# Analysis and Enhancements of a Cognitive Based Complexity Measure

D. I. De Silva[1], N. Kodagoda[4]
Faculty of Computing
Sri Lanka Institute of Information Technology
Malabe, Sri Lanka
[1]dilshan.i@sliit.lk, [4]nuwan.k@sliit.lk

S. R. Kodituwakku[2], A. J. Pinidiyaarachchi[3]
Faculty of Science
University of Peradeniya
Peradeniya, Sri Lanka
[2]salukak@pdn.ac.lk, [3]ajp@pdn.ac.lk

*Abstract*—**As stated by Tom DeMacro, something that cannot be measured is uncontrollable. Thus, a number of metrics have been developed to measure the complexity associated with software by considering various aspects such as size, control flow and data flow between modules, cognitive informatics etc. Amongst these aspects, cognitive informatics is recognized as a promising aspect in measuring software complexity. Thus, majority of the complexity metrics that were proposed after the introduction of cognitive informatics have been proposed mainly based on the cognitive aspect. Amongst them, Chhillar and Bhasins' weighted composite complexity measure is one of the few metrics that had attempted to measure the complexity of a program by considering more than three or more complexity factors. After a thorough analysis, in a previous study, the authors identified that the weighted composite complexity measure could be further improved by considering more complexity factors. This paper extends the previous study to identify the most appropriate factors that could be considered by the weighted composite complexity measure. Using the opinions of the industry experts, the authors were able to discover that compound conditional statements, threads and recursion could also be considered by the weighted composite complexity measure. Accordingly, the weighted composite complexity measure was enhanced to capture the complexities that arise due to those factors. The paper also includes a demonstration of the complexity calculation method of the improved weighted composite complexity measure with the use of three sample java programs, which were written by incorporating the above mentioned factors. In addition, an application of the weighted composite complexity measure to the same programs are also given in the paper, to illustrate the changes in complexity values of the two measures.**

***Keywords***— **Software complexity; Cognitive informatics; CB measure**

## I. INTRODUCTION

In the modern era, majority of the work force in many of the countries heavily rely on software applications and the demand for reliable and powerful software applications tend to grow with each day to come. With this, the complexity of software programs has also taken an upward trend making measurement of software complexity an essential task that each software organization should follow in order to ensure that the cost to maintain and enhance such programs does not run rampant.

An initial problem that arises when trying to understand software complexity is defining what it really means as there is no standard definition in the current literature. Many experts in the software engineering discipline share different views on software complexity. Brooks defines software complexity as a crucial property, not as an unintentional one. Thus, explanations of a software entity which removes its complexity often removes its core values [1]. According to the Institute of Electrical and Electronic Engineers (IEEE) software complexity is "the degree to which a system or component has a design or implementation that is difficult to understand and verify [2].

As stated by Tom DeMacro, something that cannot be measured is uncontrollable. Thus, a number of metrics have been developed to measure the complexity associated with a software program. Those metrics have attempted to measure complexity by considering various aspects such as size, control flow and data flow between modules, cognitive informatics etc.

Cognitive informatics is considered as a promising solution for measuring software complexity as it measures the mental effort required to read, understand and write programs. Thus, majority of the complexity metrics that were proposed after the introduction of cognitive informatics have been proposed mainly based on the cognitive aspect.

Most the cognitive based complexity metrics proposed for object-oriented (OO) programs have relied on two or three complexity factors to derive the complexity of an entire system. Weighted composite complexity (CB) measure which was introduced by Chhillar and Bhasin, is one of the few cognitive based metrics that has considered three or more complexity factors to measure the complexity of an OO system. However, in a previous study [17, 18], the authors suggested some other factors that could be considered by the CB measure to further strengthen its accuracy. This paper extends the previous study to identify the most appropriate factors that could be considered by the CB measure using the opinions of several industry experts.

A history of the complexity metrics which have been proposed mainly based on the cognitive aspect is given in the next section. The methodology that was used to conduct the study is discussed in section III. Section IV proposes the improved CB measure. Finally, the conclusion of the paper is given in section V.



978-1-5090-4096-4/17/$31.00 ©2017 IEEE

## II. Cognitive Based Complexity Metrics

Proposing cognitive based complexity metrics started with the introduction of Cognitive Functional Size (CFS) metric. It was proposed by Shao and Wang in April 2003[3]. The unique cognitive weights that was assigned for different internal basic control structures (BCS) by Shao and Wang laid the foundation for the introduction of many complexity metrics based on cognitive informatics. The CFS measure was based on three fundamental factors: the total cognitive weight of all internal BCSs, the number of keyboard inputs and outputs.

In January 2006, the Cognitive Information Complexity Measure (CICM) was proposed by Kushwaha and Misra in an attempt to understand the cognitive information complexity and the information coding efficiency of a program [4]. It was based on cognitive weight of internal BCSs, identifiers, operators and lines of code (LOC).The same year, Sanjay Misra introduced the Cognitive Weight Complexity Measure (CWCM) [5]. This measure was only based on the cognitive weight of internal BCSs. Thus, it is simple to understand and calculate values using this metric. Towards the latter part of the year, Sanjay Misra proposed the Modified Cognitive Complexity Measure (MCCM). In addition to the weights of the BCSs, it considered the total number of occurrences of operands and operators [6].

In 2007, Sanjay Misra proposed another metric named Class Complexity (CC) to compute the complexity of an OO system [7]. This measure first calculated the complexity of a method based on BCSs and then added the complexities of all methods which belonged to a class to compute the complexity of a class. Finally, the sum of the complexities of all the classes were taken into account to compute the complexity of an OO system.

In 2008, together with Akman, Sanjay Misra introduced Weighted Class Complexity measure (WCC) [8]. Like the CC measure this was also used to compute the complexity of an OO system. The process that it followed to compute the complexity was also similar to the process that was used by the CC measure. In addition to considering the cognitive weight of internal BCSs, WCC considered the complexity added to a class due to global attributes and message calls as well. The metrics that were introduced before WCC only considered the cognitive weight due to a method call (which is 2), when there was a method call from one class to another. However, WCC considered both the cognitive weight of the method call as well as the cognitive weight of that called method to compute the complexity of message calls between classes. However, if the message calls were between the methods of the same class, then only the weight due to method call was considered for complexity calculation.

Gupta and Chhabra suggested cognitive-spatial complexity measures to compute the complexity of classes and objects, in 2009 [9]. These metrics considered spatial as well as the architectural aspect of a program in calculating software complexity. The spatial aspect was taken into account by considering the distance between program elements in terms of number of LOCs and the architectural aspect was taken into account by considering the weights of BCSs.

In 2011, Sanjay Misra, Akman and Koyuncu proposed the Cognitive Code Complexity measure (CCC) to evaluate the design of an OO program [10]. Similar to their previous metrics: CC and WCC, this measure also computed complexity in three stages. In the first stage, it calculated complexity at the method level using cognitive weights of BCSs. In addition to the cognitive aspect, the inheritance aspect was also considered by this measure. The inheritance aspect was considered in the third stage when the complexity of the entire program was calculated. In July that same year two more complexity measures based on cognitive weights were introduced. One of them was Code Cognitive Complexity measure (CCC) which was proposed by J. K. Chhabra [11]. This considered the absolute distance in terms of LOCs between a module call and its definition/use, the input and output parameters of a module, and the cognitive weights. All the cognitive metrics proposed until this study used the cognitive weights defined by Shao and Wang for BCSs. In this study, Chhabra managed to define cognitive weights for variable and constant data types as well. The other metric was Chhillar and Bhasin CB measure [12]. It measured the complexity of a program using the following four factors.

1. **Inheritance**: A weight of zero was assigned for executable statements which are at the base class, a weight of one for executable statements which are at the first derived class, a weight of 2 for executable statements which are at the next derived class and so on.
2. **Type of control structures**: A weight of zero, one, two and n was assigned for sequential statements, conditional control structures, iterative control structures, and switch-case statements with n cases respectively.
3. **Nesting level of control structures**: A weight of zero was assigned to sequential statements. A weight of one was assigned to statements which are at the outer most level of nesting, a weight of two for statements which are at the next inner level of nesting and so on.
4. **Size**: Size of an executable statement was measured by counting the number of operators, operands, functions/methods and strings in that statement.

In February 2012, Code Comprehending Measure (CCM) was proposed by Gurdev, Satinderjit, and Monika. It measured the complexity based on three aspects: data volume, structural complexity and data flow [13]. Data volume factor was computed by considering the unique variables and operators in a BCS and the total number of times that those variables and operators occur. Structural complexity was computed by considering the cognitive weight of BCSs. Finally, data flow factor was computed by considering the flow of data between BCSs through variables. In June, the same year, Sanjay Misra et al introduced a set of cognitive metrics to measure the method, message, attribute, class, and code complexities of OO programs [21]. Later in that year, Aloysius and Arockiam proposed their cognitive complexity metric which measured the complexity that occurred due to different types of coupling between classes [14]. This measure considered five coupling types, namely Data Coupling (DC), Global Data Coupling (GDC), Internal Data Coupling (IDC), Lexical Content

Coupling (LCC), and Control Coupling (CC). Based on the results obtained from a set of experiments, Aloysius and Arockiam assigned different cognitive weights for the five coupling types as shown in Table I.

In 2013, S. Misra et al introduced the multi-paradigm complexity (MCM) measurement [22]. It was based on a number of OO and procedural factors: attributes, variables, BCSs, objects, invocation of methods using objects , cohesion and inheritance.

In November 2014, K. Jakhar and K. Rajnish proposed New Weighted Method Complexity (NWMC) to compute the complexity of a program [15]. This measure was based on the cognitive weights of BCSs, keyboard inputs and outputs, local and formal parameters.

M. A. Shehab et al proposed a new weighted complexity metric in 2015. It considered cognitive weights of flow chart controls, number of operations, number of variables declarations, number of external libraries and functions, number of function arguments, and number of locally called functions [16].

TABLE I. COUPLING TYPES AND THEIR COGNITVE WEIGHTS

| Coupling Type | Cognitive Weight |
|---|---|
| Control coupling | 1 |
| Global data coupling | 1 |
| Internal data coupling | 2 |
| Data coupling | 3 |
| Lexical Content Coupling | 4 |

## III. METHODOLOGY

The main intention of this study was to propose additional factors to the CB measure in order to make its complexity calculation more accurate. In previous works [17, 18], the authors presented additional factors that could be incorporated to the CB measure in order to make its complexity calculation more accurate. In the present paper, the authors extend the study to identify the most appropriate factors that could be considered by the CB measure based on the opinions of the industry experts in the software engineering field.

Initially, twelve industry experts with more than three years of working experience were explained how the CB measure calculates the complexity of a program. The industry experts were chosen in a random manner ensuring that there was at least one industry expert representing one of the top five leading software companies in Sri Lanka. Then using a questionnaire they were asked which of the factors that were previously identified [17], [18] by the authors could be incorporated in an Improved CB measure (ICB). They were also asked what would be the impact that a certain factor would have on the understandability of a program. Since the CB metric measured the complexity of a program based on its understandability, the selection of the new factors was also done with respect to the understandability of a program.

To perform the analysis, initially the impact levels which were given in the questionnaire were converted to quantitative values by ranking them on a scale of one to five. Then a weighted average rank of the factors was obtained by considering the working experience of the experts as the weighting factor. Next, the obtained continuous weighted average values were converted to discrete values and the factors which had a discrete value of four or more were considered for the ICB measure, as it meant that they have a "high" or "very high" impact on the understandability of the program. Thus, compound conditional statements, threads, and recursion were considered in the ICB measure. Table II shows the discrete weighted average ranks for each factor that were considered in the study.

## IV. IMPROVED CB MEASURE

Although the CB metric is a good measure, it could be transformed to a more accurate measure by incorporating the three factors which were identified in the previous section. Thus, the ICB measure depends on a total of seven factors.

TABLE II. WEIGHTED AVERAGE RANKS OF THE FACTORS

| Factor | Weighted Average |
|---|---|
| Compound conditional statements | 4 |
| Concurrent programs - threads | 4 |
| Recursive methods | 4 |
| Pointers and references | 3 |
| Dynamic memory access | 3 |
| Multiple inheritance | 3 |
| Declaration of unused objects | 3 |
| Number of iterations in a loop | 3 |
| Exceptions | 3 |
| Array declarations | 2 |

According to the industry experts, compound conditional statements is one factor that could be considered by the ICB measure. Although the CB measure considered complexity due to conditional statements, it did not differentiate the complexity due to simple and compound statements. Thus, the ICB measure attempts to address this by assigning a weight of one for each "&&" or "||" operator in a conditional statement. The idea behind this thinking was, inclusion of one "&&" or "||" operator would mean an increase of one in the number of conditions checked by a decisional statement. Since the CB measure allocated a weight of one for each decision statement regardless of the number of conditions it contained, the authors decided to add a weight of one for each"&&" or "||" logical operator as well. The complexity calculation of a program with a compound conditional statement using the ICB measure is demonstrated in Table IV.

Another factor that is considered by the ICB measure is threads. When computing the complexity that arises due to size, the CB measure adds a constant value of one for each operator, operand, method/function, and string in a statement. To differentiate the size of a normal statement, with a statement that includes a thread invocation, a constant value of two is added to the total size of each statement with a thread invocation. The complexity calculation of a program with a thread invocation using the ICB measure can be clearly understood by observing the size (S) values of Table V and Table VI.

The last factor that is considered by the ICB measure is recursion. To consider the effect of recursion, first the size (S) value of each statement belonging to the recursive function is multiplied by the corresponding weight (W) values of those statements. Next, the derived values are added to the final ICB value of that program. The complexity calculation of a program with recursion using the ICB measure can be clearly understood by observing the CB and ICB values of Table VII and Table VIII and the highlighted cells of Table VIII.

To demonstrate the complexity calculation of the ICB measure, three simple java programs were written by including the newly suggested factors. Those three programs are given in Fig. 1, Fig. 2, and Fig. 3. Complexities of those programs have been computed using both the ICB and CB measures. Table IV, Table VI, and Table VIII illustrates the complexity values based on the ICB measure and Table III, Table V, and Table VII illustrates the complexity values based on the CB measure.

```
public class Result{

  public void res(int marks){
   if(marks > 0 && marks < 50)
         System.out.println("Fail");
   else
         System.out.println("Pass");
  }

  public static void main(String args[]){
      Result r = new Result();
      r.res(50);
  }
}
```

Fig. 1 Sample program to demonstrate the effect of compound conditions

TABLE III. COMPLEXITY CACULATION OF CB FOR RESULT CLASS

| Executable Statement | S | Wn | Wi | Wc | W | S*W |
|---|---|---|---|---|---|---|
| public void res(int marks) | 2 | 0 | 1 | 0 | 1 | 2 |
| if (marks >0 && marks <50 ) | 8 | 1 | 1 | 1 | 3 | 24 |
| System.out.println("Fail") | 6 | 1 | 1 | 0 | 2 | 12 |
| System.out.println("Pass") | 6 | 1 | 1 | 0 | 2 | 12 |
| public static void main(String args[ ]) | 4 | 0 | 1 | 0 | 1 | 4 |
| Result r = new Result() | 5 | 0 | 1 | 0 | 1 | 5 |
| r.res(50) | 3 | 0 | 1 | 0 | 1 | 3 |
| **CB Value** | | | | | | **62** |

TABLE IV. COMPLEXITY CACULATION OF ICB FOR RESULT CLASS

| Executable Statement | S | Wn | Wi | Wc | W | S*W |
|---|---|---|---|---|---|---|
| public void res(int marks) | 2 | 0 | 1 | 0 | 1 | 2 |
| if (marks >0 && marks <50 ) | 8 | 1 | 1 | 2 | 4 | 32 |
| System.out.println("Fail") | 6 | 1 | 1 | 0 | 2 | 12 |
| System.out.println("Pass") | 6 | 1 | 1 | 0 | 2 | 12 |
| public static void main(String args[ ]) | 4 | 0 | 1 | 0 | 1 | 4 |
| Result r = new Result() | 5 | 0 | 1 | 0 | 1 | 5 |
| r.res(50) | 3 | 0 | 1 | 0 | 1 | 3 |
| **ICB Value** | | | | | | **70** |

The second statement in Table III includes a compound conditional statement. However, when computing the complexity of that statement, the CB measure does not consider about the type of the condition. Rather, it simply assigns a weight of one for the Wc factor. Thus, it is unable to accurately measure the complexity of programs with compound conditional statements.

However, this issue can be solved using the ICB measure, as it is able to differentiate the complexity based on the type of condition as well as the number of conditions. This can be clearly witnessed by observing the weight value allocated for the Wc factor of the second statement in Table IV.

```
public class MyThread implements Runnable{
  public void run( ){
   System.out.println("Thread running");
  }
  public static void main(String args[]){
      MyThread mythread = new MyThread();

      Thread thread = new Thread(mythread);

      thread.start();
  }
}
```

Fig. 2 Sample program to demonstrate the effect of threads

TABLE V. COMPLEXITY CACULATION OF CB FOR MYTHREAD CLASS

| Executable Statement | S | Wn | Wi | Wc | W | S*W |
|---|---|---|---|---|---|---|
| public void run( ) | 2 | 0 | 1 | 0 | 1 | 2 |
| System.out.println("Thread running") | 6 | 0 | 1 | 0 | 1 | 6 |
| public static void main (String args[]) | 4 | 0 | 1 | 0 | 1 | 4 |
| MyThread mythread =new MyThread() | 5 | 0 | 1 | 0 | 1 | 5 |
| Thread thread = new Thread(mythread) | 5 | 0 | 1 | 0 | 1 | 5 |
| thread.start() | 3 | 0 | 1 | 0 | 1 | 3 |
| **CB Value** | | | | | | **25** |

TABLE VI. COMPLEXITY CACULATION OF ICB FOR MYTHREAD CLASS

| Executable Statement | S | Wn | Wi | Wc | W | S*W |
|---|---|---|---|---|---|---|
| public void run( ) | 2 | 0 | 1 | 0 | 1 | 2 |
| System.out.println("Thread running") | 6 | 0 | 1 | 0 | 1 | 6 |
| public static void main (String args[]) | 4 | 0 | 1 | 0 | 1 | 4 |
| MyThread mythread = new MyThread() | 7 | 0 | 1 | 0 | 1 | 7 |
| Thread thread = new Thread(mythread) | 7 | 0 | 1 | 0 | 1 | 7 |
| thread.start() | 5 | 0 | 1 | 0 | 1 | 5 |
| **ICB Value** | | | | | | **31** |

By observing the complexity values derived in Table V for the last three statements, it can be seen that the CB metric measures the complexity of a statement which includes a thread invocation in the same manner as the other regular statements. Thus, it is unable to differentiate the complexity of a regular statement with a statement which includes a thread invocation.

However, using the ICB measure this problem can be solved. This can be clearly witnessed by observing the last three statements of Table VI.

```
public class Factorial{
 int a,b,c;

 public int fact(int n){
   if (n == 0)
       return 1;
   else
       return (n * fact(n-1));
 }
 public static void main(String args[]){
      Factorial t1 = new Factorial();
      int f = 0;
      f = t1.fact(5);
      System.out.println(f);
 }
}
```

Fig. 3 Sample program to demonstrate the effect of recursion

TABLE VII. COMPLEXITY CACULATION OF CB FOR FACTORIAL CLASS

| Executable Statement | S | Wn | Wi | Wc | W | S*W |
|---|---|---|---|---|---|---|
| public int fact(int n) | 2 | 0 | 1 | 0 | 1 | 2 |
| if (n==0) | 4 | 1 | 1 | 1 | 3 | 12 |
| return 1 | 1 | 1 | 1 | 0 | 2 | 2 |
| return (n * fact(n-1)) | 3 | 1 | 1 | 0 | 2 | 6 |
| public static void main(String args[]) | 4 | 0 | 1 | 0 | 1 | 4 |
| Factorial t1 = new Factorial() | 5 | 0 | 1 | 0 | 1 | 5 |
| int f = 0 | 4 | 0 | 1 | 0 | 1 | 4 |
| f = t1.fact(5) | 5 | 0 | 1 | 0 | 1 | 5 |
| System.out.println(f) | 5 | 0 | 1 | 0 | 1 | 5 |
| **CB Value** | | | | | | **45** |

TABLE VIII. OMPLEXITY CACULATION OF ICB FOR FACTORIAL CLASS

| Executable Statement | S | Wn | Wi | Wc | W | S*W |
|---|---|---|---|---|---|---|
| public int fact(int n) | 2 | 0 | 1 | 0 | 1 | 2 |
| if (n==0) | 4 | 1 | 1 | 1 | 3 | 12 |
| return 1 | 1 | 1 | 1 | 0 | 2 | 2 |
| return (n * fact(n-1)) | 3 | 1 | 1 | 0 | 2 | 6 |
| public static void main(String args[ ]) | 4 | 0 | 1 | 0 | 1 | 4 |
| Factorial t1 = new Factorial() | 5 | 0 | 1 | 0 | 1 | 5 |
| int f = 0 | 4 | 0 | 1 | 0 | 1 | 4 |
| f = t1.fact(5) | 5 | 0 | 1 | 0 | 1 | 5 |
| System.out.println(f) | 5 | 0 | 1 | 0 | 1 | 5 |
| **ICB Value** | | | | | | **65** |

From the complexity values derived in Table VII, it can be seen that the CB measure is unable to differentiate the complexity that occur due to a recursive and non-recursive method. However, the ICB measure address this issue and reports a higher complexity value in the presence of a recursive method. This can be witnessed by observing the complexity values derived in Table VIII.

Since the ICB measure computes the complexity of a program using a total of seven factors, it is able to measure complexity much more accurately than the CB measure.

## V. CONCLUSION

Chhillar and Bhasins' CB measure is one of the few metrics that has contravened the norm and has looked at the complexity from different aspects. It considers the complexities that arise due to inheritance level of statements, type of control structures, nesting level of control structures, and size of the program. In addition to these four factors, based on the opinions of several industry experts, this paper proposes three more additional factors that could be considered by CB measure to make it a more accurate measure. The authors are hoping to get the opinions of at least 50 industry experts in the future, to firmly determine the additional factors that could be considered by the CB measure.

## REFERENCES


[1] Fred P. Brooks, "No Silver Bullet - Essence and Accidents of Software Engineering", IEEE Computer, April 1987, 4 (20), pp.10 - 19.

[2] IEEE Std. 1061-1998 – IEEE Computer Society: Standard for Software Quality Metrics Methodology, 1998.

[3] A Shao and Y. Wang, "A new measure of software complexity based on cognitive weights," Canadian Journal of Electrical and Computer Engineering, April 2003, 28.

[4] D.S. Kushwaha and A.K. Misra, A modified cognitive information complexity measure of software. ACM SIGSOFT Software Engineering Notes, 31(1), Jan. 2006, pp.1-4.

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

[6] S. Misra, "Modified cognitive complexity measure", 21st International Conference on Computer and Information Sciences, 2006, pp. 1050 – 1059.

[7] S. Misra, "An Object Oriented Complexity Metric Based on Cognitive Weights", 6th IEEE International Conference on Cognitive Informatics (ICCI 07), 2007, pp. 134-139.

[8] S. Misra and K. I. Akman, "Weighted Class Complexity: A Measure of Complexity for Object Oriented System", Journal of Information Science and Engineering, 24 (6), November 2008, pp. 1689-1708.

[9] V. Gupta and J. K. Chhabra, "Object-oriented cognitive-spatial complexity measures", International Journal of Computer Science and Engineering, 3(6), 2009, pp.122-129.

[10] S. Misra, I. Akman, and M. Koyuncu, "An inheritance complexity metric for object-oriented code: A cognitive approach", Sadhana Academy Proceedings in Engineering Sciences, 36 (3), 2011, pp. 317 – 377.

[11] J. K. Chhabra. "Code Cognitive Complexity: A New Measure", World Congress on Engineering, July 2011, Vol. 2, London, U. K.

[12] U. Chhillar and S. Bhasin, "A new weighted composite complexity for object-oriented systems", International Journal of Information and Communication Technology Research, July 2011, 1(3), pp.101-108.

[13] G. Singh, S. Singh, and M. Monga, "Code Comprehending Measure", International Journal of Computers and Technology, 2 (1), February 2012, pp. 9 - 14.

[14] A. Aloysius and L. Arockiam, "Coupling Complexity Metric: A Cognitive Approach", International Journal of Information Technology and Computer Science (IJITCS), 4(9), August 2012, pp. 29- 35.

[15] A. K. Jakhar, K. Rajnish, "Measuring Complexity, Development Time and Understandability of a Program: A Cognitive Approach", International Journal of Information Technology and Computer Science (IJITCS), 6(12), pp.53-60, 2014.

[16] A M. A. Shehab, Y. M. Tashtoush, W. A. Hussien, M. N. Alandoli, and Y. Jararweh, "An Accumulated Cognitive Approach to Measure Software Complexity", Journal of Advances in Information Technology 6 (1), 2015, pp 27-33

[17] D. I. De Silva, S. R. Kodituwakku, A. J. Pinidiyaarachchi, N Kodagoda, Improvements to a Complexity Metric: CB Measure, IEEE 10th International Conference on Industrial and Information Systems (ICIIS), Peradeniya, Sri Lanka, December 17–20, 2015, pp. 401-406.

[18] D. I. De Silva, S. R. Kodituwakku, A. J. Pinidiyaarachchi, N Kodagoda, Limitations of an Object-Oriented Metric: Weighted Complexity Measure, 6th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, September 23–25, 2015, pp. 698-701.

[19] J. Bansiya, C. G. Davis, "A hierarchical model for object-oriented design quality assessment" IEEE Transactions on Software Engineering, 2002, pp. 4-17.

*[20]* S. R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object Oriented Design," IEEE Transactions on Software Engineering, June 1994, 20, pp. 476-493.

*[21]* S. Misra, M. Koyuncu, C Mateos, and A. Zunino, "A Suite of Cognitive Complexity Metircs", 12th International Conference on Computational Science and its Applications, June 2012, pp.234-247.

*[22]* S. Misra, F. Cafer, and I. Akman, "Multi-Paradigm Metric and its Applicability on Java projects", Journal of Applied Sciences, 10(3), 2013, pp. 203-220.