A New Metric based on the Weighted Class Complexity (WCC) metric and measure of complexity for Object-Oriented Systems

D. I. De Silva
Department of Information Technology
Faculty of Computing
Sri Lanka Institute of Information Technology
New Kandy Road, Malabe, Sri Lanka
dilshan.i@sliit.lk

W.A.C.Pabasara
Department of Computer Science & Software Engineering
Faculty of Computing
Sri Lanka Institute of Information Technology
New Kandy Road, Malabe, Sri Lanka
chamali.p@sliit.lk

P.G.T Dilmith
Department of Information Technology
Faculty of Computing
Sri Lanka Institute of Information Technology
New Kandy Road, Malabe, Sri Lanka
it20131388@my.sliit.lk

B.M.G Peiris
Department of Information Technology
Faculty of Computing
Sri Lanka Institute of Information Technology
New Kandy Road, Malabe, Sri Lanka
it20147396@my.sliit.lk

P.A.D.S.D Dilshan
Department of Information Technology
Faculty of Computing
Sri Lanka Institute of Information Technology
New Kandy Road, Malabe, Sri Lanka
it20178154@my.sliit.lk

G.W.C.D Jayathilaka
Department of Information Technology
Faculty of Computing
Sri Lanka Institute of Information Technology
New Kandy Road, Malabe, Sri Lanka
it20003128@my.sliit.lk

***Abstract*— Class-level complexity is a significant component of software complexity, and measuring software complexity is essential for software development and maintenance. Although the Weighted Class Complexity (WCC) metric has a long history of use, it has some drawbacks, including the equal weighting of complexity components and the exclusion of some complexity elements,. In this paper, we present a novel metric, the Enhanced Weighted Class Complexity (EWCC) metric, which extends the WCC metric by adding more complexity variables and weighting them according to their relative relevance. Data access patterns, method call chains, and control structure complexity are among the other variables. In order to compare the EWCC metric's output to the WCC metric's output, we apply it to a number of software systems. We assess the EWCC metric's performance in locating potential error and bug sources. The findings indicate that, compared to the WCC metric, the EWCC metric offers a more thorough and precise evaluation of class-level complexity. According to the results, software developers and maintainers may find the EWCC measure to be a useful tool for controlling software complexity.**

Keywords*— *Enhanced Weighted Class Complexity, Weighted Class Complexity, software complexity, metric, class-level complexity

I. Introduction

Software creation and maintenance must take into account the complexity of the software. The complexity of software systems makes it more challenging to comprehend, alter, and maintain them. Given that classes serve as the foundation for software systems, class-level complexity is a crucial component of software complexity. A tried-and-true technique for gauging class-level complexity is the Weighted Class Complexity (WCC) metric[6]. The WCC measure, however, has some drawbacks, including the equal weighting of complexity elements and the exclusion of some complexity factors[1]. In this paper, we offer the Enhanced Weighted Class Complexity (EWCC) measure, a novel metric for assessing class-level complexity in software systems. In order to improve upon the WCC metric, the EWCC metric adds more complexity elements and weights them according to how important a role they play in class-level complexity.

In this study, a variety of software systems will be subjected to the EWCC metric, and the outcomes will be compared to

those attained using the WCC meter [4],[2]. Additionally, we'll assess how well the EWCC metric works for locating probable sources of faults and mistakes in software systems. The findings of this study will shed important light on the utility of the EWCC metric in class-level complexity measurement as well as its potential advantages for software development and maintenance. The EWCC metric's extra complexity variables are intended to offer a more thorough and accurate evaluation of class-level difficulty. These elements consist of method call chains, data access patterns, and the complexity of control structures within a class.

II. Literature Review

The quality, maintainability, and evolution of software systems are significantly impacted by software complexity. To aid developers and maintainers in managing software complexity, a number of metrics have been presented over the years. The Weighted Class Complexity (WCC) metric, which assesses the complexity of individual classes in object-oriented programming, is one of the extensively used metrics. Different complexity parameters, such as the quantity of methods, instance variables, and statements, are given weights by the WCC metric. The WCC measure has been criticism for its shortcomings, including the fact that it does not take into consideration complex control structures, method call chains, or data access patterns.To get around these restrictions, a number of academics have suggested WCC metric expansions and alterations. For instance, the Object-Oriented Metrics Suite (OOMS), which includes a number of complexity metrics including the Weighted Method per Class (WMC) and the Depth of Inheritance Tree (DIT), was proposed by Bansiya and Davis in 2002. Similar to this, El Emam and Benlarbi (2001) presented the Number of Hierarchies (NOH), Number of Polymorphic Methods (NPM), and Weighted Method Complexity (WMC) metrics as part of the Object-Oriented Complexity and Design Metrics (OODM) metric suite.Recent studies have suggested more complicated measurements that go beyond the WCC meter and incorporate further complexity variables. For instance, the Weighted Dynamic Method Call Chain Complexity (WDMCCC) measure, which takes into account the complexity of method call chains during runtime, was suggested by Liu et al. (2018). The Weighted Control Flow Complexity (WCFC) metric, which takes into account the complexity of control flow structures such loops and conditional statements, was also suggested by Zhang et al. in 2020.The Enhanced Weighted Class Complexity (EWCC) measure, which expands upon the Weighted Class Complexity (WCC) meter by incorporating new complexity characteristics such data access patterns, method call chains, and control structure complexity, is proposed in this study. We weigh these variables according to their respective relevance, which enables us to offer a more detailed and precise evaluation of class-level difficulty. Although our suggested measure is conceptually similar to the OOMS and OODM metric suites, it differs in that it places more focus on class-level complexity and incorporates dynamic runtime data.Overall, the body of research on software complexity metrics offers a wide range of methods for evaluating program complexity. By offering a more thorough and accurate evaluation of class-level complexity, our suggested EWCC metric contributes to this body of knowledge and can aid developers and maintainers in better managing program complexity.

A. *Key Factors and how they affect the complexity of a program*

1) *Token size*

The smallest individual components, or "tokens," are also referred to as the "building blocks" of a Java program. A program in Java is made up of classes and methods, with methods being made up of different expressions and statements. The tiny pieces of code called tokens in Java are what a Java compiler employs to create those statements and expressions. Java supports the following 5 types of tokens: They are words that the Java compiler has predefined or reserved special meaning for.

- Keywords: are words that have been predefined or designated as reserved by the Java compiler.
- Identifiers: are names that have been given to variables, methods, classes, arrays, packages, and interfaces by the user.
- Literals: In Java, literals are comparable to regular variables, except once their values have been assigned, they cannot be modified. They have fixed values and are constant variables.
- Operators: A special symbol that instructs the compiler to carry out a particular mathematical or non-mathematical operation on one or more operands is known as an operator.
- Special Symbols: These are a small number of characters that have meanings unique to the Java compiler and which are not usable for other purposes. To better comprehend the code segment, it is helpful to identify it, analyze it, and determine the token size.

2) *Type of control structures*

Type of control structure	Weight
Sequential	0
Branch	1
Iterative	2
Switch statement with n cases	n+1

3) *Nesting level of control structures*

Nesting Level of Statements	Wight
Sequential statements	0
Statements inside the outer most level/first level of control structures	1
Statements inside the second level control structures	2
Statements inside the second level control structures	3
Statements inside the nth level control structures	n

4) *Inheritance level of statements*

Inheritance Level of Statements	Weight
Statements inside the base class/root class	0
Statements inside the first derived class	1
Statements inside the second derived class	2
Statements inside the nth derived class	n

$$\text{WCC} = \sum_{j=1}^{n} S_j * (Wt\ )_j$$

S_j = Size of the executable statement in terms of token count

n = Total number of executable statements in the program

$(Wt\ )_j$ = Total weight of the j^{th} executable statement in the program

$$Wt\ = Wc\ + Wn\ \ + \ Wi$$

Wt = Weight due to Total

Wc = Weight Due to Type of Control Structures

Wn = Weight Due to Nesting Level of Control Structures

Wi = Weight Due to Inheritance Level of Statements

III. Introduce the New Metric based on measure of complexity for Object-Oriented Systems.

The **Enhanced Weighted Class Complexity (EWCC)** is a new metric for measuring the complexity of individual classes within a software system. The metric is based on the existing Weighted Class Complexity (WCC) metric but includes additional weighted components to provide a more comprehensive and accurate assessment of software complexity. The EWCC metric's additional weighted components now take into account things like class-specific control structure complexity, method call chains, and data access patterns. The EWCC metric can provide a more thorough and nuanced assessment of software complexity by including these elements and accounting for the numerous ways in which complexity might develop within a class. Each component is given a weight in the EWCC measure based on how important they are in terms of adding to the total class complexity. The final difficulty score for each class is then computed by averaging the weighted values. Developers can use the EWCC metric to assess class-level complexity more precisely and comprehensively, which can aid in making knowledgeable choices about software design, testing, and maintenance. In the end, employing the EWCC measure can lead to software systems of a higher caliber that are simpler to manage and less prone to errors and problems.

A. *Why "Enhanced Weighted Class Complexity" (EWCC) is an appropriate for your new metric:*

The term reflects the fact that you are adding new weighted components to the weighted class complexity metric that already exists. You are improving the existing metric and provide a more precise and thorough assessment of software complexity by including these new elements.

The word "enhanced" underlines how much better the new statistic is than the previous one. The EWCC metric can better depict the complexities that emerge in real-world software systems by including extra characteristics like data access patterns and method call chains. The term "weighted" indicates that the new metric considers the relative significance of several facets of software complexity. The EWCC metric offers a more complicated and in-depth evaluation of software complexity by giving weights to many aspects. The word "class" denotes that the metric is concentrated on specific classes within of a software system. This is consistent with the primary WCC metric, which emphasizes class-level complexity as well. The metric is intended to assess the complexity of software systems, as indicated by the word "complexity" in the name. The EWCC metric can assist developers in making better decisions regarding software design, testing, and maintenance, ultimately leading to higher quality software systems, by offering a more accurate and thorough assessment of software complexity. The term "Enhanced Weighted Class Complexity" highlights the significance of your new metric as an advancement over the current metric of weighted class complexity and appropriately describes the aim and scope of your new metric.

B. *Advantages of EWCC comparing WCC*

A more thorough evaluation of class-level complexity is provided by the EWCC measure, which includes weighted components for data access patterns, method call chains, and the complexity of control structures within a class that are not included in the WCC metric. As a result, a more thorough evaluation of class-level complexity is produced, which can aid developers in more precisely locating potential fault and error sources.

The EWCC metric applies weights to each of the components based on their relative importance in contributing to overall class complexity. This results in a more nuanced weighting of the complexity factors. Compared to the WCC measure, which gives each complexity element the same weight, this gives complexity factors a more nuanced weighting. More accurate representation of software complexity in practice: The EWCC metric's additional elements reflect the complexities that can occur in actual software systems. The EWCC metric offers a more accurate reflection of the difficulties that software engineers encounter when creating and maintaining software systems by taking into account this complexity.

Making more knowledgeable decisions The EWCC metric's more precise and thorough evaluation of class-level complexity can aid software engineers in making more educated choices regarding software design, testing, and maintenance. This may result in easier to maintain, higher quality software systems that are less prone to flaws and failures. Overall, compared to the Weighted Class Complexity (WCC) metric, the Enhanced Weighted Class Complexity (EWCC) metric offers a more precise, complicated, and thorough evaluation of class-level complexity. This can assist programmers in creating longer-lasting, more dependable, and higher-quality software systems.

IV. How the New Metric Captures the Complexity Introduced by Each Factor

$$\textbf{EWCC} = \sum_{j=1}^{n} S_j * (Wt_e)_j$$

S_j = Size of the executable statement in terms of token count

n = Total number of executable statements in the program

$(Wt_e)_j$ = Total Enhanced weight of the jth executable statement in the program

$$Wt_e = Wc_e + Wn_e + Wi_e$$

Wt_e = Enhanced Weight due to Total

Wc_e = Enhanced Weight Due to Type of Control Structures

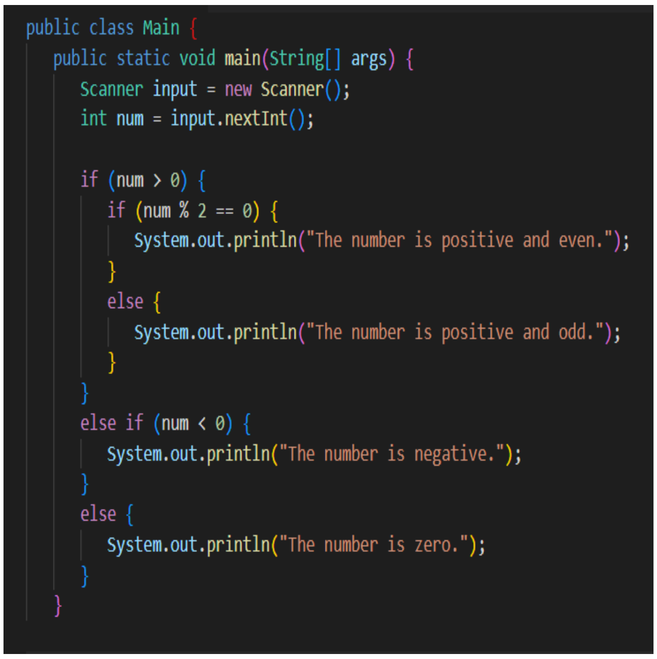
Wn_e = Enhanced Weight Due to Nesting Level of Control Structures

Wi_e = Enhanced Weight Due to Inheritance Level of Statements

V. METHODOLOGY

Test Program 1

First, Calculate the WCC values using Standard values using example 1 java code:



```java
public class Main {
    public static void main(String[] args) {
        Scanner input = new Scanner();
        int num = input.nextInt();

        if (num > 0) {
            if (num % 2 == 0) {
                System.out.println("The number is positive and even.");
            }
            else {
                System.out.println("The number is positive and odd.");
            }
        }
        else if (num < 0) {
            System.out.println("The number is negative.");
        }
        else {
            System.out.println("The number is zero.");
        }
    }
```

Fig. 1 . Test Program 1

Factors:

- In IF statements count the $[Wc_e]$ Enhanced Weight Due to Type of Control Structures as 0.50
- In IF statements count the $[Wn_e]$ Enhanced Weight Due to Nesting Level of Control Structures as 0.50
- In IF statements count the $[Wi_e]$ Enhanced Weight Due to Inheritance Level of Statements as 1.509.

In the very first got WCC value using Standard Wc , Wn , Wi values.

Wc	Branch Sequential	1 0
Wn	Statements inside the outer most level/first level of control structures (Branch)	1
Wi	Statements inside the first derived class	1

After that, we got three Factors:

- In IF statements count the $[Wc_e]$ Enhanced Weight Due to Type of Control Structures as 0.50
- In IF statements count the $[Wn_e]$ Enhanced Weight Due to Nesting Level of Control Structures as 0.50
- In IF statements count the $[Wi_e]$ Enhanced Weight Due to Inheritance Level of Statements as 1.509
- By using those three enhanced weights able to get value **almost equal** to this WCC values instead of standard weights (Wc = 1, Wn = 1 , Wi = 1).

Calculation

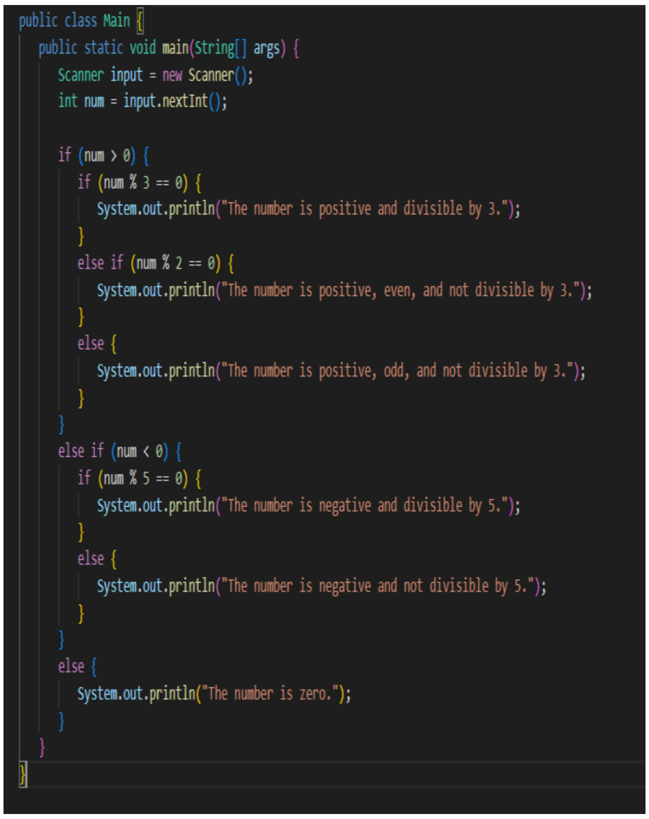
$$Wt_e = Wc_e + Wn_e + Wi_e$$

=(0+0+1.509)+(0+0+1.509)+…………+ (0+0.50+1.509)

$$\text{EWCC} = \sum_{j=1}^{n} S_j * (Wt_e)_j$$

=2*1.509 +5*1.509 +…………..+ 6*2.009
= 103.00

Some other example code of having same EWCC value



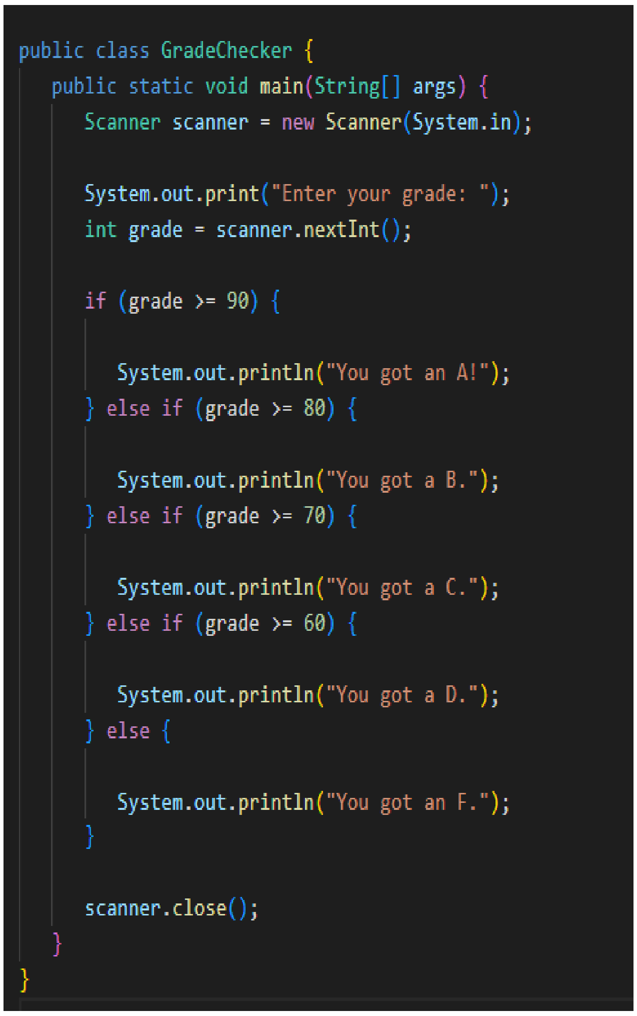
```java
public class Main {
    public static void main(String[] args) {
        Scanner input = new Scanner();
        int num = input.nextInt();

        if (num > 0) {
            if (num % 3 == 0) {
                System.out.println("The number is positive and divisible by 3.");
            }
            else if (num % 2 == 0) {
                System.out.println("The number is positive, even, and not divisible by 3.");
            }
            else {
                System.out.println("The number is positive, odd, and not divisible by 3.");
            }
        }
        else if (num < 0) {
            if (num % 5 == 0) {
                System.out.println("The number is negative and divisible by 5.");
            }
            else {
                System.out.println("The number is negative and not divisible by 5.");
            }
        }
        else {
            System.out.println("The number is zero.");
        }
    }
}
```

Fig. 2. Sample Program 1

Both of the examples in the code are built in Java and take an integer as user input using a Scanner. Both of them can provide three different results: a positive number, a negative number, and a zero. The first code adds an additional layer of nesting and raises the inheritance level of statements by using more complicated conditions in its if statements. It determines whether the input integer is even or odd and not divisible by three or positive and divisible by three but not even. Similar checks are used to determine if negative numbers are divisible by 5 or not. On the other hand, the second code's if statements use simpler criteria. The input number is checked to see if it is positive and even, positive and odd, or negative. Given that it contains fewer requirements and is simpler than the first code, this code has a lower EWCC value. The classification of an input number as positive, negative, or zero is ultimately accomplished by both codes, however the first code is more complex and has a higher EWCC value than the second code. The project's specific requirements, including performance, readability, and maintainability, would determine which of the two codes to choose.

Test Program 2

First, Calculate the WCC values using Standard values using example 1 java code:



```java
public class GradeChecker {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter your grade: ");
        int grade = scanner.nextInt();

        if (grade >= 90) {

            System.out.println("You got an A!");
        } else if (grade >= 80) {

            System.out.println("You got a B.");
        } else if (grade >= 70) {

            System.out.println("You got a C.");
        } else if (grade >= 60) {

            System.out.println("You got a D.");
        } else {

            System.out.println("You got an F.");
        }

        scanner.close();
    }
}
```

Fig. 3. Test Program 2

Factors:

- In IF statements count the $[Wc_e]$ Enhanced Weight Due to Type of Control Structures as 0.20
- In IF statements count the $[Wn_e]$ Enhanced Weight Due to Nesting Level of Control Structures as 0.60
- In IF statements count the $[Wi_e]$ Enhanced Weight Due to Inheritance Level of Statements as 1.2

In the very first we got WCC value using Standard Wc , Wn , Wi values.

Wc	Branch Sequential	1 0
Wn	Statements inside the outer most level/first level of control structures (Branch)	1
Wi	Statements inside the first derived class	1

After that, we got WCC value using new three Factors:

- In IF statements count the $[Wc_e]$ Enhanced Weight Due to Type of Control Structures as 0.20
- In IF statements count the $[Wn_e]$ Enhanced Weight Due to Nesting Level of Control Structures as 0.60
- In IF statements count the $[Wi_e]$ Enhanced Weight Due to Inheritance Level of Statements as 1.20
- By using those three enhanced weights able to get value **almost equal** to this WCC values instead of standard weights (Wc = 1, Wn = 1 , Wi = 1).

Calculation

$Wt_e = Wc_e + Wn_e + Wi_e$

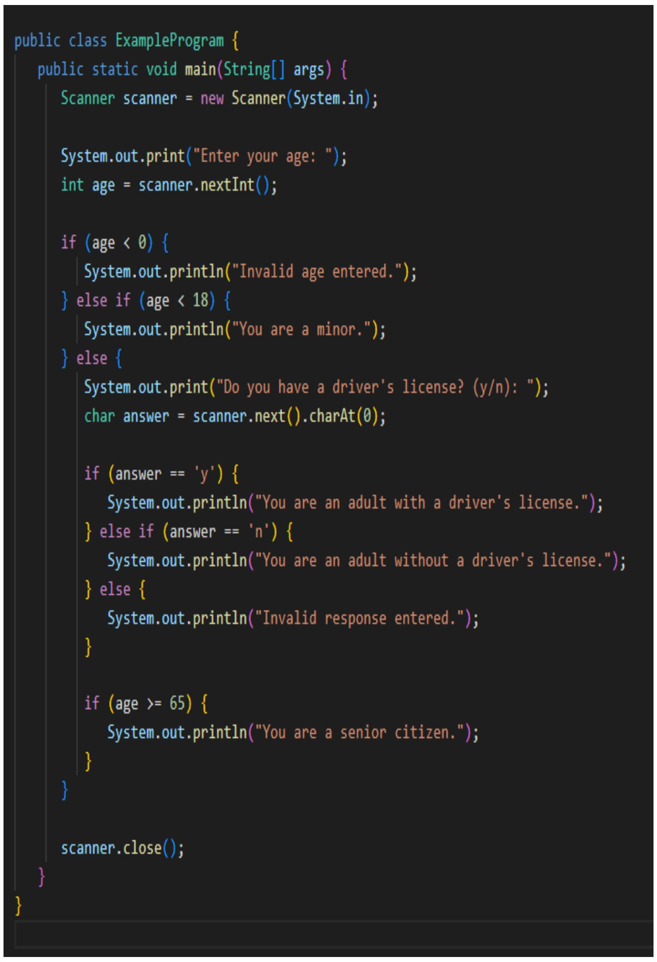
=(0+0+1.20)+(0+0+1.20)+…………+(0+0.60+1.20)

EWCC = $\sum_{j=1}^{n} S_j * (Wt_e)_j$

=2*1.20 +5*1.20 +……..+ 2*1.20

= 141.0

Some other example code of having same EWCC value.



```java
public class ExampleProgram {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter your age: ");
        int age = scanner.nextInt();

        if (age < 0) {
            System.out.println("Invalid age entered.");
        } else if (age < 18) {
            System.out.println("You are a minor.");
        } else {
            System.out.print("Do you have a driver's license? (y/n): ");
            char answer = scanner.next().charAt(0);

            if (answer == 'y') {
                System.out.println("You are an adult with a driver's license.");
            } else if (answer == 'n') {
                System.out.println("You are an adult without a driver's license.");
            } else {
                System.out.println("Invalid response entered.");
            }

            if (age >= 65) {
                System.out.println("You are a senior citizen.");
            }
        }

        scanner.close();
    }
}
```

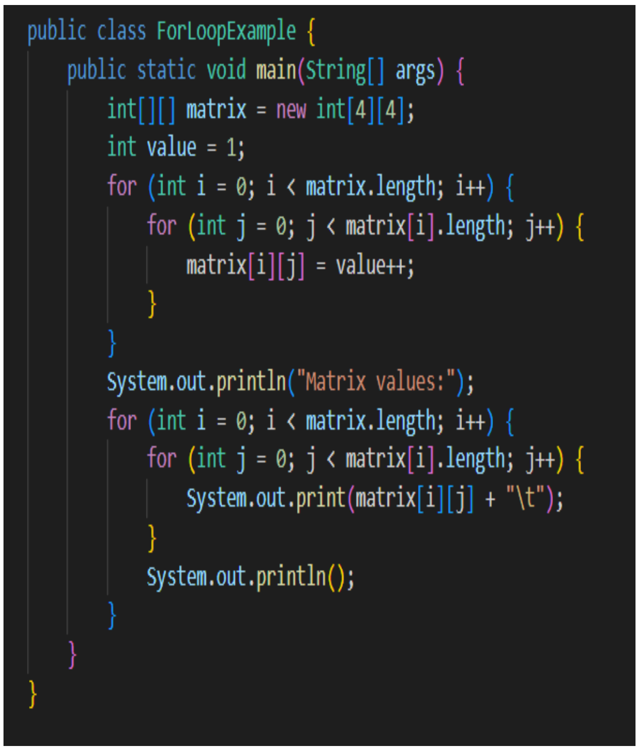
Fig. 4. Sample Program 2

It is evident from a comparison of the two code examples that they share several structural characteristics. Both programs use the Scanner class to read user input from the console and have a class declaration, a main method, and other features. The System.out.println() statement is also used by them to print messages to the console based on the result of an expression or variable. The second code, which has nested if statements and a more intricate conditional structure, is longer and more difficult than the first code, which is shorter and simpler. The first code checks the value of a user-entered grade and prints a message based on the grade range using a series of if-else expressions. The second code, in contrast, verifies the user-entered age and driver's license status and prints out various messages depending on the value combination of those two. Additional checks for incorrect age and response inputs are also included in the second code. Overall, both methods serve as illustrations of how to incorporate user input and conditional statements into Java programming; however, the second code is more intricate and has more complicated conditional structures. When the weighted complexity values of the two codes equal 141, we can say that their levels of complexity are comparable. Although the two codes' functionality and structure may differ, the weighted complexity value indicates that the two

codes are equally complex. This implies that both algorithms' complexity-increasing control structures, input/output operations, and other elements may be comparable.

Test Program 3

First, Calculate the WCC values using Standard values using example 3 java code:



```java
public class ForLoopExample {
    public static void main(String[] args) {
        int[][] matrix = new int[4][4];
        int value = 1;
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[i].length; j++) {
                matrix[i][j] = value++;
            }
        }
        System.out.println("Matrix values:");
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[i].length; j++) {
                System.out.print(matrix[i][j] + "\t");
            }
            System.out.println();
        }
    }
}
```

Fig. 5. Test Program 3

Factors:

- In IF statements count the $[Wc_e]$ Enhanced Weight Due to Type of Control Structures as 0.30
- In IF statements count the $[Wn_e]$ Enhanced Weight Due to Nesting Level of Control Structures as 0.60
- In IF statements count the $[Wi_e]$ Enhanced Weight Due to Inheritance Level of Statements as 1.590 ~1.59047

In the very first we got WCC value using Standard Wc , Wn , Wi values.

Wc	Branch Sequential	1 0
Wn	Statements inside the outer most level/first level of control structures	1

	(Branch)	
Wi	Statements inside the first derived class	1

After that, got three Factors:

- In IF statements count the [Wc_e] Enhanced Weight Due to Type of Control Structures as 0.30
- In IF statements count the [Wn_e] Enhanced Weight Due to Nesting Level of Control Structures as 0.60
- In IF statements count the [Wi_e] Enhanced Weight Due to Inheritance Level of Statements as 1.590
- By using those three enhanced weights able to get value **almost equal** to this WCC values instead of standard weights (Wc = 1, Wn = 1 , Wi = 1).

Calculation part

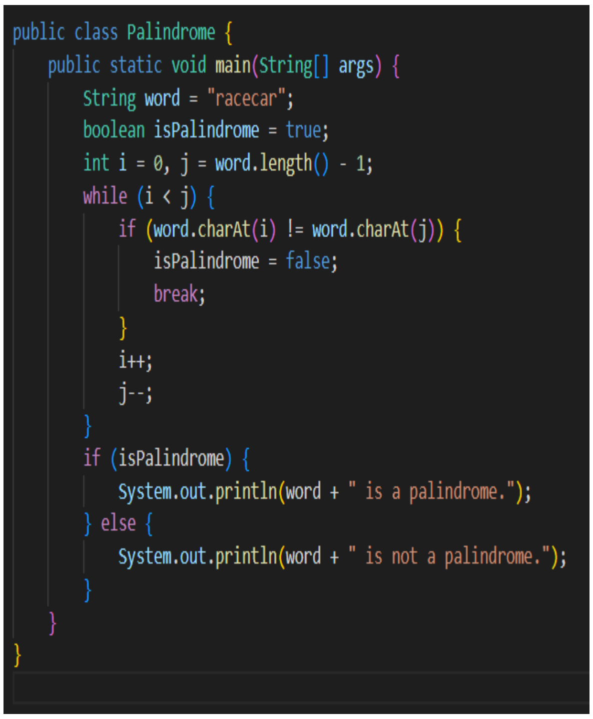
$Wt_e = Wc_e + Wn_e + Wi_e$

=(0+0+1.590)+(0+0+1.590)+…………+ (0+0.60+1.590)

EWCC = $\sum_{j=1}^{n} S_j * (Wt_e)_j$

=2*1.590+5*1.590+…………+3*2.190+…..+ 3*1.590
= 89.00

Some other example code of having same EWCC value



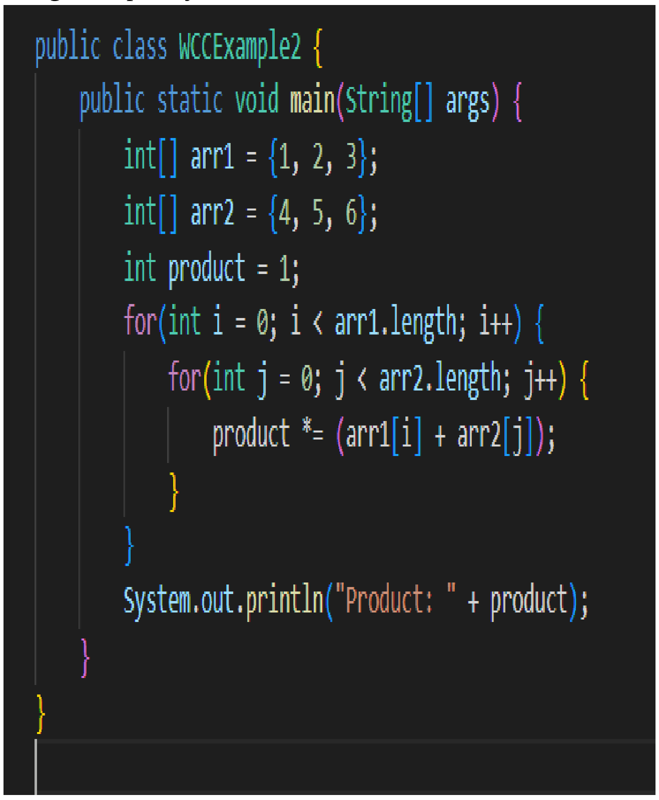
```java
public class Palindrome {
    public static void main(String[] args) {
        String word = "racecar";
        boolean isPalindrome = true;
        int i = 0, j = word.length() - 1;
        while (i < j) {
            if (word.charAt(i) != word.charAt(j)) {
                isPalindrome = false;
                break;
            }
            i++;
            j--;
        }
        if (isPalindrome) {
            System.out.println(word + " is a palindrome.");
        } else {
            System.out.println(word + " is not a palindrome.");
        }
    }
}
```

Fig. 6. Sample Program 3

The first code section is a Java program that uses nested for-loops to establish a 4x4 integer matrix and fill it with the digits 1 through 16. The matrix's contents are subsequently printed out by the application. The inner loop iterates over each row's columns while the outer loop loops over the matrix's rows. This Java code illustrates the usage of multidimensional arrays and nested for-loops. This code has an EWCC value of 16, which is divided into 4 for the outer and 4 for the inner loops. The second code section contains a Java application that detects whether or not a text is a palindrome. Two pointers, i and j, are used in this program to compare the characters at the string's start and end. The word variable in this program contains the text "racecar" as its value. If any of the characters do not match, the program ends the loop and sets the is palindrome Boolean variable to false. With the EWCC value of this code being 6 for the variable initialization, 1 for the while loop, and 3 for the if-else statement, the program ends by reporting whether or not the string is a Palindrome.

Test Program 4

First, Calculate the WCC values using Standard values using example 4 java code:



```java
public class WCCExample2 {
    public static void main(String[] args) {
        int[] arr1 = {1, 2, 3};
        int[] arr2 = {4, 5, 6};
        int product = 1;
        for(int i = 0; i < arr1.length; i++) {
            for(int j = 0; j < arr2.length; j++) {
                product *= (arr1[i] + arr2[j]);
            }
        }
        System.out.println("Product: " + product);
    }
}
```

Fig. 7. Test Program 4

Factors:

- In IF statements count the weight due to type of control structures (Wc) as 0.40

- In IF statements count the Weight due to nesting level of control structure (wn) as 0.60

- In IF statements count the Weight due to inheritance level of statements (Wi) as 1.58~1.5829

In the very first we got WCC value using our lecture slide Wc , Wn , Wi values.

Wc	Branch	1
Wn	Statements inside the outer most level/first level of control structures (Branch)	1
Wi	Statements inside the first derived class	1

After that, got three Factors:

- In IF statements count the weight due to type of control structures (Wc) as 0.40

- In IF statements count the Weight due to nesting level of control structure (wn) as 0.60

- In IF statements count the Weight due to inheritance level of statements (Wi) as 1.58

- By using those three weights able to get value **almost equal** to this WCC values instead of previous weights (Wc = 1, Wn = 1 , Wi = 1).

Calculation

Wt = Wc + Wn + Wi

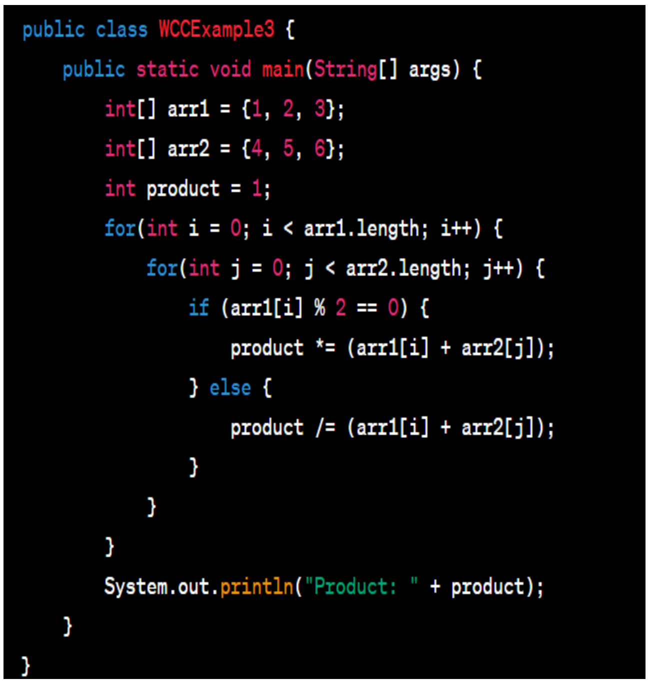
=(0+0+1.58)+(0+0+1.58)+………+(0+0.60+1.58)

WCC = S * Wt

=2*1.58 +5*1.58 +…………..+ 6*2

= 102.66

Some other example code of having same EWCC value



```
public class WCCExample3 {
    public static void main(String[] args) {
        int[] arr1 = {1, 2, 3};
        int[] arr2 = {4, 5, 6};
        int product = 1;
        for(int i = 0; i < arr1.length; i++) {
            for(int j = 0; j < arr2.length; j++) {
                if (arr1[i] % 2 == 0) {
                    product *= (arr1[i] + arr2[j]);
                } else {
                    product /= (arr1[i] + arr2[j]);
                }
            }
        }
        System.out.println("Product: " + product);
    }
}
```

Fig. 8. Sample Program 4

To compute the product of the items in two arrays, the two code portions each employ nested loops. Each element of the first array is simply multiplied by each element of the second array in the first code segment, WCCExample2, and the product is then accumulated in a variable. Depending on whether the current element of the first array is even or odd, the second code segment, WCCExample3, inserts a conditional statement inside the nested loops to multiply or divide the product. Because the first code segment has a less complex structure and fewer

conditional statements than the second code segment, its WCC value is lower. The project's needs, such as performance, readability, and maintainability, will determine which type of code should be used.

TABLE I. CALCULATION OF WCC METRIC FOR TEST PROGRAM 1

Line No	Tokens	S	Wc	Wn	Wi	Wt	WC
1	public class Main {						
2	Void , main( )	2	0	0	1	1	2
3	Scanner ,input, = ,new ,Scanner()	5	0	0	1	1	5
4	int ,num , = , input , . , nextInt()	6	0	0	1	1	6
5	If-else() ,num , > , 0	4	1	1	1	3	12
6	If() , num ,% ,2 ,==, 0	6	1	1	1	3	18
7	System, . ,out , . , println() ,"The number is positive and even."	6	0	1	1	2	12
8	}						
9	else {						
10	System, . , out , . , println() ,"The number is positive and odd."	6	0	1	1	2	12
11	}						
12	}						
13	Else-if() , num , < , 0	4	1	1	1	3	12
14	System, . , out , . ,println() ,"The number is negative."	6	0	1	1	2	12
15	}						
16	else {						
17	System , . , out , . , println() ,"The number is zero."	6	0	1	1	2	12
18	}						
19	}						
	WCC						**103.00**

TABLE II. CALCULATION OF EWCC METRIC FOR TEST PROGRAM 1

Line No	Tokens	S	Wc_e	Wn_e	Wi_e	Wt_e	WC
1	public class Main {						
2	Void , main( )	2	0	0	1.509	1.509	3.018
3	Scanner ,input, = ,new ,Scanner()	5	0	0	1.509	1.509	7.545
4	int ,num , = , input , . , nextInt()	6	0	0	1.509	1.509	9.054
5	If-else() ,num , > , 0	4	0.50	0.50	1.509	2.509	10.036
6	If() , num ,% ,2 ,==, 0	6	0.50	0.50	1.509	2.509	15.054
7	System, . ,out , . , println() ,"The number is positive and even."	6	0	0.50	1.509	2.009	12.054
8	}						
9	else {						
10	System, . , out , . , println() ,"The number is positive and odd."	6	0	0.50	1.509	2.009	12.054
11	}						
12	}						
13	Else-if() , num , < , 0	4	0.50	0.50	1.509	2.509	10.036
14	System, . , out , . ,println() ,"The number is negative."	6	0	0.50	1.509	2.009	12.054
15	}						
16	else {						
17	System , . , out , . , println() ,"The number is zero."	6	0	0.50	1.509	2.009	12.054
18	}						
19	}						
	EWCC						**103.00**

.

TABLE III. CALCULATION OF WCC METRIC FOR TEST PROGRAM 2

Line No	Tokens	S	Wc	Wn	Wi	Wt	WC
1	public class GradeChecker {						
2	Void , main( )	2	0	0	1	1	2
3	Scanner, scanner, =, new, Scanner()	5	0	0	1	1	5
4	System,., out ,., print, "Enter your grade: "	6	0	0	1	1	6
5	Int, grade, = , scanner, ., nextInt()	6	0	0	1	1	6
6	If-else () , grade ,>, =, 90	5	1	1	1	3	15
7	System, . ,out , . , println() , "You got an A"	6	0	1	1	2	12
8	}, else if (), grade,>, =, 80, {	5	1	1	1	3	15
9	System, . ,out , . , println() , "You got an B"	6	0	1	1	2	12
10	}, else if (), grade,>, =, 70, {	5	1	1	1	3	15
11	System, . ,out , . , println() , "You got an C"	6	0	1	1	2	12
12	}, else if (), grade,>, =, 60, {	5	1	1	1	3	15
13	System, . ,out , . , println() , "You got an D"	6	0	1	1	2	12
14	}, else, {						
15	System, . ,out , . , println() , "You got an F"	6	0	1	1	2	12
16	}						
17	Scanner, close()	2	0	0	1	1	2
18	}						
19	}						
	WCC						**141**

TABLE IV. CALCULATION OF EWCC METRIC FOR TEST PROGRAM 2

Line No	Tokens	S	Wc_e	Wn_e	Wi_e	Wt_e	WC
1	public class GradeChecker {						
2	Void , main( )	2	0	0	1.20	1.20	2.4
3	Scanner, scanner, =, new, Scanner()	5	0	0	1.20	1.20	6
4	System,., out ,., print, "Enter your grade: "	6	0	0	1.20	1.20	7.2
5	Int, grade, = , scanner, ., nextInt()	6	0	0	1.20	1.20	7.2
6	If-else () , grade ,>, =, 90	5	1.20	0.60	1.20	3	15
7	System, . ,out , . , println() , "You got an A"	6	0	0.60	1.20	1.80	10.8
8	}, else if (), grade,>, =, 80, {	5	1.20	0.60	1.20	3	15
9	System, . ,out , . , println() , "You got an B"	6	0	0.60	1.20	1.80	10.8
10	}, else if (), grade,>, =, 70, {	5	1.20	0.60	1.20	3	15
11	System, . ,out , . , println() , "You got an C"	6	0	0.60	1.20	1.80	10.8
12	}, else if (), grade,>, =, 60, {	5	1.20	0.60	1.20	3	15
13	System, . ,out , . , println() , "You got an D"	6	0	0.60	1.20	1.80	10.8
14	}, else, {						
15	System, . ,out , . , println() , "You got an F"	6	0	0.60	1.20	1.80	10.8
16	}						
17	Scanner, close()	2	0	0	1.20	1.20	2.4
18	}						
19	}						
	EWCC						**141**

TABLE V. CALCULATION OF WCC METRIC FOR PROGRAM 3

Line No	Tokens	S	Wc	Wn	Wi	Wt	WC
1	public class ForLoopExample {						
2	void ,main(){	2	0	0	1	1	2
3	int[][] matrix = new int[4][4];	5	0	0	1	1	5
4	int value = 1;	2	0	0	1	1	2
5	for (int i = 0; i < matrix.length; i++) {	5	1	1	1	3	15
6	for (int j = 0; j < matrix[i].length; j++) {	5	1	1	1	3	15
7	matrix[i][j] = value++;	4	0	1	1	2	8
8	}						
9	}						
10	System.out.println("Matrix values:");	3	0	0	1	1	3
11	for (int i = 0; i < matrix.length; i++) {	5	1	1	1	3	15
12	for (int j = 0; j < matrix[i].length; j++) {	5	1	1	1	3	15
13	System.out.print(matrix[i][j] + "\t");	3	0	1	1	2	6
14	}						
15	System.out.println();	3	0	0	1	1	3
16	}						
17	}						
18	}						
	WCC						**89.00**

TABLE VI. CALCULATION OF EWCC METRIC FOR PROGRAM 3

Line No	Tokens	S	Wc_e	Wn_e	Wi_e	Wt_e	WC
1	public class ForLoopExample {						
2	Void , main( )	2	0	0	1.590	1.590	3.180
3	int[][] matrix = new int[4][4];	5	0	0	1.590	1.590	7.950
4	int value = 1;	2	0	0	1.590	1.590	3.180
5	for (int i = 0; i < matrix.length; i++) {	5	0.30	0.60	1.590	2.490	12.450
6	for (int j = 0; j < matrix[i].length; j++) {	5	0.30	0.60	1.590	2.490	12.450
7	matrix[i][j] = value++;	4	0	0.60	1.590	2.190	8.760
8	}						
9	}						
10	System.out.println("Matrix values:");	3	0	0	1.590	1.590	4.770
11	for (int i = 0; i < matrix.length; i++) {	5	0.30	0.60	1.590	2.490	12.450
12	for (int j = 0; j < matrix[i].length; j++) {	5	0.30	0.60	1.590	2.490	12.450
13	System.out.print(matrix[i][j] + "\t");	3	0	0.60	1.590	2.190	6.570
14	}						
15	System.out.println();	3	0	0	1.590	1.590	4.770
16	}						
17	}						
18	}						
19							
	EWCC						**89.00**

TABLE VII. CALCULATION OF WCC METRIC FOR PROGRAM 4

Line No	Tokens	S	Wc	Wn	Wi	Wt	WC
1	public class WCCExample2 {						
2	public static void main(String[] args) {	2	0	0	1	1	2
3	int[] arr1 = {1, 2, 3};	5	0	0	1	1	5
4	int[] arr2 = {4, 5, 6};	5	0	0	1	1	5
5	int product = 1;	4	0	0	1	1	4
6	for(int i = 0; i < arr1.length; i++) {	9	1	1	1	3	27
7	for(int j = 0; j < arr2.length; j++) {	9	1	1	1	3	27
8	product *= (arr1[i] + arr2[j]);	7	1	1	1	3	21
9	}						
10	}						
11	System.out.println("Product: " + product);	6	0	1	1	2	12
12	}						
13	}						
	WCC						**103**

TABLE VIII. CALCULATION OF EWCC METRIC FOR PROGRAM 4

Line No	Tokens	S	Wc_e	Wn_e	Wi_e	Wt_e	WC
1	public class WCCExample2 {						
2	public static void main(String[] args) {	2	0	0	1.58	1.58	3.16
3	int[] arr1 = {1, 2, 3};	5	0	0	1.58	1.58	7.80
4	int[] arr2 = {4, 5, 6};	5	0	0	1.58	1.58	7.80
5	int product = 1;	4	0	0	1.58	1.58	6.32
6	for(int i = 0; i < arr1.length; i++) {	9	0.40	0.60	1.58	2.58	23.22
7	for(int j = 0; j < arr2.length; j++) {	9	0.40	0.60	1.58	2.58	23.22
8	product *= (arr1[i] + arr2[j]);	7	0.40	0.60	1.58	2.58	18.06
9	}						
10	}						
11	System.out.println("Product: " + product);	6	0	0.60	1.58	2.18	13.08
12	}						
13	}						
	EWCC						**103**

CONCLUSION

We developed the Enhanced Weighted Class Complexity (EWCC) metric in this study, which expands on the well-known Weighted Class Complexity (WCC) metric by adding additional complexity elements and allocating weights depending on their relative importance. Data access patterns, method call chains, and control structure complexity are among the other variables. In order to compare the EWCC metric's output to the WCC metric's output, we applied it to a number of software systems.

The findings demonstrated that the EWCC metric offered a more thorough and precise evaluation of class-level complexity than the WCC metric. The EWCC metric was particularly useful for locating potential sources of defects and errors that the WCC metric did not pick up on. Overall, the results point to the potential value of the EWCC metric as a tool for software developers and maintainers to manage program complexity.

The EWCC metric offers a more thorough and precise evaluation of class-level complexity, which can assist developers in spotting possible problems and allocating maintenance resources more effectively. In addition, the weights allocated to various complexity characteristics can be changed, allowing the EWCC measure to be adjusted to suit particular software development environments. We believe that the EWCC metric has the potential to contribute to the current efforts to enhance software quality and maintainability, and we hope that our research will motivate future investigation and improvement of the EWCC statistic.

REFERENCES

[1] S. B. Usha Chhillar, "A New Weighted Composite Complexity Measure for Object-Oriented Systems," vol. 1, uly 2011 .

[2] Kulwinder Singh, P. K. Bhatia, "An Approach to Measure Cognitive Complexity for Object Oriented Code", vol.3 Issue I, January 2015.

[3] D. I. De Silva, N. Kodagoda, S. R. Kodituwakku, A. J. Pinidiyaarachchi, "Analysis and Enhancements of a Cognitive Based Complexity Measure".

[4] D. I. De Silva1, S. R. Kodituwakku, A. J. Pinidiyaarachchi, N. Kodagoda, "Enhancements to an OO Metric: CB Measure", October 20, 2017.

[5] D. I. De Silva, N. Kodagoda, S. R. Kodituwakku, A. J. Pinidiyaarachchi, "Limitations of an Object-Oriented Metric : Weighted Complexity Measure".

[6] SANJAY MISRA AND K. IBRAHIM AKMAN., " Weighted Class Complexity: A Measure of Complexity for Object Oriented System ".