International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2017 IJSRCSEIT | Volume 2 | Issue 4 | ISSN : 2456-3307

An Analysis of Object Oriented Complexity Metrics

K. Maheswaran, A. Aloysius
Department of Computer Science, St. Joseph's College (Autonomous), Tiruchirappalli, Tamil Nadu, India

ABSTRACT

Software metrics are essential to measure the quality of software products. Number of metrics related to software complexity, quality, reusability, reliability, maintainability has been developed in the past and are still being proposed. Software metrics are tools to control the complexity of software. This paper briefly discusses cognitive and non- cognitive complexity metrics in Object Oriented (OO) design with respect to the complexity of a class, code, inheritance, interface and polymorphism.

Keywords: Software Metrics, Software Complexity, Cognitive Informatics, Cognitive Complexity

I. INTRODUCTION

Software Metrics are used to measure the quality of software products (design, source code etc.), processes (analyses, design, coding, testing etc.) and professionals (efficiency or productivity of an individual designer). Software metrics allow us to quantitatively define the degree of success or failure of a product, process, or person. It also useful for managerial and technical decisions related to cost, effort, time, complexity, quality etc. Thus, incorporating metrics into software development process is a valuable step towards creating better systems. The software complexity metric is an essential and critical part of the software system. The software complexity metrics focuses on the quality of source codes. There are several software metrics proposed for capturing the complexity, effort, quality, reliability, maintainability of object oriented design. These metrics provide ways to evaluate the software and their use in earlier phases of software development life cycle. If one can quantify the design and thereby increase the quality of the design, the lower probability of the software error.

Object Oriented Design (OOD) is an interesting area for current researchers. Many researchers have worked in recent years Software engineers used software metrics to measure and predict software systems. To produce high quality Object Oriented (OO) applications, a strong emphasis on design aspects is highly necessary. The popularity of the OO software development is due to its powerful features like inheritance, message passing, polymorphism, dynamic binding, encapsulation and objects composition. Further, it is characterized by classes and objects, which are defined in terms of attributes (data) and operations (methods). These elements are defined in class declarations. Among these, the method plays an important role since it operates on data in response to a message. Several authors like Fenton's [1], Shepperd's [2] and others were proposed different software complexity metrics. Cognitive Informatics [13] acts an important role in understanding the fundamental characteristics of the software. Number software complexity measures [13, 23] have been proposed in last few decades by the researchers based on cognitive informatics. Cognitive Complexity Metrics that are used in the procedural programming to identify the complexity of the program, and a few of them are modified in order to satisfy the Object-Oriented programming.

Cognitive complexity measures the human effort needed to perform a task or difficulty to understand the software. This paper also discusses the some of the existing cognitive metrics in the field of OO Software Development. The aim of this analysis is to list out few existing Metrics and to make the reader aware of their existence and to offer references for further reading. The entire paper has been organized into five major sections. In the following section, the basic definitions

are introduced. The various OO software metrics and their variations are analysed in section III. The comparative analyses of various metrics and observations for possible enhancement were presented in section IV. Finally, we make the conclusion in section V.

II. BASIC DEFINITIONS

A. Class

A class is nothing but a blueprint or a template for creating different objects which defines its properties and behaviours

B. Object

An object is an instance of a class. Each object of a class contains a set of data and code to manipulate the data.

C. Inheritance

According to IBM, Inheritance is a mechanism of reusing and extending existing classes without modifying them, thus producing hierarchical relationships between them.

D. Interface

An interface is a blueprint of a class. It has static constants and abstract methods only.

E. Cohesion

Cohesion refers to the degree to which the elements of a module belong together.

F. Coupling

Coupling is the degree to which one class is connected with another class.

G. Polymorphism

Polymorphism means ability to take more than one form. An operation may exhibit different behaviours in different contexts.

H. Cognitive Complexity

The cognitive complexity means the mental burden on the user who deals with the code. It can be calculated in terms of time taken to understand the code.

III. OBJECT ORIENTED METRICS

Number of software complexity metrics are developed to assure the quality of the software such as maintainability, reliability etc., were proposed the past and present. Several traditional metrics were designed for structured systems. Among them McCabe's Cyclomatic complexity metric, Halstead's complexity metric and Kafura's and Henry's fan-in, fan-out are most commonly used metrics [3, 4, 5]. For object-oriented systems Chidamber and Kemerer metric suite forms the foundations. In this section, various Object-Oriented software metrics were discussed one by one.

A. Class Level Metrics

Chidamber et al., [6] listed six metrics for OO design. Out of that first and foremost metric is Weighted Method per Class (WMC) which is represented in eq. (1). It only considers the method complexity rather than other parameters of the class. Let $c_1 \ldots c_n$ be the complexity of the methods.

$$\text{WMC} = \sum_{i=1}^{n} C_i \qquad (1)$$

Balasubramanian [3] suggested the improved version of CK metrics as Class Complexity (CC). This metric considers instance variables apart from the methods of a class. It does not have the OO features.

$$\text{CC} = \text{iv} + \text{wm} \qquad (2)$$

In eq. (2), *iv* denotes number of instance variables in a class and *wm* denotes sum of weighted method complexity in a class.

Misra et al., presented Class Complexity Metric by using method complexity and complexity due to inheritance which covers cognitive complexity [8]. There are *m*-levels of depth in the OO code and level j has n classes then the class complexity (CC) of the system is calculated in eq. (3)

$$\text{CC} = \prod_{j=1}^{m}\left[\sum_{k=1}^{n} W_{cjk}\right] \qquad (3)$$

$$W_e = \sum_{j=1}^{q}\left[\prod_{k=1}^{m}\sum_{i=1}^{n} W_c(j,k,l)\right] \qquad (3.1)$$

Where W_c is the weight of the particular class. It is calculated as the sum of cognitive weights of its q linear blocks composed in individual Basic Control

Structures (BCS's). Each block consists of 'm' layers of nested Bas BCS's and each layer has 'n' linear BCS. There are some disadvantages of the CC. A well-defined metric not only considers the number of methods, classes, subclasses and relation. It also considers the internal structure of the method.

Sanjay Misra [9] modified the Class Complexity metric and proposed a new metric called Weighted Class Complexity (WCC) in eq. (4). The complexity of a method is calculated by complexity of the code of operation in the method and as well as on the number of attributes in the method.

$$WCC = N_a + \sum_{p=1}^{s} MC_p \quad (4)$$

Where N_a stands for the total number of attributes and MC is the method complexity. The main drawback of the WCC is not considering the object-oriented features of a class.

Arockiam et al. [10] extended WCC as Extended Weighted Class Complexity (EWCC) which contains one of the OO feature namely the cognitive complexity due to Inheritance is calculated in eq. (5) as follows:

$$EWCC = N_a + \sum_{i=1}^{n} MC_i + \sum_{j=1}^{m} ICC_j \quad (5)$$

Where N_a is the total number of attributes, MC is the method complexity, ICC is the inherited class complexity.

$$ICC = (DIT \times C_L) \times \sum_{k=1}^{s} RMC_k + RN_a \quad (5.1)$$

Where s is the number of inherited methods, RNa is the total number of Reused attributes, RMC is the Reused Method Complexity.

C_L is the cognitive complexity of L^{th} level which will differ from person to person. The value of C_L is assumed to be 1. The limitation of EWCC is cognitive load for L^{th} level inheritance which is not clearly defined. It needs to be well defined and more specific for the inheritance level.

Aloysius altered EWCC as AWCC (Attribute Weighted Class Complexity) [11]. In EWCC, every attribute has the same cognitive weight (value) but in general, cognitive load in understanding the different types of attributes cannot be the same. Hence, a new metric namely AWCC was proposed in eq. (6). In AWCC, the cognitive weights were assigned for the attributes based on the effort needed to understand the different data types. Table 1 shows the weights of different attributes based on cognitive aspects described by Wang [23].

$$AWCC = \sum_{i=1}^{n} AC_i + \sum_{j=1}^{m} MC_j + \sum_{k=1}^{m_1} ICC_k \quad (6)$$

$$AC = (PDT * W_b) + (DDT * W_d) + (UDDT * W_u) \quad (6.1)$$

Where PDT is the number of Primary Data Type attributes, DDT is the number of Derived Data Type attributes, UDDT is the number of User Defined Data Type attributes.

TABLE I

COGNITIVE WEIGHTS OF DIFFERENT TYPES OF ATTRIBUTES

Category	Weights
Sub-Conscious Cognitive Attribute (PDT)	1
Meta Cognitive Attribute (DDT)	2
Higher Cognitive Attribute (UDDT)	3

AWCC only focuses the data type (attribute) complexities. It does not consider other parameters of a class.

Sandip Mal et al. suggested a new complexity metric for Object-Oriented (OO) design to measure the complexity of a software system [12]. Complexity Metric (CM) of a class is defined in the following eq. (7).

CM= NOM + INST + EXT + NSUP + TCC + NSUB (7)

Where NOM = No. of methods in a class, INST = No. of instance variables in a class, EXT = No. of external variable in a class, NSUP = No. of super class of a class, TCC = Total Cyclomatic Complexity of a Class, NSUB = Number of subclass of a class.

Higher value of CM indicates more mental exercise is required to design and code the class and vice versa. This metric is used for predicting the possibility of reuse a class in a large system, how much easy to understand and how much complex the design and more empirical validation is needed.

Vinay et al. proposed a new class complexity metric CCC (Complete Class Complexity metric). It focuses each dimension of a class (Nine different parameters) to measure the complexity which is represented in eq. (8) [27].

CCC = NOMT + AVCC + MOA + EXT + NSUP+ NSUB + INTR + PACK + NQU (8)

Where NOMT - Number of Methods, AVCC - Average Cyclomatic Complexity, MOA - Measure of aggregation

EXT - External Method calls, NSUP - Number of Super Class, NSUB - Number of Sub Class, INTR - Interface Implemented, PACK - Package Imported, NQU - Number of Queries.

The CCC metric involves all the possible attribute of the class and is predictor of how much time and effort is required to design and maintain the class. The value of the CCC metric is directly linked with the testability and understandability of the class. The increased the value of CCC metric shows that more the effort needed to maintain the class.

Kumar Rajnish suggested a new Class Complexity Metric (CCM) of an Object-Oriented (OO) program. To calculate CCM, Total Cyclomatic Complexity (TCC) of a class, Number of Methods (NOMT) of a class, Number of Instance Variables (INST) declared, Number of External Methods (EXT) called, Number of Local Methods (LMC) called, and Total Lines of Code (NLOC) have been taken [28]. The formula for CCM is written in eq. (9).

CCM = k +w1 * TCC + w2 * NOMT + w_3 * INST+ w4 * EXT + w5 * LMC + w_6 * NLOC (9)

Where w_1, w_2, w_3, w_4, w_5, w_6 were the weights and the constant k are derived at least square regression analysis. CCM is used to predict the understandability of classes. It does not concentrate performance indicators such as maintenance effort and System performance.

Kumar Rajnish presented [26] a new complexity metric for OO design measurement to calculate the design stage whether the classes become more complex, Moderate complex or less complex. A metric named Attribute Method Complexity (AMC) is used to measure the complexity of class in the design stage itself. AMC may be defined as follows:

AMC = A' + M' (10)

In eq. (10), A' and M' represents the attributes and method range values based on the sum of the actual attributes and methods (private, public and protected) of a class. In large applications, AMC is used to predict how much effort would be required to reuse a system, how much easy to understand and how much complex the design. AMC measures reusability, understandability and complex design. The higher number of classes denotes the lower reusability. Higher design complex harder to understand.

B. CODE/PROGRAM LEVEL COMPLEXITY METRIC

Wang et al., [13] proposed complexity measure of a software program named as Cognitive Functional Size (CFS) in eq. (11). The functional size of software depends on three parameters input, output and internal control flow. The internal control flow of a program derived from Basic Control Structures.

$$CFS = (N_i + N_o) * W_c \quad (11)$$

Where N_i is the number of inputs to the program and N_o is the number of output from the program and W_C is the entire cognitive weight of all BCSs.

$$W_c = \sum_{j=1}^{q} [\prod_{k=1}^{m} \sum_{i=1}^{n} W_c(j,k,i)] \quad (11.1)$$

Kushwaha [14] defines Cognitive Information Cognitive Measure (CICM) is the function of operators and operands.

$$CICM = WICS * W_c \quad (12)$$

In eq. (12), W_c is the weight of the BCS and WICS stands for Weighted Information Count of Software which is defined in eq. (12.1)

$$WICS = \sum_{k=1}^{LOCS} WICL_k \quad (12.1)$$

WICL means weighted Information Count of k^{th} LOC.

$$WICL_k = ICS_k / [LOCs - k] \quad (12.2)$$

ICS stands for Information Contained in Software.

$$ICS = \sum_{k=1}^{LOCS} I_k \quad (12.3)$$

Misra developed [15] a new metric named as Cognitive Program Complexity Measure (CPCM) is expressed that the total number of occurrences of input and output is strongly effect of the cognitive complexity of software. The CPCM is defined in eq. (13).

$$CPCM = SIO + W_c \quad (13)$$

$$SIO = N_i + N_o \quad (13.1)$$

Where N_i is total occurrence of input variables and N_o is total occurrence of output variables.

Amit Kumar [16] proposed a metric to count the number of variables and constants line by line and multiplies it with its BCS's weight. This technique is used to measure the cognitive complexity of a program. In this measure operators are not considered. The New Cognitive Complexity of Program (NCCoP) is calculated in following eq. (14).

$$\text{NCCoP} = \sum_{k=1}^{LOCs} \sum_{v=1}^{LOCs} N_v * W_c(k) \quad (14)$$

C. INHERITANCE AND POLYMORPHISM BASED METRICS

Inheritance and Polymorphism are the important concept of OO programming paradigm. The following identifies some the existing metrics for the same. Sanjay Misra [17] suggested that an inheritance complexity metric for object-oriented code using cognitive approach. In this measure first calculate Method Complexity. Second stage contains calculation of class complexity. The third important stage is to find Cognitive Code Complexity (CCC) calculated eq. (15)

$$\text{CCC} = \prod_{j=1}^{m}\left[\sum_{k=1}^{n} CC_{jk}\right] \quad (15)$$

If more than one class hierarchies, then we simply add CCCs of each hierarchy to calculate the complexity of the whole OO system. The Class Complexity Unit (CCU) of a class is defined as the cognitive weight of the simplest software component.

Deepti Misra [18] proposed two complexity metrics for inheritance, one at class level CCI expanded as Class Complexity due to Inheritance is represented in eq. (16) and another metric for program level ACI stands for Average Complexity of a program due to Inheritance is defined in eq. (16.2).

$$\text{CCI}_i = \sum_{ifrom=1}^{k} CCI_i\, from + \sum_{j=1}^{l} MC_j \quad (16)$$

Where P denotes the number of predicates in a method and D is the maximum depth of control structures in a method

$$\text{Method Complexity (MC)} = P + D + 1 \quad (16.1)$$

Where $\text{CCI}_{\text{ifrom}}$ is the complexity of an ith class is due to inheritance, 'k' is the number of classes, ith class is inheriting, $\text{CCI}_{\text{ifrom}}$ is the complexity of a parent class, i^{th} class is inheriting, '*l*' is the number of methods in i^{th} class, MC_j is the complexity of j^{th} method in i^{th} class is calculated using eq. (16.1) . ACI defined as follows:

$$\text{ACI} = \frac{\sum_{i=1}^{n} \text{CCI}_i}{n} \quad (16.2)$$

Ankita et al., proposed a measure for design complexity of different types of inheritance after analysing various Cohesion metrics. Based on the results the author suggested that hierarchical inheritance should be avoided and suggested to use Interfaces for better design [19].

Sheldon et al., proposed [24] two metrics Average Degree of Understandability (AU) and Average Degree of Modifiability (AM). AU is defined as in eq. (17).

$$\text{AU of a class inheritance} = \sum_{i=1}^{t}(\text{PRED(Ci)} + 1) / t \quad (17)$$

$$\text{U of class } C = \text{PRED}(C_i) + 1 \quad (17.1)$$

Where C_i is i^{th} class, PRED (C_i) is the total numbers of predecessors of class i, 't' is the total number of classes in the class inheritance tree, and SUCC (C_i) stands for successors of class i. Average Degree of Modifiability (AM) is defined as follows:

$$\text{AM} = \text{AU} + \left(\sum_{i=1}^{t}(SUCC(C_i)/2)\right)/t \quad (17.2)$$

Rajnish et al., proposed three different metrics for class inheritance hierarchies [20]. Derive Base Ratio Metric (DBRM), Average Number of Direct Child (ANDC) Metric and Average Number of Indirect Child (ANIC) Metric for measure the complexity in design stage itself. DBRM is the ratio of the total number of derived classes (TD) to the total number of base classes (TB) in the class inheritance tree. DBRM is calculated in eq. (18).

$$\text{DBRM} = \left(\sum_{i=0}^{N} TD(C_i)\right)/\left(\sum_{i=0}^{N} TB(C_i)\right) \quad (18)$$

ANDC metric is the ratio of the total number of immediate child (TDC) to the total number of classes (N) in the inheritance tree which is calculated as follows:

$$\text{ANDC} = \sum_{i=0}^{N}(TDC(C_i))/N \quad (18.1)$$

ANIC metric is the ratio of the total number of indirect child TIC) to the total number of classes (N) in the inheritance tree. ANIC metric is as follows:

$$\text{ANIC}=\sum_{i=0}^{N}(TIC(C_i))/N \quad (18.2)$$

Abreu et al. proposed Metrics [22, 25] for Object Oriented Design (MOOD). It refers set of metrics for the object-oriented paradigm. The two metrics Method Hiding Factor (MHF) and Attribute Hiding Factor (AHF) were used together to measure the feature encapsulation. AHF and MHF correspond to the average amount of hiding between all classes in the system. Method Inheritance Factor (MIF) and Attribute Inheritance Factor (AIF) are used to measure complexity of an inheritance. The other MOOD metrics are Coupling Factor (COF), Polymorphism Factor (POF), Clustering Factor and Reuse Factor. The two main features used in MOOD metrics are methods and attributes. Methods are used to perform operations of several kinds such as obtaining or modifying the status of objects.

Francis proposed a new cognitive complexity metric called Cognitive Weighted Polymorphism Factor (CWPF) to identify the complexity of polymorphism on the basis of its types namely pure, static and dynamic polymorphism. It calculates not only the architectural complexity of the polymorphism, but also the cognitive complexity arising from the effort needed to comprehend different types of polymorphism in the Object-Oriented software system [21].

$$CWPF = \frac{\sum_{i=1}^{TC} CWM_o(C_i)}{\sum_{i=1}^{TC}[M_n(C_i)*DC(C_i)]*ACW} \quad (19)$$

$$CWM_o(C_i) = N_{PP} * CW_{PP} + N_{SP} * CW_{SP} + N_{DP} * CW_{DP}, \mathrm{M_n(C_i)} \quad (19.1)$$

In eq. (19), M_n (C_i) represents number of overriding methods in class C_i, DC (C_i) means number of children for class C_i, TC stands for Total number of Classes. N_{PP} abbreviate number of pure polymorphism, N_{SP} stands for number of static polymorphism, N_{DP} = number of dynamic polymorphism, CW means cognitive weight. ACW is defined in eq. (19.2)

$$\text{ACW} = (CW_{PP} + CW_{SP} + CW_{DP}) / 3 \quad (19.2)$$

CWPF is more comprehensive in nature and the metrics were verified by case study by conducting a set of comprehension test. It is concluded that the CWPF is a better indicator for calculating class complexity than the existing PF metric.

Francis [21] introduced a new complexity metric called Cognitive Weighted Attribute Hiding Factor for finding class complexity due to encapsulation is calculated in eq. (20).

$$CWAHF = \frac{\sum_{i=1}^{TC} A_h(C_i)}{\sum_{i=1}^{TC} A_h(C_i) + \sum_{i=1}^{TC} A_v(C_i)} \quad (20)$$

where,

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

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

$A_p(C_i)$ = Number of private attributes in class C_i
$A_d(C_i)$ = Number of default attributes in class C_i
$A_t(C_i)$ = Number of protected attributes in class C_i
$A_u(C_i)$ = Number of public attributes in class C_i
CW_{pa} = Cognitive Weight of private attribute
CW_{da} = Cognitive Weight of default attribute
CW_{ta} = Cognitive Weight of protected attribute
CW_{ua} = Cognitive Weight of public attribute
TC = Total number of Classes in the whole system

CWAHF was defined mathematically and it is a indicator of class complexity, due to the encapsulation and attributes scopes, than the AHF proposed by Abreu [22].

IV. COMPARATIVE ANALYSIS

This paper addresses the various metrics both cognitive and non-cognitive approach with respect to class, code, inheritance, polymorphism and encapsulation. The comparative analysis of these metrics is shown in Table II.

Some of the observations from the existing metrics are listed below for further enhancement.

i. The inheritance complexity metric with all existing limitations can be improved for a better calculation of class complexity based on cognitive phenomenon.
ii. Method complexity (MC) plays a vital role for computing the class complexity. Therefore, the redefined MC by incorporating the different parameters of methods including cognitive approach could be developed.

iii. The cognitive class complexity metric by including the different access specifiers of the class attributes and methods could be proposed.

iv. A cognitive complexity metrics for advanced object oriented concepts like interface, packages, etc., can be designed.

v. An unified or integrated metric with the help of existing metrics by incorporating all the major features of OOPs including cognitive aspects need to be developed.

Table II. Comparison of various OO Metrics

Measure / Metric	Class Complexity	Code Complexity	Inheritance	Polymorphism/ Encapsulation	Cognitive
WMC	✓	-	-	-	-
CC (Bala)	✓	-	-	-	-
CC (Misra)	✓	-	-	-	✓
WCC	✓	-	-	-	✓
EWCC	✓	-	✓	-	✓
AWCC	✓	-	✓	-	✓
CM	✓	-	-	-	-
CCC (Vinay)	✓	-	-	-	-
CCM	✓	-	-	-	-
AMC	✓	-	-	-	-
CFS	-	✓	-	-	✓
CICM	-	✓	-	-	✓
CPCM	-	✓	-	-	✓
NCCoP	-	✓	-	-	✓
CCC (Misra)	-	-	✓	-	✓
CCI	-	-	✓	-	
AM	-	-	✓	-	✓
MHF & AHF	-	-	-	✓	-
MIF & AIF	-	-	✓	-	-
COF & POF	-	-	-	✓	-
DBRM , ANDC ANIC	-	-	✓	-	-
CWPF	-	-	-	✓	✓
CWAHF	-	-	-	✓	✓

V. CONCLUSION

Software metrics are used by the project manager, developer, and tester in order to ensure the quality of the software products. The primary objective of this paper is to investigate the various object oriented metrics both cognitive and non-cognitive approach. The class level, program or code level, polymorphism, encapsulation and inheritance aspects of complexity metrics are analysed and tabulated. From the existing literature, some of the observations and future directions are also discussed.

VI. REFERENCES

[1]. N. Fenton and S.L. Pfleeger, "Software Metrics: A Rigorous & Practical Approach", Third edition, International Thomson Computer Press, 2014.

[2]. M. J. Shepperd and D. Ince, "Derivation and Validation of Software Metrics", Oxford University Press, 1993.

[3]. Halstead, Elements of Software Science, New York: Elsevier North, 1977.

[4]. Kafura D and Henry S, Software Quality Metrics Based on Interconnectivity, Journal of Systems and Software, Vol. 2, No. 2, pp 121-131, 1981,

[5]. McCabe, A Complexity Measure, IEEE Trans. on Software Engg., Vol. 2, No. 4, pp.308-320, 1976.

[6]. S.R. Chidamber and C.F. Kemerer, A metrics suite for object oriented design, IEEE Transactions on Software Engineering, pp. 476-493, Vol. 20, No.6, 1994.

[7]. N.V. Bala Subramanian, "Object-Oriented Metrics", Asian Pacific Software Engineering Conference (APSEC-96), IEEE Xplore, pp. 30-34, 1996.

[8]. Sanjay Misra and Ibrahim Akman, "A New Complexity Metric Based on Cognitive Informatics", Proceedings of 3rd International Conference on Rough Sets and Knowledge Technology, pp.620–627, 2008.

[9]. Sanjay Misra and k. Ibrahim Akman, "Weighted Class Complexity: A Measure of Complexity for Object Oriented System," Journal of Information Science and Engineering, pp. 1689-1708, 2008.

[10]. L. Arockiam, A. Aloysius, J. Charles selvaraj, "Extended Weighted Class Complexity: A new measure of software complexity for objected

oriented systems", Proceedings of International Conference on Semantic E-business and Enterprise computing (SEEC), pp. 77-80, 2009.

[11]. L. Arockiam, A. Aloysius, "Attribute Weighted Class Complexity: A New Metric For Measuring Cognitive Complexity Of OO Systems", Proceedings of International Conference on Computational Intelligence and Cognitive Informatics, 2011.

[12]. Sandip Mal and Kumar Rajnish, "Measuring System Complexity Using New Complexity Metric", Software Engineering: An International Journal, Vol. 3, No. 2, pp.35-43, 2013.

[13]. Y. Wang and J. Shao, "Measurement of the Cognitive Functional Complexity of Software", The 2nd IEEE International Conference on Cognitive Informatics, IEEE CS Press, pp. 67-74, 2003.

[14]. D. S. Kushwaha and A. K Misra, "Robustness Analysis of Cognitive Information Complexity Measure using Weyuker Properties", ACM SIGSOFT SEN, Vol. 31, No. 1, 2006.

[15]. Sanjay Misra, "Cognitive Program Complexity Measure", Proceedings of 6th IEEE International Conference on Cognitive Informatics, 2007.

[16]. Amit Kumar, "A New Cognitive Approach to Measure the Complexity of Software's", International Journal of Software Engineering and Its Applications Vol.8, No.7, pp.185-198, 2014.

[17]. Sanjay Misra, "An inheritance complexity metric for object-oriented code: A cognitive approach", Indian Academy of Sciences, Vol. 36, No. 3, pp. 317–337, 2011.

[18]. Deepti Mishra, "New Inheritance Complexity Metrics for Object-Oriented Software Systems: An Evaluation with Weyuker's Properties", Computing and Informatics, Vol. 30, No.2, pp. 267–293, 2011.

[19]. Ankita Mann, Sandeep Dalal and Neetu Dabas, "Measurement of Design Complexity of Different types of Inheritance using Cohesion Metrics", International Journal of Computer Applications, Vol. 77, No.3, 2013.

[20]. Kumar Rajnish, Arbind Kumar Choudhary, Anand Mohan Agrawal, "Inheritance Metrics for Object-Oriented Design", International Journal of Computer Science & Information Technology (IJCSIT), Vol. 2, No. 6, pp. 13-26, 2010.

[21]. Francis Thamburaj, Aloysius A, "Cognitive Weighted Polymorphism Factor: A Comprehension Augmented Complexity Metric", International Journal of Computer, Electrical, Automation, Control and Information Engineering Vol. 9, No. 11, pp.2307-2312, 2015.

[22]. F. B. Abreu, and R. Carapuça., "Object-oriented software engineering: Measuring and controlling the development process," Proceedings of the 4th international conference on software quality, pp. 1-8, 1994.

[23]. Wang. Y, "A new measure of software complexity based on cognitive weights", Canadian Journal of Electrical and Computer Engineering, Vol. 28, No. 2, pp. 1-6, 2003.

[24]. Sheldon T. F, Jerath. K and Chung. H," Metrics for maintainability of class inheritance hierarchies", Journal of software maintenance and evolution: Research and practice, Vol. 14, No. 3, pp. 147-160, 2002.

[25]. Abreu, Fernando B, "Design metrics for OO software system", ECOOP'95, Quantitative Methods Workshop, 1995.

[26]. Kumar Rajnish, "Another New Complexity Metric for Object-Oriented Design Measurement", International Journal of Hybrid Information Technology, Vol.7, No.2, pp.203-216, 2014.

[27]. Vinay Singh, Vandana Bhattacherjee, "A New Complete Class Complexity Metric", International Journal of Soft Computing And Software Engineering (JSCSE), Vol. 3, No. 9, pp.1-9, 2013.

[28]. Kumar Rajnish, "Class Complexity Metric to Predict Understandability", International Journal of Information Engineering and Electronic Business, Vol. 6, No. 1, pp. 69-76, 2014.