A Survey on Metric of Software Cognitive Complexity for OO design

[1]A.Aloysius, [2]L. Arockiam

***Abstract*—**In modern era, the biggest challenge facing the software industry is the upcoming of new technologies. So, the software engineers are gearing up themselves to meet and manage change in large software system. Also they find it difficult to deal with software cognitive complexities. In the last few years many metrics were proposed to measure the cognitive complexity of software. This paper aims at a comprehensive survey of the metric of software cognitive complexity. Some classic and efficient software cognitive complexity metrics, such as Class Complexity (CC), Weighted Class Complexity (WCC), Extended Weighted Class Complexity (EWCC), Class Complexity due to Inheritance (CCI) and Average Complexity of a program due to Inheritance (ACI), are discussed and analyzed. The comparison and the relationship of these metrics of software complexity are also presented.

***Keywords*—**Software Metrics, Software Complexity, Cognitive Informatics, Cognitive Complexity, Software measurement

I. Introduction

NUMEROUS term software metrics related to software quality assurance have been proposed in the past and are still being proposed. Several books presenting such metrics exist, such as Fenton's [1], Shepperd's [2] and others. Although most of these metrics are applicable to all programming languages, some metrics apply to a specific set of programming languages. Among these metrics some have been proposed based on cognitive complexity called Cognitive Complexity Metrics. Wang[3] observed that the traditional measurements cannot actually reflect the real complexity of software systems in software design, representation, cognition, comprehension, and maintenance. The cognitive complexity is an ideal measure of software functional complexities and sizes, because it represents the real semantic complexity by integrating both the operational and architectural complexities.

Nowadays, a quality engineer can choose a suitable metrics for the software development process from a large number of Cognitive Complexity Metrics. Now, the question posed is the lack of metric suite and the selection of those metrics which meet the specific needs of each software project.

The metrics presented in this survey are for Object–oriented programming. There are a number of 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. Still there are other Object Oriented Cognitive Complexity Metrics specially developed for Object Oriented programs.

Cognitive Informatics [6] also plays an important role in understanding the fundamental characteristics of the software. Many software complexity measures [4] [5] and [6] based on cognitive informatics have been proposed in the last decade. Cognitive complexity metrics measure the human effort needed to perform a task or difficulty in understanding the software. In cognitive informatics, it has been found that the functional complexity of the program depends on internal architecture of the software, input and output [4].

The aim of this survey is to list out some of the existing Cognitive Complexity Metrics and to make the reader aware of their existence and to offer references for further reading. The entire paper has been segregated into four major sections. In section 2, the Definition and the Classification of Software Complexity metric are introduced. The Classic Software complexity metrics and their variations are analyzed in section 3. Finally, we make the conclusion in section 6.

II. Definition And The Classification Of Software Complexity

A. Software Metrics

The software metric is the measurement, usually using numerical ratings, to quantify some characteristics or attributes of a software entity. Typical measurements include the quality of the source codes, the development process and the accomplished applications. Some major measurements are listed in table 1.

TABLE I
DIFFERENT MEASUREMENTS IN TERMS OF DIFFERENT ROLES

Role	Measurements
User	Usability, Simplicity, Stability, Cost…
Designer	Extendibility, Scalability, Manageability…
Programmer	Complexity, Maintainability…

B. The Metric of Software Complexity

The metric of the software complexity is an essential and critical part of the software metric. The metric of the software complexity focuses on the quality of source codes. The complexity of software can be divided into three classes: the essential complexity, the selecting complexity and the

A. Aloysius, Assistant Professor, Department of Computer Science, St. Joseph's College (Autonomous), Tiruchirappalli – 620 002, Tamil Nadu, India(Mobile: 9443399227; e-mail: aloysius1972@gmail.com).

Dr. L. Arockiam, Associate Professor, Department of Computer Science, St. Joseph's College (Autonomous), Tiruchirappalli – 620 002, Tamil Nadu, India(Mobile: 94439 05333; e-mail: larockiam@yahoo.co.in).

incidental complexity [7]. The essential complexity is determined by the problems that the software tries to solve. The selecting complexity is determined by the program languages, the problem modeling methods and the software design methods. The incidental complexity is determined by the quality of the involved implementer. Some information about them is shown in the table 2.

TABLE II
THE CLASSIFICATION OF THE COMPLEXITY

Class	Origin	Influence	Improvement
Essential	Problem	Huge	Extreme Hard
Selecting	Method	Medium	Hard
Incidental	Implement	Small	Fair

The target of measuring the software complexity is to manage and reduce the incidental complexity, and improve the development of the software and the software itself.

C. Cognitive Informatics

Cognitive informatics is a trans-disciplinary enquiry of cognitive and information sciences that investigates the internal information processing mechanisms and processes of the brain and natural intelligence, and their engineering applications via an interdisciplinary approach [3]. In CI, the cognitive information and knowledge modeled in the brain can be divided into different abstraction levels, such as analogue objects, natural languages, professional notation systems, mathematics, and philosophies. A classification of the cognitive abstract levels is shown in table 3. In order to identify, whether a given object can be categorized as a type of information in the abstract world, or a type of an entity in the physical world, table 3 would facilitate. According to the classification of abstract levels of cognitive information in table 3, it can be seen that software is a kind of information at abstract levels 3 or 4, rather than a physical entity.

TABLE III
ABSTRACT LEVELS OF COGNITIVE INFORMATION

Level	Category		Description
1	Analogue objects		Empirical artifacts
2	Natural languages		Empirical methods, heuristic rules
3	Special notation systems		Professional languages, formal methods
4	Mathematics		High-level abstraction of objects, attributes, and their relations and rules, particular those that are time and space independent.
	4.1	Formulae	Mathematical description of relations or rules
	4.2	Theorems	Proved relations or rules
	4.3	Corollaries	Derived conclusions based of relations or rules
5	Philosophies		The highest-order abstraction of generic objects and their relations and rules, particular those that are time and space independent.

According to Wang the major problems yet to be solved in CI are: the architectures of the brain, mechanisms of the natural intelligence, cognitive processes, mental phenomena and personality. It is interesting in computing and software engineering arena to explain the mechanisms and processes of memory, learning and thinking. It is expected that any breakthrough in CI will profoundly pave the way to the development of the next generation technologies in informatics, computing, software, and cognitive sciences.

D. Classifications of the metrics of software cognitive complexity

Before discussing the details of the software cognitive complexity metrics, we can classify the metrics of software cognitive complexity by what they are calculated on. The classification of the 5 different metrics is shown in the table 4.

TABLE IV
THE CLASSIFICATION OF THE COGNITIVE COMPLEXITY METRICS BY THEIR CALCULATION BASIS

Metrics	Target			
	Method	Attribute	Inheritance	Average
CC	*			
WCC	*	*		
EWCC	*	*	*	
CCI	*		*	
ACI	*		*	*

III. Classic Metrics Of Software Complexity And Variations

A. Class Complexity(CC)

a. Definition and computing of CC

Mishra [9,10] proposed a complexity metric for Object Oriented system by using cognitive weights known as Class Complexity(CC). This metric first calculates the weight of individual method in a class by associating a number (weight) with each member function (method), and then it simply adds all the weights of all methods. This gives the complexity (weight) of a single class.

B. Some Findings

There are two cases for calculating the whole complexity of the entire system depending on the architecture:

- If the classes are in the same level then their weights are added.
- If they are subclasses or children of their parent classes then their weights are multiplied (Inheritance).

If there are *m* level of depth in the OO code and level *j* has *n* classes then the class complexity (CC) of the system is given by equation (1),

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

where, W_c is the weight of the concerned class. The weight of a single method is given by equation (2)

$$W_c = \sum_{j=1}^{q} \left[\prod_{k=1}^{m} \sum_{i=1}^{n} W_c\,(j,k,i) \right] \qquad (2)$$

Where, total cognitive weight of a software component W_c is defined by Wang[9] as the sum of cognitive weights of its q linear blocks composed in individual BCS's. Each block consists of m layers of nested BCS's and each layer has n linear BCS. A higher weight indicates a higher level of effort required to understand the software and reduced maintainability.

The CC is validated using Weyuker's properties and principles of measurement theory and found that the proposed measure satisfies seven Weyuker's property out of nine, and satisfied most of the parameters required by the perspective of measurement theory. As a result, this measure was established as well structured one.

c. Advantages and Disadvantages of CC

In CC Calculation of the complexity is very simple, understandability of the code is easy and language independent. Low complexity value gives better design information and the outcome of CC is also good. However, there are some shortages of the CC. A good metric should not only consider the number of methods, classes, subclasses and relation between them but also consider the internal structure of the method.

C. Weighted Class Complexity (WCC)

3.2.1Definition and computing of WCC

Mishra [11] modified the CC metric and proposed a new metric called weighted class complexity (WCC). It considered object orientation as a form of expression relation between the data and function, the class can be assumed as a set of data and set of method accessing them. Hence, the complexity of the class should be measured by complexity of methods and attributes. In his proposed measure, the complexity of a class was the sum of complexity of the operation in methods, complexity due to data members (attributes) and complexity due to message call. Further, complexity of method is calculated by complexity of the code of operation in method and as well as on the number of attributes in the method. Weighted Class Complexity (WCC) can be calculated using the equation (3)

$$\mathrm{WCC} = \mathrm{N_a} + \sum_{p=1}^{s} \mathrm{MC_p} \qquad (3)$$

where,

N_a is the Number of Attribute

MC is the Method Complexity, which is calculated by equation (2)

If there are y classes in an object oriented code, then the total complexity of the code is given by the sum of weights of individual classes.

$$\text{Total Weighted Class Complexity} = \sum_{x=1}^{y} \mathrm{WCC_x} \qquad (4)$$

b. Advantages and Disadvantages of WCC

The drawback of CC has been rectified in WCC by including the complexity due to the internal structure of methods and attributes. By using, Weyuker's [12] properties WCC has been validated and found that it satisfies six properties out of nine, which established this measure as well structured one. WCC can be used to calculate the complexity of OO code with different size. WCC metric gives valuable idea about the design quality of object oriented codes. High WCC value indicates that understandability and maintainability of the code is difficult. Ultimately, it helps the software developer for better design information. The better OO metric should not only consider the internal structure of method and the number of attributes in a class but it should also consider the concepts of OOP like inheritance, encapsulation, overloading and polymorphism.

C.Extended Weighted Class Complexity (EWCC)

a. Definition and computing of EWCC

Arockiam et al. [13] have proposed a new complexity measure namely Extended Weighted Class Complexity (EWCC) which is an extension of Weighted Class Complexity (WCC). EWCC is the sum of cognitive weights of attributes and methods of the class and that of the classes derived. EWCC includes the cognitive complexity due to Inheritance.

If there are n methods in a class and the class is derived from m no of classes then, the EWCC of that class can be derived using the Equation (5)

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

Where

N_a is the total number of attributes,

MC is the method complexity,

ICC is the inherited class complexity.

The Method complexity (MC)is calculated by equation (1) and ICC can be calculated using the Equation (6)

$$\mathrm{ICC} = (\mathrm{DIT} \times \mathrm{C_L}) \times \sum_{k=1}^{s} \mathrm{RMC_k} + \mathrm{RN_a} \qquad (6)$$

Where **s** is the no of inherited methods.

RNa is the total number of Reused attributes,

RMC is the Reused method complexity.

ICC is the inherited class complexity.

DIT is the Depth of Inheritance Tree

CL is the cognitive complexity of Lth level

CL is the cognitive complexity of Lth level which will differ from person to person according to the cognitive maturity level [4]. Here, the value of CL is assumed to be 1.

b. Advantages and Disadvantages of EWCC

Arockiam et al. [14] have validated Extended Weighted Class Complexity (EWCC) and other complexity metrics with respect to program comprehension. In EWCC, the complexity of the class included the internal complexity of the class and the inherited classes' complexity. It also includes the cognitive complexity due to internal architecture of the methods, attributes and the inherited class complexity. On one hand, this makes EWCC to be a better indicator of the class level complexity metric. On the other hand, EWCC has some limitations. Cognitive Load has been assigned for Lth level inheritance which is not clearly defined, it simply assigned the value is 1 so, it needs to be well defined and more specific for the inheritance level.

D. Class Complexity due to inheritance (CCI)

a. Definition and computing of CCI

Mishra[15] has proposed a Class complexity due to inheritance (CCI) which can be calculated as:

$$\mathrm{CCI_i} = \sum_{\mathrm{i_{form}}=1}^{\mathrm{k}} \mathrm{C_{i_{form}}} + \sum_{\mathrm{j}=1}^{\mathrm{l}} \mathrm{MC_j} \qquad (7)$$

Where, CCIi is complexity of a ith class due to inheritance, k is the number of classes ith class is inheriting directly, Ciform is the complexity of an inherited class, l is the number of methods excluding constructors, destructors, pure virtual functions ith class has, MCj is the complexity of jth method in ith class and can be calculated using new proposed method complexity metric (MC).

$$\text{Method Complexity (MC)} = \mathrm{P} + \mathrm{D} + 1 \qquad (8)$$

Where, P is the number of predicates, D is the maximum depth of control structures in method; if there is no nested control structures than D=0; if there is one inside another than D=1

b. Advantages and Disadvantages of CCI

CCI considers the number of classes inherited in an inheritance tree. Inheritance metrics should not only consider the number of classes a particular class is inheriting but also the complexities of the inherited classes. Constructors and destructors are not considered in these calculations as they are not inherited. Complexities of inherited classes as well as complexity of the derived class are all taken into consideration in CCI. The average complexity of the Inherited classes are not consider in CCI, this is consider to be one of the limitation of CCI.

E. Average Complexity of a Program due to inheritance (ACI)

a. Definition and computing of ACI

Average Complexity of a program due to Inheritance (ACI)[15] is calculated by equation (9)

$$ACI = \sum_{i=1}^{n} CCI_i \Big/ n \qquad (9)$$

where, n is the total number of classes in the program. CCI is the Class Complexity due to inheritance, which can be calculated by using the equation (7).

b. Advantages and Disadvantages of ACI

ACI considers the average number of classes inherited in an inheritance tree by considering how complex the inherited classes are or how many methods are inherited or how complex inherited methods are. Inheritance metrics should not only consider the number of classes a particular class is inheriting but also the complexities of the inherited classes. Constructors and destructors are not considered in these calculations as they are not inherited. If inheritance metric considers only the number of classes inherited, then it gives value 1 for all the class. But if the number of classes inherited, complexities of inherited classes as well as complexity of the derived class are all taken into consideration the value changes.

IV. Conclusion

This survey presents various Cognitive Complexity Metrics which act as a base of Cognitive Complexity Metrics. The metrics are classified as Cognitive Complexity Metrics for OOP, such as Class Complexity (CC), Weighted Class Complexity (WCC), Extended Weighted Class Complexity (EWCC), Class Complexity due to Inheritance (CCI) and Average Complexity of a program due to Inheritance (ACI) are discussed in detail. From this survey, it can be observed that this Cognitive Complexity Metrics is an emerging field in the software complexity metrics. Most of the metrics presented here are not yet empirically evaluated. There are a few metric proposed for OOP, which does not take care of all the features in it. Future scope of improvement in these areas could be that a Cognitive Complexity Metrics can be proposed to include the other OOP features like polymorphism, cohesion and coupling. A unified Cognitive Complexity Metrics for OOPs can be empirically validated. The current trend in IT industry is moving towards Component Based System, because of its various advantages where Cognitive Complexity Metrics is lacking. The researchers can propose Cognitive Complexity Metrics for Component Based System. The result of this survey can be of great assistance to quality engineers in selecting the proper metrics for their software projects.

References

[1] N. Fenton & S.L. Pfleeger, “Software Metrics: A Rigorous & Practical Approach”, Second edition, International Thomson Computer Press, 1997, ISBN-10: 0534954251, ISBN-13: 978-0534954253
[2] M.J. Shepperd and D. Ince, “Derivation and Validation of Software Metrics”, Oxford University Press, USA, 1993, ISBN-10: 0198538421, ISBN-13: 978-0198538424.
[3] Y. Wang, “The Theoretical Framework of Cognitive Informatics”, International Journal of Cognitive Informatics and Natural Intelligence, 2007, pp.1–27.
[4] Y. Wang, "On Cognitive Informatics", Proceedings of the 1st IEEE International Conference on Cognitive Informatics, 2002, pp.34-42.
[5] T. Klemola and J. Rilling, "A Cognitive complexity metric based on Category learning", Proceedings of the 2nd IEEE International Conference on Cognitive Informatics (ICCI'03), 2003, pp.103-108.
[6] Y. Wang. and J. Shao, "Measurement of the Cognitive Functional Complexity of Software," Proceedings of IEEE (ICCI'03), 2003, pp.69-74.
[7] Eric S. Raymond, “The Art of Unix Programming”, Addison-Wesley, 2004,pp. 320-350.
[8] Sanjay Misra, “An Object Oriented Complexity Metric Based On Cognitive Weights”, Proceedings of 6th IEEE International Conference on Cognitive Informatics(ICCI’07), 2007, pp. 134-139.
[9] Sanjay Misra and Ibrahim Akman, “A New Complexity Metric Based on Cognitive Informatics”, Proceedings of 3rd International Conference on Rough Sets and Knowledge Technology, 2008, pp.620–627.

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

[11] E.J. Weyuker, "Evaluating software complexity measure". IEEE Transaction on Software Engineering, 1988, pp.1357–1365.

[12] L. Arockiam, A. Aloysius and J. Charles selvaraj, "Extended Weighted Class Complexity: A new of software complexity for objected oriented systems", Proceedings of International Conference on Semantic E-business and Enterprise computing (SEEC), 2009, pp. 77-80.

[13] L. Arockiam, K. Geetha and A. Aloysius, "On Validating Class Level Cognitive Complexity Metrics", CiiT International Journal of Software Engineering and Technology, 2010, pp.152-157

[14] Deepti Mishra and Alok Mishra, "Object-Oriented Inheritance Metrics: Cognitive Complexity Perspective", Proceedings of the 4th International Conference on Rough Sets and Knowledge Technology, 2009, pp. 452–460.