Analysis of Cognitive Complexity with Cyclomatic Complexity Metric of Software

Dinuka R. Wijendra
Sri Lanka Institute of Information Technology
Sri Lanka

K.P. Hewagamage
University of Colombo School of Computing
Sri Lanka

ABSTRACT
The complexity of a software can be derived by using software complexity metrics which determines various software attributes quantitatively. The cognitive complexity metric, which is considering as a prominent factor of calculating the complexity of a software, evaluates how the human brain processes the given software with respective to different aspects, which involves the concept of cognitive Informatics. The McCabe's cyclomatic complexity is currently using as a standard complexity metric to determine the software complexity in terms of the number of linear independent paths. Thus, a broad analysis is carried on how the cognitive complexity derived based on Cognitive Information Complexity Measure (CICM) and the McCabe's cyclomatic complexity relates and varies with the computation of the given software, resulting that the cognitive complexity value becomes high with respective to its cyclomatic complexity. The cognitive complexity computation beyond the CICM value does not have a strong linear relation of the computation with cyclomatic complexity, which may be derived with a certain combination of relationships based on the factors involved within the cognitive complexity determination.

General Terms
Cognitive Complexity Metric, Cyclomatic Complexity, Cognitive Informatics, Basic Control Structures, Software Complexity

Keywords
BCS, CC, CICM, LOC

1. INTRODUCTION
Software complexity metric can be defined as a quantitative measurement that can be obtained from the software itself, related documents and the processes which are going to be followed. IEEE defines the software complexity as the degree to which a system or component has a design or implementation that is difficult to understand and verify [1]. Measuring the software complexity using different metrics results to have an opportunity with decreasing the complexity as well as it tends to maintain its quality and the cost. The greater the complexity value, the software becomes more error prone and difficult to maintain while its quality becomes low and the cost becomes increasing. The term "Complexity" can be calculated either by using size metrics, structural metrics and object-oriented metrics since it can be determined with respective to another software attributes. Therefore, many software complexity metrics have been standardized in order to compute their complexity values according to the software attribute which has been considered. Therefore, much effort has been taken to identify the techniques and software metrics to measure the software complexity [2].

The cognitive complexity metrics plays a predominant role of determining the human comprehension effort behind a given software, which is still under the validation process. Many researches have been conducted to propose different cognitive complexity measurements considering numerous software factors. The cyclomatic complexity is a validated software complexity metric, which is being used in the software industry to compute software complexity. Hence, the evaluation of the non-validated cognitive complexity and the cyclomatic complexity has to be performed, so that the relations ship among both of the metrics can be outlined.

2. COGNTIVE COMPLEXITY METRIC OF SOFTWRE
The term "Cognitive Complexity" comes under the field of Cognitive Informatics which studies the internal information processing mechanism of the human brain as well as its software application [3]. Therefore, the cognitive complexity tries to measure the human effort needed to perform a task or to understand the logic behind the given software [4]. The human effort needed to understand or to develop a given software is always vary from human, which can be considered as a subjective measurement. The same software can be quantitatively analyzed by different set of people such that the cognitive complexity calculation can be performed with respective to various aspects. So that many researches have been come up with the computation of the cognitive complexity of a software by proposing several aspects which are different with another proposed cognitive complexity calculation. Since it is very difficult to analyze a common methodology of performing the cognitive complexity calculation, a standard way for the cognitive complexity computation is still not arrived.

One of the researches of computing the cognitive complexity was performed by A. K. Misra and D. S. Kushwaha [3], by adhering to the idea of cognitive informatics, which identifies the functional complexity of a software depends on the internal architecture flow and its inputs and outputs [5], [6]. Thus, a software had been represented as a collection of information in which the information can be represented as a set of operators and operands, while some cognitive weights have been assigned for the Basic Control Structures (BCS) within the source code. The same computation was further improved with the concept of nested BCS s in which the cognitive complexity is calculated by the weights assigned with the multiplication corresponding to the nesting level [7]. Another computation of a cognitive complexity had been introduced by S. Misra [4], such that the cognitive weight complexity measure was defined as the cognitive weight of the simplest software component, which is the linear structured BCS s. Therefore, the cognitive complexity was calculated according to the weightages assigned for the BCS s and the cognitive weight units inside the given source code.

So that it can be concluded that the amount of information inside the software was calculated according to the cognitive weights assigned for each information category defined.

As another approach of deriving the cognitive complexity, J. K. Chhabra [8] determined the cognitive complexity in which it considered the way of scattering of the internal information in terms of Lines Of Codes (LOC) inside the source code. Because it has been discovered that the effort that the human brain goes high if the distance of the module declaration and its usage is high rather than that happens in directly without much more distance. This includes the number of LOC, which are there form the variable or method declaration, initialization and its actual usage. Together with the Spatial aspect which have been computed, it combined the cognitive complexity value of the internal information by assigning the cognitive weights as in previous research outcomes, then the summation of both aspects derived as the total cognitive complexity of the given software. The same spatial aspect was continued by considering the recursive functions' spatial capability [9]. Based on the relationship between structures, Y. Choe, C. Jong and S. Han introduced a way of computing the cognitive complexity of a software [10]. Basically, the complexity value of that approach mainly considered with the scope of the variables used inside the software. Also, by considering the LOC value and the identifiers in the source code, another way of cognitive complexity computation was introduced [11].

Some researches were conducted to calculate the cognitive complexity for an object-oriented code rather than just not limited only for the procedural source codes only. As a result of that, U.Chhillar and S. Bhasin proposed a way of computing the Cognitive complexity of an object oriented code by considering the inheritance level of statements in classes, types of control structures, nesting of control structures and the size of the program [12]. As another approach, D. S. Kuashwaha and A. K. Misra proposed a way of cognitive complexity computation by considering the number of methods per class, reference to other object, number of independent functions performed by methods in the class, number of lines of code per method, probability of use of instance variable and the amount of functional overlap of classes in the object-oriented code given [13]. As another approach of the spatial aspect of the object-oriented code was introduced with respective to the method location rating, class relation measure and object relation measure [9].

Any software complexity metric should be validated with the standard software complexity metrics to determine the practicability and the comprehension of the proposed software with the real time applications. Among the available software complexity frameworks, Weyuker properties and the properties under Briand's framework can be considered as prominent. Weyuker defines nine software complexity properties [14] where as five properties have been declared under Briand's framework [15]. Satisfaction of most of the properties under one or both frameworks can let the proposed software complexity metric under the real usage of applications. Hence most of the proposed cognitive complexity metrics have been validated against these frameworks in order to verify their comprehended usage [16-20].

With respective to the researches which have been conducted to compute the cognitive complexity of either a procedural or an object-oriented code, most of the researches pointed out the basic aspects of how the human brain affects with understanding the logic at a time where the same research has been continued with the solutions for the drawbacks of the previous aspect and sometimes with another aspect as well. Therefore, it can be concluded that those computations were highly based on the amount of information inside the source code and their spatial aspects within the source code. The information within the source code can be further categorized into data types, data structures, BCS s and the user defined functions. Thus, by considering those two basic aspects of calculating the cognitive complexity, an analysis of how it relates with the computation of the cyclomatic complexity metric which is considered as a standard software complexity metric to evaluate the complexity of a software, is going to be discussed.

3. CYLOMATIC COMPLEXITY METRIC OF SOFTWARE

Cyclomatic complexity of metric was introduced by McCabe [21], which indicates the structural complexity of a module quantitatively by considering the control flow of the given method or a module of a program. It is widely used in many industrial applications to compute the complexity value. Simply it can be said that the cyclomatic complexity value is measuring the number of linear independent paths within the source code given so that higher the complexity value, it is more complex to understand, high number of test cases to modify the particular code. Thus, it can lead to a high cost and effort as well. Basically, it is calculated with respective to the graph theory in which the procedural statements of the source code are going to be converted to a control flow graph [22]. The control flow graph describes the logical structure of the software modules in which its nodes represent the computational statements or expressions and the edges represent the control between nodes [23-24]. Therefore, the Cyclomatic Complexity (CC) metric is going to be defined as,

$$CC = e - n + 2$$

where,

CC - the Cyclomatic Complexity value of the control flow graph (G) drawn pf the given program,

e – number of edges in G,

n – number of nodes in G.

Furthermore, the same complexity value can be obtained by determining the number of decision statements which directly affects for the complexity in the program and can be calculated as,

$$CC = d + 1$$

where,

d – number of decision statements inside the program.

Decision statements can be considered as if statements, number of cases within switch, all kinds of loops and try-catch statements.

The calculation of the cyclomatic complexity value with respective to the edges and nodes inside the control flow graph demonstrated in Fig 2 based on decision statements inside the source code under Fig 1 is mentioned below.

```
void IsNumber (String S)
{
boolean b= true;
for (i=0,i<S.length,i++)
{
if ((ASCI(CharAT(i)) > value) & (ASCI(CharAT(i)) < value))
        b=true;
else
        break;
}
```

Fig 1: Sample source code to calculate the cyclomatic complexity

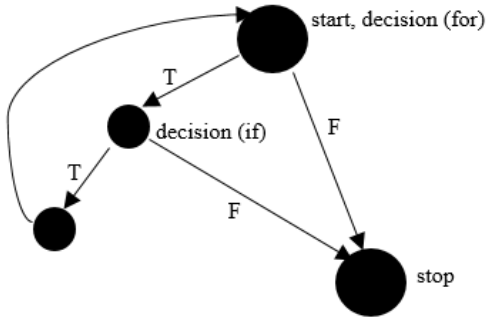

Fig 2: Control flow graph for the source code in Fig. 1

$$CC\ (manual) = e - n + 2 = 5 - 4 + 2 = 3$$

$$CC\ (manual) = d + 1 = 2 + 1 = 3$$

$$CC\ (RSM) = 3 + 1\ (compund\ statements) = 4$$

Meanwhile some researches were conducted to analyze the problems of cyclomatic complexity. It was found as an issue of calculating the cyclomatic complexity value manually, using the equations and by using Resource Standard Metric (RSM) which is a commercially available tool for the code quality analysis, is different such that the manual computation does not count the multiple conditions involved within one statement but RSM counts it. Several researches have been conducted such that proposing some improvements to be done for the existing cyclomatic complexity metric by addressing its issues. One solution was that to improve the existing cyclomatic complexity with the level of module interactions and the module coupling value [23]. Same research has been conducted to extend with introducing two formulas on diversity of modules with different types of coupling. Although there were some modifications of extending the way of computing the cyclomatic complexity, still the same standard way of calculating the cyclomatic complexity with respective to the standard two equations is being followed.

4. RELATIONSHIP BETWEEN CYCLOMATIC COMPLEXITY AND COGNITIVE COMPLEXITY

It is essential to analyze the relationship between the cyclomatic complexity with existing standard complexity metrics. Most of the analysis have been done with the standard cyclomatic complexity concept with other available standard complexity metrics to come up with the relationship among those metrics. The relationship between cyclomatic complexity with LOC value has been analyzed such that it was concluded that both are having a linear relationship with few investigate statistical issues namely the distribution among both [25]. With respective to the computation of the cognitive complexity of a given software with most of the researches conducted, it can be concluded that it basically determines the amount of internal information inside the source code quantitatively by assigning the corresponding cognitive weights and also the spatial aspect of how the information scatters through the software in terms of LOC. Moreover, the cyclomatic complexity of a software determines the number of linear independent paths that its internal logic can be gone through. Therefore, to analyze the relationship between the cognitive complexity and the cyclomatic complexity, the relation of the internal information, their spatial capacities and the number of linear independent paths inside the given software should be thoroughly considered.

A source code, which contains lots of data variables, structures, data types and BCS s results with high total cognitive weight, so that the amount of information will be high. Furthermore, if the distance between variable declaration, initialization and functions' definition to their actual usage or calling is high, their spatial capacity also tends to be high. Therefore, the cognitive complexity of such a source code will be getting a high value. The cyclomatic complexity of that source code fully depends on the number of BCS s inside it, irrespective of the amount of information and their spatial capacities. Thus, the cyclomatic complexity of a software in which its cognitive complexity value is high, can be either low, medium or high depending on BCS s is there in the source code given. The possibilities of determining the cyclomatic complexity with respective to the cognitive complexity of the same software with the architectural and the spatial aspects are showed in Table 1.

Table 1. Relationship of cognitive complexity and the Cyclomatic complexity under different possibilities

Cognitive Weight		**Spatial Capacity**	**Cognitive Complexity**	**CC**
Data types & structures	**BCS**			
High	High	High	High	High
High	High	Low	Medium	High
Low	High	High	Medium	High
Low	High	Low	Medium/ Low	High
High	Low	High	Medium	Low
High	Low	Low	Medium/ Low	Low

Low	Low	Low	Low	Low
Low	Low	High	Medium/ Low	Low

5. RESULTS AND DISCUSSION

To analyze the relationship between the cognitive complexity and the cyclomatic complexity, fifteen "C" programs have been selected from E. Balagurusamy's book, "Programming in ANSI C" [26]. The Cognitive Information Complexity Measure (CICM) has been calculated according to D. Kushwaha and A. K. Misra's approach [3] is listed in Table 2.

Table 2. CICM, LOC and CC values for sample source codes

Program No	CICM value	LOC	CC value	Ref.of source code
1	2.52	10	2	pp.38
2	3.08	9	1	pp.49
3	16.16	16	2	pp.9
4	4.75	17	1	pp.69
5	10.48	11	2	pp.64
6	11.24	15	2	pp.61
7	10.32	16	2	pp.43
8	11.68	15	3 ; RSM=4	pp.103
9	14.04	12	2	pp.102
10	14.52	17	2	pp.42
11	13.3	15	2	pp.133
12	18.4	16	2	pp.39
13	28.49	20	3	pp.106
14	36.69	17	4	pp.113
15	39.2	25	5	pp.122

The dependency levels of CICM, LOC and CC values of those fifteen source codes are graphically represented in Fig 3.

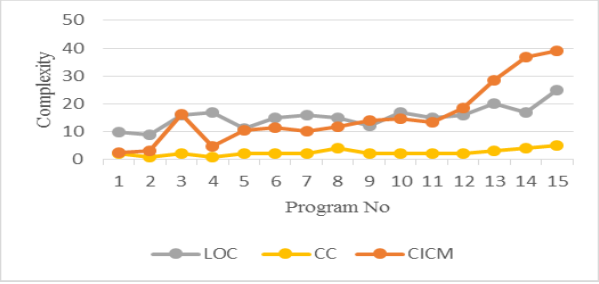

Fig 3: Dependency between CICM, LOC and CC values for the sample programs

The CICM value computed in the Table 2 demonstrate the cognitive complexity of a given source code with respective to the weighted information count, which defines the LOC value, total number of identifiers and operators and the cognitive weights assigned for BCS s [3]. The cyclomatic complexity computation is limited only for the occurrences of linear independent paths based on the BCS s irrespective of the amount of information behind the source code. Therefore, it is inevitable that the cognitive complexity value generated in Table 2 is always higher than its cyclomatic complexity value. Since the cognitive complexity value is considered in one way according to a prior research, the problem of occurring high cognitive complexity with respective to its cyclomatic complexity have to be analyzed. Table 1 specifies the various situations that the cognitive complexity and the cyclomatic complexity derives, so that it can be concluded that the cognitive complexity cannot be always derived as a higher value with respective to the cyclomatic complexity value. Thus, the relationship between both the metrices should be analyzed through its definitions, which may go beyond its quantitative computations.

The cognitive complexity is defined as the human comprehension effort to understand a given software, where the cyclomatic complexity defines the number of linear independent paths inside the given source code. The human comprehension effort of a software is a subjective measurement, since different users tends to understand the software in different ways. This also includes the users background, which includes the programming language, past experience and the knowledge regarding the computing environment. The amount of information plays a vital role of determining the effort of understandability such that the higher effort should be taken to comprehend a source code with lots of information. The special capacity of a source code is another factor, which defines the LOC count between the declaration and the usage of the code segments. The spatial capacity occurs high, when the distance between a segment declaration and its usage is high, resulting a high cognitive complexity. Moreover, the tools available with the programming environment namely Integrated Development Environment (IDE) and the documentation available such as manuals and commenting procedures will reduce the comprehension effort of understandability, which results in low cognitive complexity. Moreover, there may be many different factors, which can be considered as the factors for cognitive complexity determination as per the users involved with it. The cyclomatic complexity value is an objective measurement, which does not vary with the human involvement of the source code. Thus, deriving an exact relationship between both the metrices is a difficult process, such that both the metrices varies in different scenarios.

A source code with higher cognitive complexity can be occurred due to the high amount of information, high spatial aspect, problems in users background, less facilities provided and due to many other factors. The higher number of information may not be occurred only due to the BCS s, but also due to the number of identifiers and operators as well. A source code containing high information with less number of BCS s will lead to a less cyclomatic complexity, where its cognitive complexity may end up with a higher value due to the amount of information, if it is considered for the cognitive complexity calculation. On the other hand, the comprehension effort of the same source code may less, if the participant has a prior knowledge of the source code logic. Thus, both of the metrics values will be less. Therefore, the deviation of both the metrics for a same source code also depends on many factors, which cannot be limited to a certain context resulting in different scenarios of their relationships.

The relationship among the cognitive complexity metric based on CICM value computation and the cyclomatic complexity

metric clearly shows that the cognitive complexity of a source code will be always high with respective to its cyclomatic complexity value. The above relationship will show a deviation with respective to the number of factors considered for the cognitive complexity computation, which does not clearly end up with a linear relation.

6. CONCLUSION

The cognitive complexity metric defines how the human brain goes through the internal logic of the given software which includes the theory of Cognitive Informatics. It is a subjective measurement, such that the way of understanding level depends from the person resulting that a standard way of determining of cognitive complexity metric is still under progress. According to the past researches under cognitive complexity, it can be concluded that it covers the amount of information within the source code quantitatively by assigning the corresponding cognitive weights and how each information scatters through the software. This paper analyzes how the computation of the cognitive complexity through one of the computations namely CICM, differs with cyclomatic complexity, which calculates the number of linear independent paths through the source code. According to the results obtained, it can be derived that the CICM of a software will be always high comparing to its cyclomatic complexity. Since CICM is one of the cognitive complexity computations, the analysis of cognitive complexity should go beyond that computation with its definition. Therefore, it can be derived that both of the computations do not have a specific relationship, whereas varied relation can be existed under some circumstances with respective to the BCS s inside the source code. This can be occurred in which both are evaluating the complexity of a software in two different perspectives, although both evaluates the complexity of the same software.

7. FUTURE WORK

The relationship between the cognitive complexity and the cyclomatic complexity has been analyzed with respective to fifteen sample programs addressed for different problems solving as a preliminary step. The analysis should be further expanded into more programs to obtain a comprehensive relationship among both the metrics and the observe whether the current relationship is existed for verification.

8. ACKNOWLEDGMENTS

The completion of this analysis could not have been achieved without the continues motivation and the guide given by Prof. K. P. Hewagamage, University of Colombo School of Computing, Sri Lanka.

9. REFERENCES

[1] IEEE Computer Society: IEEE Standard Glossary of Software Engineering Terminology, IEEE Standard 610.12-1990.

[2] J. C. Munsona and T. M. Khoshgoftaar, "The dimensionality of program complexity," Proceedings of the 11th International Conference on Software Engineering, pp. 245–253, 1989.

[3] D. S. Kushwaha and A. K. Misra, "A Modified Cognitive Information Complexity of Software", ACM SIGSOFT Software Engineering Notes, vol. 31, no. 1, January 2006.

[4] S. Misra, "A Complexity Measure Based on Cognitive Weights", International Journal of Theoretical and Applied Computer Sciences, vol. 1, pp. 1-10, 2006.

[5] Y. Wang and J. Shao, "Measurement Of The Cognitive Functional Complexity of Software", IEEE International Conference on Cognitive Informatics, 2003.

[6] Y. Wang, "The Real-Time Process Algebra (RTPA)", Annals of Software Engineering: An International Journal, Vol. 14, USA, pp. 235 – 274, 2002.

[7] D. S. Kushwaha and A. K. Misra, "Improved Cognitive complexity Measure: A Metric that Establishes Program Comprehension Effort", ACM SIGSOFT Software Engineering Notes, vol. 31, no. 5, September 2006

[8] J. K. Chhabra, "Code Cognitive complexity: A New Measure", World Congress on Engineering, vol. 2, July 6-8 2011.

[9] C. R. Douce, P. J. Layzell and J. Buckley, "Spatial measures of software complexity", in Proceedings of 11th meeting of Psychology of Programming Interset Group, Leeds, January 1999.

[10] Y. Choe, C. Jong and S. Han, "Software Cognitive Information Measure based on Relation between Structures", 2013.

[11] T. Klemola and J. Rilling, "A Cognitnve Complexity metric based on Category learning", in Prceeding of the 2nd IEEE International Conference of Cognitnve Informatics (ICCI'03), 2003.

[12] U. Chhillar and S. Bhasin, "A New Weighted Composite Complexity Measure for Object-Oriented Systems". International Journal of Information and Communication Technology Research.vol.1, no. 3, July 2011.

[13] D. S. Kushwaha and A. K. Misra, "Cognitive Information Complexity measure of Object Oriented Software – A Practitioner's Approach", in Proceedings of the 5th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems, pp174-179, February 15-17, 2006.

[14] E. Weyuker, "Evaluating Software Complexity Measure", IEEE Transaction on Software Complexity Measure, 14 (9), pp. 1357 – 1365, 1988.

[15] L. Briand and S. Morasca, "Property Based Software Engineering Measurement", IEEE Transactions on Software Engineering, vol. 22, no. 1, January 1996.

[16] D. S. Kushwaha and A. K. Misra, " Robustness Analysis of Cognitive Information Complexity Measure using Weyuker Properties" , ACM SIGSOFT Software Engineering Notes, vol. 31, no. 1, January 2006.

[17] S. Misra and A. Misra, "Evaluation and Comparison of Cognitive complexity Measure", ACM SIGSOFT Software Engineering Notes, vol. 32, no. 2, March 2007.

[18] S. Misra, "Validating Modified Cognitive complexity Measure", ACM SIGSOFT Software Engineering Notes, vol. 32, no. 3, May 2007.

[19] J. K. Chhabra and V. Gupta, "Evaluation of Object-Oriented Spatial Complexity Measures", ACM SIGSOFT Software Engineering Notes, vol. 34, no. 3, May 2009.

[20] S. Misra and A. Misra, "Evaluating Cognitive complexity Measure with Weyuker Properties", Third IEEE International Conference on Cognitive Informatics

(ICCI'04), 2004.

[21] T. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, December 1976.

[22] T. McCabe, "Structured Testing: A Testing Methodology Using the Cyclomatic complexity Metric," NIST Special Publication, pp. 500-235, September 1996.

[23] A. Madi, O. K. Zein and S. Kadry, " On the Improvement of Cyclomatic complexity Metric", International Journal of Software Engineering and its Applications, vol 7, No 2, March 2013.

[24] G. K. Gill and C. F. Kemerer, "Cyclomatic complexity Density and Software Maintainance Productivity", IEEE Transactions of Software Engineering, vol 17, No 2, December 1991.

[25] J. Graylin, R. K. Smith, N. A. Kraft, J. E. Hale and D. Hale, "Cyclomatic complexity and Lines of Code: Empirical Evidence of a Stable Linear Relationship", International journal of Software Engineering and Applications, pp. 2: 137-143, October 2009.

[26] E. Balagurusamy,Programing in ANSI - C, Tata McGraw-Hill Publishing Company Limited, New Delhi,Second edition, 1992.