complexity (Coupling Weight for a Class (CWC)):

Two classes are said to be coupled if and only if there exist a message call from one class to the other. CWC adds the weights of internal and external message calls to CWC, rather than counting the total number of calls. Here, the complexities due to message calls are assessed by summing the weights of call and the weights of called methods, which can be formulated as shown in equation 2.

$$CWC = \sum_{i=1}^{n} (2 + MC_i) \quad \ldots (2)$$

Where, the number 2 is the weight of message to an external method and MC_i is the weight of called method. If the number of external calls is 'n', then CWC is computed as sum of weights of all message calls.

Attribute Complexity (AC):

AC is designed with the principle that the complexity of a class is high, if it contains more number of attributes. The attributes that instantiates an object used in one method may not be used in other methods increases the complexity of a class. The weight of AC is the total number of attributes associated in the class which can be denoted as shown in equation 3.

$$AC = \sum_{j=1}^{n} 1, \quad \ldots (3)$$

Where 'n' is the total number of attributes in the class.

Weighted Class Complexity (WCC):

The structure of OO programming purely depends upon classes and objects whose elements are methods and attributes. The complexity of the class is measured by the number of attributes and the methods that exists in the class. Hence, the WCC of a class is the sum of attribute weights and method weights of the corresponding class. The formula for calculating the WCC is denoted as shown in equation 4.

$$WCC = AC + \sum_{j=1}^{n} MC_j \quad \ldots (4)$$

Where WCC is the sum of attribute complexity and sum of all the method complexities of class.

Code Complexity (CC):

The complexity of the individual modules does not represent the complexity of the entire software. In order to measure the complexity involved in the whole software, there is need for understanding the relationship between the modules. Thus, CC emphasizes on the concepts of inheritance property as the classes of a module may either be parents or children classes. For instance, a child class may inherit the properties of its parent classes. With this principle the CC of the entire software is defined as

- If classes are of same level, add the weights
- If classes are sub or child classes, multiply the weights

If there are 'm' levels of depth in the object-oriented code and level j has n classes then the Code Complexity (CC) of the system is given by equation 4.

$$CC = \prod_{i=1}^{n} \left[\sum_{j=1}^{m} WCC_{jk} \right] \quad \ldots (5)$$

Aloysius et al. [3] have proposed a cognitive complexity metric suite such as Attribute Weighted Class Complexity (AWCC), Cognitive Weighted Response For a Class (CWRFC) and Cognitive Weighted Coupling Between Objects (CWCBO) for measuring cognitive complexity arising due to inheritance, message passing and coupling respectively.

Attribute Weighted Class Complexity (AWCC):

The complexity of a class with AWCC is calculated using the method and attribute complexities with the complexity of the inherited members. Supposing, if a class holds 'm' attributes and 'n' methods and the class is derived from 'o' number of classes then, the AWCC of the corresponding class can be calculated as shown in equation 6.

$$AWCC = \sum_{x=1}^{m} AC_x + \sum_{y=1}^{n} MC_y + \sum_{z=1}^{o} ICC_x \quad \ldots (6)$$

Where, AC is the attribute complexity
MC is the method complexity
ICC is the inherited class complexity

The attribute complexity of CWCBO is the sum of multiplication of data type weights with the number of attributes belonging to the data type, which can be denoted using the formula denoted as equation 7.

$$AC = (PDT * W_p) + (DDT * W_d) + (UDDT * W_u) \quad \ldots (7)$$

Where, PDT is the number of attributes belonging to primary data type
DDT is the number of attributes belonging to derived data type
UDDT is the number of attributes belonging to user-defined data type
W_p is the weight for PDT which is 1
W_d is the weight for DDT which is 2
W_u is the weight for UDDT which is 3
The MC is calculated as defined by Misra et al. and ICC
The formula for calculating the ICC is denoted in equation 8.

$$ICC = (DIT \times C_L) \times \sum_{e=1}^{s} RMC_e + RNA$$

… (8)

Where DIT denotes the depth inheritance Tree metric
CL is the cognitive Load of level L
S is the number of inherited methods
RNA is the total number of reused attributes
RMC is the reused method complexity
IC is the inherited complexity

Cognitive Weighted Response For a Class (CWRFC)

CWRFC metric is used for measuring the complexity involved in message passing [4]. 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 9.

$$CWRFC = \sum_{i=1}^{n} RSC_i$$

… (9)

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.

$$RSC = \forall_i R_i + M$$

… (10)

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 11.

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

… (11)

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

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 [5]. The unnecessary object coupling increases the complexity the chances of system exploitation. CWCBO can be calculated using the equation 12.

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

… (12)

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

Cognitive Weighted Polymorphism Factor (CWPF)

Thamburaj et al. [6] 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 13.

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

… (13)

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 14.

$$ACW = (CW_{PP} + CW_{SP} + CW_{DP})$$

… (14)

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

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 [7]. 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 15, 16 and 17 denotes the calculation of CWAHF.

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

… (15)

$$\sum_{i=1}^{TC} A_h(C_i) = \sum_{i=1}^{TC} A_p(C_i) \times CW_{pa} + A_d(C_i) \times CW_{da} + A_t(C_i) \times CW_{ta} \quad \ldots (16)$$

$$\sum_{i=1}^{TC} A_v(C_i) = \sum_{i=1}^{TC} A_u(C_i) \times CW_{ua} \quad \ldots (17)$$

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

III.CONCLUSION

This survey has discussed the various efforts that are made to assess the quality of software products in the perspective of cognitive complexity. All the metrics that are described in the paper are designed to address the complexities involved in method, attribute, class, code, inheritance, coupling, message passing and polymorphism. But, still there are plenty of OO benefits such as cohesion, modularity, dynamic bindings and method overloading are yet to be addressed in the cognitive analysis. Hence, the future direction of this paper focuses on the development of cognitive complexity metrics for the remaining benefits of the OO programming to enhance the quality measurement of software products.

IV.REFERENCES

[1]. Jakhar, Amit Kumar, and Kumar Rajnish. "Measuring complexity, development time and understandability of a program: A cognitive approach." International Journal of Information Technology and Computer Science (IJITCS) 6, no. 12 (2014): 53.
[2]. 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.
[3]. Dr. L.Arockia 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
[4]. Dr. L.Arockia 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.
[5]. Dr. L.Arockia and A.Aloysius, "Coupling Complexity Metric: A Cognitive Approach", I.J. Information Technology and Computer Science, 2012
[6]. T. Francis Thamburaj, A. Aloysius, "Cognitive Weighted Polymorphism Factor:A Comprehension Augmented Complexity Metric", International Journal of Computer, Electrical, Automation, Control and Information Engineering Vol:9, No:11, 2015
[7]. 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
[8]. Mel Ó Cinnéide, Laurence Tratt, Experimental Assessment of Software Metrics Using Automated Refactoring 2012 ACM
[9]. J. Al Dallal and L. Briand. A precise method-method interaction-based cohesion metric for object-oriented classes.ACM Transactions on Software Engineering and Methodology , 2010.
[10]. J. Al-Dallal and L. C. Briand. An object-oriented high-level design-based class cohesion metric Information & Software Technology , 52(12):1346–1361,2010.
[11]. K.K.Aggarwal, Yogesh Singh, software design metrics for object-oriented software journal of object technology, Vol. 6, No. 1, January-February 2006