International Journal of Engineering and Management Research
e-ISSN: 2250-0758 | p-ISSN: 2394-6962
https://ijemr.vandanapublications.com

Peer Reviewed & Refereed Journal
Volume-13, Issue-5 (October 2023)
https://doi.org/10.31033/ijemr.13.5.8

CodePlex: Software Complexity Measuring Tool based on ECB Measure

Dharmathilake K. A. D. K. D[1], Nuwanthika P. G. P. J.[2], Fernando N. K. B.[3] and Bhanuka H. L.[4]
[1]Student, Department of Software Engineering, Sri Lanka Institute of Information Technology, SRI LANKA
[2]Student, Department of Software Engineering, Sri Lanka Institute of Information Technology, SRI LANKA
[3]Student, Department of Software Engineering, Sri Lanka Institute of Information Technology, SRI LANKA
[4]Student, Department of Software Engineering, Sri Lanka Institute of Information Technology, SRI LANKA

[1]Corresponding Author: it21214820@my.sliit.lk

Received: 17-09-2023 Revised: 03-10-2023 Accepted: 21-10-2023

ABSTRACT

The surge in popularity of object-oriented programming as a predominant paradigm in software development has spurred numerous studies to introduce metrics for assessing the complexity of object-oriented programs. These metrics typically fall into two primary categories: those focusing on object-oriented aspects and those centered on cognitive aspects, delineating their principal areas of concern when evaluating program complexity. Within the realm of cognitive aspects, the majority of metrics have historically been confined to the consideration of no more than three complexity variables. However, the ECB (Enhanced Cognitive Based) measure stands as a notable exception, capable of encompassing and addressing four or more intricate facets in the assessment of software program intricacy and difficulty. This research paper undertakes the exploration of the incorporation of these multidimensional metrics as refinements to the existing weighted composite complexity CB measure, originally introduced by Chhillar and Bhasin. In doing so, it endeavors to furnish a more comprehensive and holistic framework for the evaluation of program complexity, accommodating both object-oriented and cognitive dimensions. Furthermore, the study assumes the pivotal role of empirically validating the practical effectiveness of the ECB measure, seeking to bridge the chasm between theoretical metrics and their tangible applicability in real-world settings. Such an endeavor holds profound significance for software developers and researchers, proffering invaluable insights that can advance our understanding and management of intricate object-oriented programs.

***Keywords*--** CB Measure, Software Complexity Measure, Object Oriented Metric, Weighted Composite Complexity

I. INTRODUCTION

The literal meaning of complexity is referred to as a state of difficulty having parts of understanding or comprehending something. [1] In the context of software, complexity pertains to the intricacy that hinders the clear understanding of various aspects of the software process. It is the degree to which challenges are faced when verifying and understanding software systems and their components from the design phase to the implementation and maintenance phase [2]. Software complexity is not a new concept. Many researchers have been interested in this subject since the 1970s [3] [4]. Due to that, it was evident that the importance of having metrics to measure the software complexity is necessary. Over time, a multitude of metrics for measuring software complexity have been introduced. [3] [4] [5] [6] [7] [8] [9] Also, several classifications of these metrics were identified to understand the scope. Notably, a seminal classification waspresented by Halstead [3].

- Product Metric – Metrics to describe and quantifyqualities of the software product such as size, complexity, design features, performance, and quality level.
- Process Metric – Metrics to improve software development and maintenance such as identification and removal of defects during development, defect testing patterns, and time taken to respond to defects.
- Project Metric- Metric to explain qualities of the project such as the number of software programmers, life cycle over the software, cost, and productivity.

Though there are several metrics introduced, certain problems still arise related to this context. One is the ambiguousness and inconsistency of these metrics. [10] Though the metrics are well-defined and proven with evidence it was found out that most metrics cannot be applicable to each software program. Also due to the improved large diversity of programming languages and concepts these metrics which might have been useful before are no longer applicable.

The second one is the difficulty in correctly identifying the most suitable metric among many of the metrics. Due to the very same reason mentioned above it was hard to select the best and most appropriate metric to measure the software complexity and comprehend actual

Peer Reviewed & Refereed Journal
Volume-13, Issue-5 (October 2023)
https://doi.org/10.31033/ijemr.13.5.8

meaning and the benefit behind the measurements.

Another problem is to identify which quality should be taken into consideration when measuring software complexity. As a solution for this Wyuker has introduced 6 different properties that software complexity metrics should satisfy [11]. According to her observation, both Halstead's metrics [3] and Cyclomatic complexity [4] which are considered a few of the most accepted complexity metrics worldwide fail to satisfy all these properties.

With the Object Oriented (OO) concepts came into light another problem was raised, which is the scarcity of metrics that measure programs with OO concepts. Even though there are metrics that already considered this subject, most of these metrics only measure a few attributes and elements regarding OO concepts. This denotes that a single metric will not be enough to measure all the qualities of object-oriented program design. The purpose of this research is to introduce a software metric specifically for software programs which will cover multiple aspects of Object-Oriented design. This metric aims to address the above-mentioned challenges.

On average around 80% - 90% of the annual cost of the software life cycle would be spent on the software maintenance phase [12]. There is a fundamental relationship between software complexity and software maintenance. With the proper application of software complexity metric, this huge expenditure on maintenance can be greatly reduced. With the help of complexity measures a proper understanding of the quality and the reliability of software could be enhanced and the proper controls on complexity will enhance many operations related to maintenance. [6] [8] [2] [13].

One of the metrics that is concerned with several features of Object-oriented concepts at a time is Chhillar and Bhasin's Cognitive Based (CB) Measure named Weighted Composite Complexity (WCC) Measure [8]. This metric will quantify four factors of a program which are inheritance, control structure, nesting level, and the size of the program. In this metric specific weight would be allocated for the above factors. The objective of this research will be to introduce and develop a tool named "Codeplex" based on this WCC alongside improvements for this. As it suggests enhancing the measure it will be named after Enhanced Cognitive Based (ECB) Measure.

In summary, this paper will discuss the importance of software complexity and software metrics as well as the interrelationship between these metrics and the maintenance of software as it will suggest a new method as well as develop a tool based on ECB to calculate, analyze code, and support decision-making related to software development phase in the hope of improving the maintainability and reliability of a software product.

II. LITERATURE REVIEW

The very first attempt to identify the complexity of code was Wolverton in 1974 by introducing Line of Code (LOC) which measures the production ratio of programmers [13]. In 1976, McCabe introduced cyclomatic complexity which is a mathematical technique to identify software complexity based on control flows in software modules [4]. He introduced a unique name, a cyclomatic number that represents a maximum number of linear paths in the control flow in terms of cognitive weights. These cognitive weights are measured based on the extent to which the software program is difficult to comprehend.

In 1977 Halstead introduced another set of metrics which was also improving LOC [3]. Halstead's metrics are used to estimate the size, complexity, and effort requirement of the software. However, it was later found out that these metrics have limited scope, limited applicability, and limited accuracy with the increase of the complexity of a program. [14]. These metrics will be based on the number of operators and operands in the program.

In 1981, Henry Kafura introduced a complexity measure named fan-in-fan-out where fan-in is the number of information flowing into a program module and fan-out is the number of information flowing out of a program module.

In 1994, Chdamber and Kamerer [5] introduced a new suite of metrics that is very famous for measuring the complexity of object-oriented design. This includes 6 different metrics namely:

- Weighted Methods per Class (WMC)
- Depth of Inheritance Tree (DIT)
- Number of children (NOC)
- Coupling between object classes (CBO)
- Response for a Class (RFC)
- Lack of Cohesion in Methods (LCOM)

In 1997, Another metric by T.Mayer and T.Hall introduced an improved version of the MOOD metric. It covers the following factors namely:

- Method Hiding Factor (MHF)
- Attribute Hiding Factor (AHF)
- Method Inheritance Factor (MIF)
- Attribute Inheritance Factor (AIF)
- Polymorphism Factor (PF)
- Coupling Factor (CF)

In 2000, Etzkom and Delugach [15] introduced another tool for object-oriented design where complexity measures are done for the factors such as the class cohesion, class interface complexity, class overlap. The metric was named Logical Relatedness of Methods

Peer Reviewed & Refereed Journal
Volume-13, Issue-5 (October 2023)
https://doi.org/10.31033/ijemr.13.5.8

(LORM). In 2005 Costagliola and Tortura [7] introduced a functional point- like approach named class point which also supports estimating the size of Object-oriented products.

In 2007 Misra and Sanjay [2] introduced complexity metrics which is specifically focused on class in the method level which is an attempt to identify and evaluate methodsof Object-oriented design by considering the internal architecture of the class.

In 2011, Chhillar and Bhasin introduced the WCC measureand in 2015, Arbi and Gahazarian [10] introduced a universal complexity theory called Informational Volumetric Complexity (IVC).

III. PROPOSED ENHANCED COGNITIVE BASED COMPLEXITY MEASURE

According to Chhillar and Bhasin's WCC measure, different cognitive-based weights are allocated for fourdifferent factors. The weight allocation of WCC is as follows:

Inheritance Level (W_i): The deeper the inheritance is, the harder it is to understand. Therefore, it affects complexity. For the WCC measure, difference weights will be allocated considering the class level. For example, Base Class: In the case of a base class, it is assigned a weight of 0, indicating the lowest complexity level. First Derived Class from Base Class: When a class is in first level of derivation from the base class, it is given a weight of 1, signifying a moderate level of complexity Second Derived Class: For a class that is two levels deepin the inheritance hierarchy, it is allocated a weight of 2, indicating a higher level of complexity.

Types of Control Structures (W_c): A program with diverse control structures is difficult to comprehend and different control structures contribute complexity in different ways. In WCC measures weights will be allocated considering the type of control structure. Sequential statements, which represent a linear flow of code execution, are assigned a weight of 0. Conditional statements, which introduce decision-making logic into the code, are allocated a weight of 1. Iterative statements, responsible for creating loops and repetitive execution, receive a weight of 2. Switch Cases with n Statements (W_c): For switch cases with 'n' individual statements inside, the weight is directly set to 'n.'

Nesting of Control Structures (W_n): A program with more levels of nesting in the control structure is difficult to comprehend, thus it will also affect the program's complexity. Sequential statements are assigned a weight of 0. Control structures and statements at the outermost level of nesting are assigned a weight of 1. Control structures and statements at the innermost levels of nesting are assigned a weight of 2.

Size (S_j): Since the start of this software complexity came to be size is also considered one of the parameters of software complexity. A class with more methods, operators, and operands are way more difficult to understand than a simple program with few statements. Therefore, size is obviously a factor that affects the complexity. In terms of WCC total number of tokens (operators, operands, strings, and methods/functions) will be considered as the weight for the size factor.

By considering the above factors following WCC measure introduces the following formula to calculate total weight of the program.

$$C_w(P) = \sum_{j=1}^{n} (S_j)^* (W_t)_j \qquad 1$$

C_w (P) = Total weight for the program

Pn = Total number of executable statements in program P,

W_t = Total weight of j^{th} executable statement in program P,

$W_t = W_n + W_i + W_c$

Except for the above 4 factors, another four factors will beconsidered in the introduced ECB measures. They will be as follows.

Multiple Inheritance: Class with multiple inheritance will be difficult to comprehend thus contributing to the complexity of the program. This was not considered when measuring the complexity in the WCC metric. In this ECB, additional weight will be allocated for classes with multiple inheritance. For each class that is derived from more than one class weight of one will be allocated to W_i.

Compound Conditional Statements: When it comes to conditional statements, there are two types namely simple and compound conditions. In the WCC metric, this point was not considered. In ECB, additional weight will be allocated for conditions with compound conditional statements. For each compound conditional statement, an additional one unit of weight will be allocated for W_c.

Methods with Multiple Statements: When it comes to measuring the size of the methods WCC does not measurethe difference between two methods which have a considerable number of statements than the other. A function with a greater number of statements with complicated steps would be more difficult to comprehend than the simple method with few steps. Therefore, by considering the number of statements in each method different weights will be allocated in ECB measure. For a function with more than 5 statements, an additional 1 unit of weight will be allocated. For an example method with 5 statements will be given 1 point. The method with 10

Peer Reviewed & Refereed Journal
Volume-13, Issue-5 (October 2023)
https://doi.org/10.31033/ijemr.13.5.8

statements will be given 2 points. Likewise, additional tokens will be added to S_j.

Array and Object Declaration: WCC has also not commented on anything on array and object declaration as well. Though it might be less complex compared with other factors it also poses a complexity to the program to a certain extent. In ECB measure, this will be also considered when measuring the complexity. An array declaration will be considered as normal variable declaration and due to the uniqueness of an array data structure additional points will be allocated for the S_j. The object declaration will also be considered the same as a normal variable declaration which was also not mentioned in WCC.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

The main objective of this paper is to identify the places that can be improved within the CB measure dons in WCC. As mentioned in the Section III authors have identified 4 different factors that can be improved in WCC. Those are multiple inheritance, array and object declaration, compound conditional statements and functions with multiple statements. The weights will be allocated according to the methodology and a calculation has been done separately for each WCC and ECB. *Table 1, Table 3, Table 5, Table 7* are examples measured according to the WCC, and *Table 2, Table 4 Table 6, and Table 8* are examples measured according to the ECB.

According to the above-mentioned methodology, weights will be allocated for considering the inheritance level of the program. Additionally, on ECB multiple inheritance will be considered when measuring the complexity.

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  class Shape {
6  public:
7
8      void printType() {
9          cout << "Shape Type: "<< endl;
10     }
11 };
12
13 class Color {
14 public:
15     void printColor() {
16         cout << "Color: "<< endl;
17     }
18 };
19
20 class ColoredShape : public Shape, public Color {
21 public:
22     void printInfo() {
23         printType();
24         printColor();
25     }
26 };
27
28 int main() {
29     ColoredShape myShape;
30     myShape.printInfo();
31     return 0;
32 }
```

Figure 1: Sample program to demonstrate the effect of Inheritance

Table 1: CB for Result Program for multiple inheritance

Line	S	W_n	W_i	W_c	W_t	S*W
8	2	0	1	0	1	2
9	5	0	1	0	1	5
15	2	0	1	0	1	2
16	5	0	1	0	1	5
22	2	0	2	0	2	4
23	1	0	2	0	2	2
24	1	0	2	0	2	2
28	2	0	0	0	0	0
29	2	0	0	0	0	0
30	3	0	0	0	0	0
31	1	0	0	0	0	0
WCC CB value						**22**

But in here notice than in *Table1*, line 9, class ColoredShape is an example with multiple inheritance. In WCC this aspect was not considered. But the authors decided to include this for measuring ECB. The newly calculated values are listed below. (*Table2)*

Peer Reviewed & Refereed Journal
Volume-13, Issue-5 (October 2023)
https://doi.org/10.31033/ijemr.13.5.8

Table 2: ECB for Result Program for multiple inheritance

Line	S	W_n	W_i	W_c	W_t	S*W
8	2	0	1	0	1	2
9	5	0	1	0	1	5
15	2	0	1	0	1	2
16	5	0	1	0	1	5
22	2	0	3	0	3	6
23	1	0	3	0	3	3
24	1	0	3	0	3	3
28	2	0	0	0	0	0
29	2	0	0	0	0	0
30	3	0	0	0	0	0
31	1	0	0	0	0	0
ECB value						**26**

Assigning a weight of one to control structure attributes (W_c) whenever the "&&" or "||" operator is used. The impact of compound situations on programming intricacy is shown through an example of C++ code presented in *Figure 2*. The program's careful design highlights the complexities of measuring simplicity when distinct control systems are implemented. Through *Tables 3* and *Table 4*, the subsequent examination presents a detailed and straightforward depiction of how software complexity is measured for the code displayed in *Figure 2*.

```
#include <iostream>

class SampleCode {
public:
    // Method with compound condition (&&)
    void methodWithCompoundCondition(int x, int y) {
        if (x > 0 && y > 0) {
            std::cout << "Both x and y are positive." << std::endl;
        } else {
            std::cout << "At least one of x or y is not positive." << std::endl;
        }
    }
};

int main() {
    SampleCode sample;
    sample.methodWithCompoundCondition(2, 4);
    return 0;
}
```

Figure 2: Sample program to demonstrate the effect of compound conditions

Table 3: CB for Result Program for compound conditional statements

Line	S	W_n	W_i	W_c	W_t	S*W
3	0	0	0	0	0	0
6	2	0	1	0	1	2
7	8	1	1	1	3	24
8	5	1	1	0	2	10
9	0	0	0	0	0	0
10	5	1	1	0	2	10
15	2	0	0	0	0	0
16	2	0	0	0	0	0
17	3	0	0	0	0	0
WCC CB Value						46

Table 4: ECB for Result Program for compound conditional statements

Line	S	W_n	W_i	W_c	W_t	S*W
3	0	0	0	0	0	0
6	2	0	1	0	1	2
7	8	1	1	2	4	32
8	5	1	1	0	2	10
9	0	0	0	0	0	0
10	5	1	1	0	2	10
15	2	0	0	0	0	0
16	2	0	0	0	0	0
17	3	0	0	0	0	0
ECB Value						54

Complexity that array, and object declarations in a program bring out. To assess software complexity, it is crucial to comprehend how these declarations impact program structure and behavior. This problem is addressed and a nuanced understanding of program complexity in realistic software development contexts is provided by the suggested ECB measure.

Peer Reviewed & Refereed Journal
Volume-13, Issue-5 (October 2023)
https://doi.org/10.31033/ijemr.13.5.8

```
class MyClass {
public:
    MyClass(int val); // Constructor declaration
    void PrintValue();
    void DerivedClass(); // Declare the DerivedClass function
private:
    int value;
};

// Constructor definition
MyClass::MyClass(int val) {
    value = val;
}

void MyClass::PrintValue() {
    std::cout << "Value: " << value << std::endl;
}

// Define the DerivedClass function outside the class
void MyClass::DerivedClass() {
    int myArray[5] = {1, 2, 3, 4, 5};
    MyClass myObject(42);
    myObject.PrintValue();
}
```

Figure 3: Sample program to demonstrate the effect of Object array declaration

Table 5: CB for Result Program for object and array declaration

Line	S	W_n	W_i	W_c	W_t	S*W
7	2	0	0	0	0	0
11	3	0	1	0	1	3
12	3	0	1	0	1	3
15	4	0	1	0	1	4
16	7	1	1	0	2	14
20	4	0	1	0	1	4
21	4	0	1	0	1	4
22	0	0	1	0	0	0
23	3	0	1	0	1	3
WCC CB Value						**35**

But here notice that in Table 6, line 21 and 22, an array and object declaration can be found. In WCC this aspect was not considered. But authors found those can be attributed to increasing the code complexity. Therefore, additional points are allocated in S_j in *Table 6*.

Table 6: ECB for Result Program for object and array declaration

Line	S	W_n	W_i	W_c	W_t	S*W
7	2	0	0	0	0	0
11	3	0	1	0	1	3
12	3	0	1	0	1	3
15	4	0	1	0	1	4
16	7	1	1	0	2	14
20	4	0	1	0	1	4
21	5	0	1	0	1	5
22	2	0	1	0	1	2
23	3	0	1	0	1	3
WCC CB Value						**38**

When considering the Methods with Multiple Statements feature, the ECB software complexity measure produces intriguing outcomes. Within methods, different statement counts lead to cognitive disparities that WCC does not identify. The introduction of weighting according to statement count increases ECB calculation. Methods featuring abundant statements, complex control flow, logical conditionals, and multiple operations tend to receive better ECB ratings.

```
class ComplexityCalculator {
public:
    int methodWithOneStatement() {
        int x = 10;
        return x;
    }
    int methodWithMultipleStatements() {
        int total = 0;
        for (int i = 1; i <= 5; i++) {
            total += i;
        }
        return total;
    }
};

int main() {
    ComplexityCalculator calculator;
    int sizeComplexity = 0;
    sizeComplexity += countTokens(calculator.methodWithOneStatement());
    sizeComplexity += countTokens(calculator.methodWithMultipleStatements());
```

Figure 4*: Sample program to demonstrate the effect of Methods with Multiple Statements*

Peer Reviewed & Refereed Journal
Volume-13, Issue-5 (October 2023)
https://doi.org/10.31033/ijemr.13.5.8

Table 7: CB for Result Program for methods with multiple statements

Line	S	W_n	W_i	W_c	W_t	S*W
3	2	0	1	0	1	2
4	4	0	1	0	1	4
5	1	0	1	0	1	4
7	2	0	1	0	1	2
8	4	0	1	0	1	4
9	10	1	1	2	4	40
10	4	1	1	2	4	16
12	1	0	1	0	1	1
16	2	0	0	0	0	0
17	0	0	0	0	0	0
18	4	0	0	0	0	0
19	7	0	1	0	1	7
20	6	0	1	0	1	6
WCC CB Value						**86**

But here notice that in Table 7 line 20, class Complexity Calculator is an example with multiple statements. In WCC this aspect was not considered and allocated the same weights for both methods with one and multiple statements inside. But authors found that methods with multiple statements might be more complex and allocated more weight for that. The allocation is done as one more weight.

Table 8: ECB for Result Program for methods with multiple statements

Line	S	W_n	W_i	W_c	W_t	S*W
3	2	0	1	0	1	2
4	4	0	1	0	1	4
5	1	0	1	0	1	4
7	2	0	1	0	1	2
8	4	0	1	0	1	4
9	10	1	1	2	4	40
10	4	1	1	2	4	16
12	1	0	1	0	1	1
16	2	0	0	0	0	0
17	0	0	0	0	0	0
18	4	0	0	0	0	0
19	7	0	1	0	1	7
20	7	0	1	0	1	7
ECB Value						**87**

V. CONCLUSION

In conclusion, our research has resulted in the development of the Enhanced Cognitive Based (ECB) measure, a tool made to satisfy the requirements of developers and technical leads. A more advanced and enhanced version of the conventional Weighted Code Complexity (WCC) metric is represented by ECB. It is a useful asset since it provides increased explanatory power. Compared to traditional code complexity measures, ECB provides a higher level of accuracy and dependability based on object, array declarations, multiple inheritance, compound conditional statements, methods and multiple statements.

Instead of just creating another generic metric calculator, our aim was to pioneer a revolutionary approach that addresses the endemic problems in software engineering. With the ECB measure, we offer a potent tool that enables professionals to improve the quality and efficiency of their software development projects, streamline their coding procedures, and make informed judgments.

REFERENCES

[1] "Cambridge Dictionary," Cambridge University, [Online]. Available: https://dictionary.cambridge.org/dictionary/english/complexity?q=complexity.. [Accessed 25 08 2023].

[2] Misra Sanjay. (2007). An object irented complexity metric based on cognitive weights. *6th IEEE International Conference on Cognitive Informatics*.

[3] H.Halstead. (1977). *Elements of of software science*.

[4] T.J.McCabe. (1976). A complexity measure. *IEEE Transaction on Software Engineering*.

[5] S.R.Chidamber & C.F. Kamerer. (1994). A metric suite for obect oriented design. *IEEE Transactions on Software Engineering, 20*(6), pp. 476-493.

[6] Software structure metrics based on software engineering. (1981). *IEEE Transaction Metrics Bases on Information Flow, SE-7*(5), pp. 510-

518.

[7] G. Costagliola, F. Ferrucci, G. Tortora & G. Vitiello. (2005). Class point: an approach for the size estimation of object-oriented systems. *IEEE Transactions on Software Engineering*, *31*(1), pp. 52-74.

[8] Usha Chhillar & Shuchita Bhasin. (2011). A new weighted composite complexity measure for object-oriented systems. *International Journal of Information and Communication Technology Research*.

[9] T. Mayer & T. Hall. (1999). Measuring OO systems: A critical analysis of the MOOD metrics. In: *Proceedings Technology of Object-Oriented Languages and Systems. TOOLS 29 (Cat. No.PR00275)*.

[10] Gahzarian, Arbi. (2015). A theory of software complexity. In: *IEEE/ACM 4th SEMAT Workshop on a General Theory of Software*.

[11] E. Weyuker. (1988). Evaluating software complexity measures. *IEEE Transactions on Software Engineering, 14*, pp. 1357-1365.

[12] A.Kushnir. (2011). "bamboo agile," Bamboo Group, 11 10 2011. [Online]. Available: https://bambooagile.eu/insights/software-maintenance- costs/. [Accessed 11 10 2023].

[13] Tu Honglei, Sun Wei & Zhang Yanan. (2009). The research on software metrics and software complexity metrics. In: *International Forum on Computer Science- Technology and Applications, Chongqing, China*.

[14] Software Engineering | Halstead's Software Metrics | GeeksForGeeks, "GeeksForGeeks," 11 07 2023. [Online]. Available: https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/. [Accessed 23 10 2023].

[15] L. Etzkorn & H.Delugach. (2000). Towards a semantic metrics suite for object-oriented design. In: *34th International Conference on Technology of Object-Oriented Languages and Systems*.

[16] Dipti Pawade, Devansh J.Dave, & Aniruddha Kamath. (2016). Exploring software complexity metric from procedure oriented to object oriented. In: *International Conference - Cloud System and Big Data Engineering*.

[17] R. Harrison, S. Counsell & R. Nithi. (1997). An overview of object- oriented design metrics. In: *Eighth IEEE International Workshop on Software Technology and Engineering Practice incorporating Computer Aided Software Engineering*.

[18] P.Ashok Reddy, Dr.K.Rajasekhara Rao & Dr.M.Babu Reddy. (2015). Performance evaluation of procedural metrics and,. *International Journal of Research Studies in Computer Science and Engineering (IJRSCSE), 2*, 69-72.

[19] D. I. De Silva, N. Kodagoda, S. R. Kodituwakku & A. J. Pinidiyaarachchi. (2015). Improvements to a complexity metric: CB measure. In: *IEEE 10th International Conference on Industrial and Information Systems (ICIIS), Peradeniya*.

[20] D. I. De Silva, N. Kodagoda, S. R. Kodituwakku & A. J. Pinidiyaarachchi. (2015). Limitations of an object-oriented metric: Weighted complexity measure. In: *6th IEEE International Conference on Software Engineering and, Beijing, China*.

[21] D. I. De Silva, N. Kodagoda, S. R. Kodituwakku & A. J. Pinidiyaarachchi. (2017). Analysis and enhancements of a cognitive based,. In: *IEEE International Symposium on Information Theory (ISIT), Aachen, Germany*.

[22] D. I. D. Silva. (2016). Analysis of weighted composite complexity measure. In: *International Conference on Computational Techniques in*, New Delhi, India.

[23] D. I. De Silva, S. R. Kodituwakku, A. J. Pinidiyaarachchi & N. Kodagoda. (2018). Enhancements to an OO Metric: CB Measure. *Journal of Software, 13*, 72-81.

[24] Hansini M. Fernando, Damith R. Kothalawala, Dilshan I. De Silva & Nuwan Kodagoda. (2012). Automated code analyser,. In: *Proc. IASTED International Conference on Engineering and Alied Science(EAS), Colombo, Sri Lanka*.