An Interface based Cognitive Weighted Class Complexity Measure for Object Oriented Design

K. Maheswaran
Research Scholar
Department of Computer Science
St. Joseph’s College (Autonomous)
Tiruchirappalli, Tamil Nadu, India.
mahes161@gmail.com

A. Aloysius
Assistant Professor
Department of Computer Science
St. Joseph’s College (Autonomous)
Tiruchirappalli, Tamil Nadu, India.
aloysius1972@gmail.com

Abstract- Measurement can be used from the beginning to end of a software project development to support in estimation, quality control, complexity assessment and decision making processes. The human effort is an important aspect of software development and designing. Cognitive informatics is vital in understanding the important aspects of the software. Class complexity is measured by different researchers based on Line of Codes, Functional Points, Number of Methods, Number of Attributes, Inheritance etc. In software engineering, interface has been used by the developers for more than two decades. But, there are no proper metrics to assess, if an interface is implemented in the class, then complexity of a class is affected. This paper aims to propose a metric by using cognitive weights to measure the complexity of a class by adding the cognitive complexity of a user defined interfaces from the existing metrics.

Keywords: Cognitive weight, Interface Complexity

I. INTRODUCTION

Software complexity refers to “the effort required to understand, modify and maintain a software”[1]. It deals with psychological difficulty while comprehending the software or program. The cognitive complexity means the mental burden on the user who deals with the code. It is an important parameter to measure the complexity of a class in an object oriented (OO) paradigm. Software metrics always help software developers to assure quality. In general, quality objectives may have the following factors like performance, reliability, availability and maintainability, which are strictly related to software complexity[2]. According to IEEE, complexity is "the degree to which a system or a component has a design or implementation that is difficult to understand and verify” [3]. The researchers have given solutions to their problems in the field like software engineering, cognitive sciences and artificial intelligence using cognitive informatics [4–6]. These techniques can also be adopted by IT industries into their software development practices. The reliability and maintainability of software can easily be predicted by using these metrics [7]. The basic elements of an object oriented software are classes, subclasses, objects, attributes, methods and messages. These elements are identified in class declarations, which are the responsible elements of the class complexity. These parameters alone, don’t exactly predict the complexity of a class. The other factors like inheritance, interfaces are also affecting the complexity of a class. Very few researchers have discussed the effect of interface complexity of a class.

Cognitive weights are defined as “the extent of difficulty or relative time and effort required for comprehending given software and measure the complexity of the logical structure of the software” [1]. If the complexity value of the program or software is increased then the effort required for understanding the program or software is also increasing. The cognitive complexity is high due to many reasons, namely the high fault-proneness, low maintenance, etc. The high cognitive complexity value reveals that the design is very poor, and is uncontrollable [8]. In such a situation, the effort to main the software is increasing significantly. The cognitive complexity of software can be identified using many cognitive processes out of which the program comprehension is one of the major factor. [9].
II. REVIEW OF LITERATURE

Software metrics help to measure the complexity of software by using the different aspects or parameters. Earlier, the OO metrics does not consider the cognitive characteristics while evaluating the complexity of the class, which directly affects software development and maintenance. Researchers proposed many metrics for object oriented systems. Chidamber and Kemerer (C&K) proposed metric suites of OO which consists of six different metrics namely Weighted Method per Class (WMC), Depth of Inheritance Tree (DIT), Response for Class (RFC), Number Of Children (NOC), Lack of Cohesion of Methods (LOC) and Coupling Between Objects (CBO) [11]. Rajnish et al. [10, 13] has suggested a metric which is used to measure the design stage complexity of the class. Balasubramanian [12] presented another complexity metric for a class which is evaluated by adding the number of instance variable and static complexity of local methods in class. Mishra [15] suggested the Class Complexity (CC) metric which calculates by assigning the cognitive weights to each member function, and then by adding the weights of all methods. Sandip Mal [20] suggested an OO complexity metric measure known as weighted class complexity (WCC) [14], which is calculated by the method complexity and the Number of Attributes in the class.

Mishra [17] developed a metric known as Class complexity due to inheritance (CCI) which is defined in the following equation (1).

\[
CCI_i = \sum_{t_{form}=1}^{k} C_{t_{form}} + \sum_{i=1}^{l} M_{C_i}
\]  

(1)

Rajnish [19] proposed another OO measure to evaluate the design phase complexity of software with the help of attributes and methods. He assigned weights for attributes and methods in the class. Sandip Mal [20] suggested a new complexity metric by computing the number of methods, instance, external attributes, the number of super, subclass and Cyclomatic Complexity of a Class.

Vinay et al., proposed a complex metric named CCC (Complete Class Complexity). It focuses each dimension of a class with nine different parameters namely number of methods (NOM), average Cyclomatic complexity(AVCC), etc., to identify the complexity of a class [21] as shown in the equation (2).

\[
CCC = NOMT + AVCC + MOA + EXT + NSUP + NSUB + INTR + PACK + NQU
\]  

(2)

Where MOA - Measure of aggregation, EXT - External Method calls, NSUP - Number of Super Class, NSUB - Number of Sub Class, INTR - Interface Implemented, PACK - Package Imported, NQU - Number of Queries.

Rajnish suggested another Class Complexity Metric (CCM) of an OO program by adding the following parameters of a class, namely Total Cyclomatic Complexity (TCC), Number of Methods (NOMT), Number of Instance Variables (INST) declared, Number of External Methods (EXT) called, Number of Local Methods (LMC) called, and Total Lines of Code (NLOC) [22] which is represented in equation (3).

\[
CCM = k + w1 \times TCC + w2 \times NOMT + w3 \times INST + w4 \times EXT + w5 \times LMC + w6 \times NLOC
\]  

(3)

Where, \( w_i \) stands for the weights and the constant ‘k’ are derived at least square regression analysis. CCM helps to forecast the understandability of classes, but it does not concentrate performance parameter such as maintenance effort and system performance.

Arockiam et al., [16] proposed class complexity metric named as Extended Weighted Class Complexity (EWCC) which is a continuation of Weighted Class Complexity (WCC). EWCC computes the inherited complexity of a class (ICC). In EWCC, the complexity of the attributes doesn’t take accurately. It computes only the number of attributes (variables) used in a class. Aloysius et al., extended EWCC and named it as Attribute Weighted Class Complexity (AWCC) [18]. In AWCC, the attributes were classified by three categories namely primary, derived and user defined with weights 1, 2 and 3 respectively. AWCC is defined by the following equation (4).

\[
AWCC = \sum_{i=1}^{n} A_{C_i} + \sum_{j=1}^{m} M_{C_j} + \sum_{k=1}^{l} ICC_k
\]  

(4)

Where AC - Attribute Complexity, MC - Method Complexity is computed by allocating the CWS proposed by Wang in [1], ICC - Inherited Class Complexity,

\[
ICC = (DIT \times CL) + \sum_{k=1}^{x} RMG_k + RN_k
\]  

(5)

In equation (5), where ‘x’ is the number of inherited methods, RN_k - Total number of Reused attributes, RMC - Reused Method Complexity, DIT - Depth of Inheritance Tree and CL - Cognitive Complexity of L_k level.

The main limitation of AWCC is the cognitive load (CL) to understand the different levels of inheritance hierarchy is always assumed to be one. In Real time, the cognitive load must vary for the different levels of inheritance hierarchies. Hence, it needs to be improved.

Maheswaran et al., proposed a metric named as cognitive weighted inherited class complexity (CWICC) by overcoming the limitations of AWCC [23].

\[
CWICC = \sum_{i=1}^{n} A_{C_i} + \sum_{j=1}^{m} M_{C_j} + \sum_{k=1}^{l} ICC_k
\]  

(6)

In equation (6), AC - Attribute Complexity is defined in equation (10), MC - Method Complexity, ICC - Improved Inherited Class Complexity which is calculated in following equation (7).

\[
IIICC = CDC + \sum_{k=1}^{x} RMG_k + \sum_{i=1}^{t} RAC_i
\]  

(7)

Where CDC - Cognitive Depth Complexity is calculated in equation (8). RAC stands for Reused Attribute Complexity.

\[
CDC = DC \times (CIW_k)
\]  

(8)
Where DC - Depth of Class from the root. CW\_i - Cognitive Weight of the particular level in inheritance hierarchy and it varies between levels. The main drawback of the doe metric is not considering the user-defined interface complexity while measuring the class complexity. Hence, this paper overcomes the drawback of CWICC and suggests an improved metric.

III. INTERFACE BASED COGNITIVE WEIGHTED CLASS COMPLEXITY (ICWCC)

Consider a class with 'm' attributes, 'n' methods and 'o' number of classes are derived and 'p' interfaces are implemented in the class. ICWCC of the class can be calculated using the following equation (9).

\[
ICWCC = \sum_{i=1}^{m} AC_i + \sum_{j=1}^{n} MC_j + \sum_{k=1}^{a} IICC_k + \sum_{x=1}^{p} ICC_x
\] (9)

Where AC - Attribute Complexity is defined in equation (10), MC - Method Complexity, IICC - Improved Inherited Class Complexity, IIC - Interface Implemented Complexity.

\[
AC = (PDT \times W_b) + (DDT \times W_d) + (UDDT \times W_u)
\] (10)

Where PDT - Primary Data Type, DDT - Derived Data Type attributes, UDDT - User Defined Data Type attributes, W\_b Cognitive Weights of the PDT, W\_d - Cognitive Weights of the DDT, W\_u - Cognitive Weights of the UDDT attributes. IICC is computed as defined in the equation (7). Interface Implemented Complexity (IIC) is defined by the following equation (11).

\[
IIC = (NSII \times CW) + NMSI + (NEII \times CW) + by \ NMEI + (NNII \times CW) + NMNI
\] (11)

Where NSII - Number of Simple Interface Implemented, NEII - Number of Extended Interface Implemented, NNII - Number of Nested Interface Implemented, NMSI - Number of Methods in Simple Interface, NNEI - Number of Methods in Extended Interface, NMNI - Number of Methods in Nested Interface. CW\_a, CW\_b, CW\_c are the weights of simple, extended, nested interfaces. These weighting factors of interfaces are built on the taxonomy of cognitive phenomena as proposed by Wang [1].

Calibration of cognitive weight for interfaces

The experimentation was organized to identify cognitive weight for the several types of interface, namely simple, extended, and nested interface. A test of comprehension was conducted with six programs to discover the time taken to realise the Java class programs with different types of user-defined interface. There are 85 postgraduate students who have sufficient exposure in the OO programs were selected to take part in the test. In Table 1, The six different programs and its Average Comprehension Time (ACT) and the assigned weights based on the ratio of ACT are displayed. The value of ACT is multiples of 120. Hence the corresponding cognitive weights are 3, 4 and 5.

<table>
<thead>
<tr>
<th>Program</th>
<th>ACT</th>
<th>Category</th>
<th>Category wise ACT (in Seconds)</th>
<th>ACT Rounded</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>345</td>
<td>SI</td>
<td>356</td>
<td>360</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>366</td>
<td>EI</td>
<td>483</td>
<td>480</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>490</td>
<td>NI</td>
<td>605</td>
<td>600</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>475</td>
<td>EI</td>
<td>483</td>
<td>480</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>595</td>
<td>NI</td>
<td>605</td>
<td>600</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>615</td>
<td>SI</td>
<td>356</td>
<td>360</td>
<td>3</td>
</tr>
</tbody>
</table>

IV. CASE STUDY

The proposed metric ICWCC is compared with other widely known metrics WCC and CWICC. The following program pro\_inter demonstrates how the weights are calculated or assigned for different attributes, methods, interfaces, statement, etc.,

```
interface interist {
    public void read();
    public void read2();
    public void read3();
}

interface intersec extends interist {
    public void calcadd();
    public void calcsub();
}

interface interthird {
    public void disptrans();
}

class lst_class implements intersec { //IIC = (1 * 4) + 4 = 8
    public int a[] = new int[3][3];
    public int b[] = new int[3][3];
    public int arr[] = new int[3][3];
    public int c[] = new int[3][3];
    public int d[] = new int[3][3];
    public int i;
    public int sum;
}

Scanner scan= new Scanner(System.in);

public void read() {
    System.out.print(“Enter Elements : “);
    for(i=0; i<3; i++)
    for(j=0; j<3; j++)
        System.out.print(“Enter Elements : “);
}
```

2773
a[i][j] = scan.nextInt();
}
public void read2() { // (10)
System.out.print("Enter elements :");
for (i = 0; i < 3; i++)
{
for (j = 0; j < 3; j++)
{
b[i][j] = scan.nextInt();
}
}
public void calcadd() { // (10)
System.out.print("calculation is below\n");
for (i = 0; i < 3; i++)
{
for (j = 0; j < 3; j++)
{
c[i][j] = a[i][j] + b[i][j];
}
}
public void calcsub() { // (10)
System.out.print("Subtraction of two metrics is below\n");
for (i = 0; i < 3; i++)
{
for (j = 0; j < 3; j++)
{
d[i][j] = a[i][j] - b[i][j];
}
}
public void sum1() { // (10)
for (i = 0; i < a.length; i++)
{
for (int j = 0; j < a[i].length; j++)
{
sum = sum + a[i][j];
}
System.out.println("The sum of row = " + sum);
}
public void trans() { // (10)
System.out.print("calculation performed...\n");
for (i = 0; i < 3; i++)
{
for (j = 0; j < 3; j++)
{
arrt[i][j] = a[j][i];
}
}
}
public static void main(String args[])
{
Sec_class obj = new Sec_class();
obj.read();
obj.read2();
obj.sum1();
obj.trans();
obj.calcadd();
obj.calcsub();
obj.displ1();
obj.displ2();
obj.disptran();
}
class Sec_class extends istclass implements interthird
// IICC = (1*1) + 6 = 7     // IIC = (1*3) + 1 = 4
{ int i; j;
public void displ1() { // (2*1)
System.out.print("The Result is :\n");
for (i = 0; i < 3; i++)
{
for (j = 0; j < 3; j++)
{
System.out.print(c[i][j] + " "); // RAC 2
}
}
public void displ2() { // (10)
System.out.print("The Result is :\n");
for (i = 0; i < 3; i++)
{
for (j = 0; j < 3; j++)
{
System.out.print(d[i][j] + " "); RAC \2
}
}
public void disptran() { // (10)
System.out.print("Result is :\n");
for (i = 0; i < 3; i++)
{
for (j = 0; j < 3; j++)
{
System.out.print(arrt[i][j] + " "); RAC \2
}
System.out.println();
}
}
class extendmodify
{ public static void main(String args[])
{ Sec_class obj = new Sec_class();
obj.read();
obj.read2();
obj.sum1();
obj.trans();
obj.calcadd();
obj.calcsub();
obj.displ1();
obj.displ2();
obj.disptran();
}
}

The values represented in comments sections is weights of the corresponding methods or attributes or statements.
V. COMPARATIVE STUDY AND CORRELATION ANALYSIS

The proposed metric ICWCC is an extension of CWICC. It also calculates the complexity due to the different types of interface when it is implemented in the class. Another advantage of the proposed metric is by considering and assigning cognitive weights for different categories of user defined interfaces to predict the complexity of the user defined classes more accurately. The three programs were taken for comparison and the metric values are calculated (Refer Table 2) and displayed in Table 3. The diagrammatic representation of the metric values is displayed via chart in Fig 1.

![Fig 1. Comparison chart of proposed metric with WCC and CWICC](image)

Table 2. Calculation of metric values for the program P3_E

<table>
<thead>
<tr>
<th>Classes</th>
<th>ICWCC</th>
<th>CWICC</th>
<th>WCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ist_class</td>
<td>AC = (3<em>1) + (5</em>2) = 13</td>
<td>AC = (3<em>1) + (5</em>2) = 13</td>
<td>AC = 8</td>
</tr>
<tr>
<td></td>
<td>MC = 10 + 10 + 10 + 10 + 10 = 60</td>
<td>MC = 10 + 10 + 10 + 10 = 60</td>
<td>MC = 10 + 10 = 60</td>
</tr>
<tr>
<td></td>
<td>IICC = 0</td>
<td>IICC = 0</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>ICWCC = 13 + 60 + 0 + 8 = 81</td>
<td>CWICC = 13 + 60 = 73</td>
<td>WCC = 8 + 60 = 68</td>
</tr>
<tr>
<td></td>
<td>AC = 2</td>
<td>AC = 2</td>
<td>AC = 2</td>
</tr>
<tr>
<td></td>
<td>MC = 10 + 10 + 10 = 30</td>
<td>MC = 10 + 10 + 10 = 30</td>
<td>MC = 10 + 10 = 30</td>
</tr>
<tr>
<td></td>
<td>IICC = (1*1) + 6 = 7</td>
<td>IICC = (1*1) + 6 = 7</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>ICWCC = 2 + 30 + 7 + 4 = 43</td>
<td>CWICC = 2 + 30 + 7 = 39</td>
<td>WCC = 2 + 30 = 32</td>
</tr>
<tr>
<td>Sec_class</td>
<td>Total complexity of a program</td>
<td>81 + 43 = 124</td>
<td>73 + 39 = 112</td>
</tr>
</tbody>
</table>

The WCC and CWICC values were compared with proposed metric and found that ICWCC measure is higher than CWICC. Hence the proposed metric predicts the complexity more accurately.

Table 3. Comparison of ICWCC metric with WCC and CWICC

<table>
<thead>
<tr>
<th>PROG_NAME</th>
<th>WCC</th>
<th>CWICC</th>
<th>ICWCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1_N</td>
<td>25</td>
<td>36</td>
<td>43</td>
</tr>
<tr>
<td>P2_D</td>
<td>57</td>
<td>67</td>
<td>72</td>
</tr>
<tr>
<td>P3_E</td>
<td>100</td>
<td>112</td>
<td>124</td>
</tr>
</tbody>
</table>

To show that the proposed metric (ICWCC) is a better indicator for predicting class complexity, one step ahead another comprehension test was conducted with sixty PG students. Each student was given seven different Java programs which include the programs containing different types of user-defined interface. The main objective of the test is to record the time taken to complete the test (in seconds). The average time taken by all the students is displayed in Table 4, along with their calculated metric values.

Table 4. Comparison of AWCC and CWICC proposed ICWCC Metric

<table>
<thead>
<tr>
<th>Program #</th>
<th>WCC</th>
<th>CWICC</th>
<th>ICWCC</th>
<th>ACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>34</td>
<td>41</td>
<td>46</td>
<td>250</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>21</td>
<td>26</td>
<td>200</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>13</td>
<td>20</td>
<td>270</td>
</tr>
<tr>
<td>4</td>
<td>52</td>
<td>64</td>
<td>70</td>
<td>600</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>31</td>
<td>40</td>
<td>450</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>7</td>
<td>15</td>
<td>200</td>
</tr>
<tr>
<td>7</td>
<td>31</td>
<td>43</td>
<td>47</td>
<td>300</td>
</tr>
</tbody>
</table>
From Table 4. It is observed that ICWCC metric value is more than the WCC and CWICC and is proximate to Average Comprehension Time (ACT). The Correlation analysis is done between WCC vs ACT, CWICC vs ACT and ICWCC vs ACT which yield r = 0.734862, 0.756884 and 0.803694 respectively. ICWCC is more positively correlated to ACT than WCC and CWICC. This ascertains that ICWCC is an enhanced indicator of complexity of the classes when the user-defined interface complexity was included.

V. THEORETICAL VALIDATION

Weyuker proposed nine different properties to determine the efficiency and robustness for various software complexity measures formally [24]. These properties are used to evaluate the ICWCC metric.

Property 1: Non-coarseness
\( (\exists P \ (\exists Q \ (|P| \neq |Q|) \text{ where } P \text{ and } Q \text{ are two different programs (classes))).) \)

All of the classes is not same ICWCC. From Table 2, the classes 1st_class and Sec_class are different values 81 and 43, respectively. Hence Property 1 is satisfied.

Property 2: Granularity
(\( \text{Let } c \text{ be a non-negative number. There are only finitely many programs of complexity } c. \))

There are only a finite number of classes in a program having same BCSs and type of interfaces. So only finite numbers of classes have same complexity. Hence, the ICWCC metric does hold property 2.

Property 3: Non-uniqueness
(\( \text{There are distinct classes } P \text{ and } Q \text{ such that } |P| = |Q|. \))

There are many classes of the same complexity. From a given number of classes, there is a chance that, more than one class having the same number of methods, attributes and interface. So their complexities may be the same. Hence Property 3 is satisfied.

Property 4: Design Implication
(\( (\exists P \ (\exists Q \ (P \equiv Q \text{ and } |P| \neq |Q|)).) \))

If P and Q are two classes having the same functionality with the same output, then their internal architecture may vary. The complexities of these two classes are not equal. The proposed measure satisfied the property 4.

Property 5: Monotonicity
(\( (\exists P \ (\exists Q \ (|P| \leq |P; Q|) \text{ and } |P; Q| \leq |P|)). \))

Consider two classes P and Q, the complexity of the class P and Q is less than the combined complexities of P,Q. For example, in the prog_inter the complexity of 1st_class and sec_class are 81 and 43 respectively. They are all less than the combined (entire) complexity i.e 124. Hence the property.

Property 6: Non-equivalence of Interaction
(\( (\exists P \ (\exists Q) \ (|P| = |Q|) \text{ and } |P; R| = |Q; R|)). \))

If two classes P and Q have the similar complexity than a third-class R is appended to both of them, the complexities of the two new combined class P and R can be dissimilar than the Q and R since the internal interaction and architecture may change when the classes are appended, hence the complexities are not equal. This means the proposed measure doesn’t satisfy property 6.

Property 7: Significance of Permutation

There are program bodies P and Q, such that the statements inside P are permuted to form Q. The main aim is to ensure that result of the permutation of classes leads to change in metric values. This property does not hold for any OO metric, since the alteration the order in which methods or attributes are initialized does disturb the order in which they are executed. Thus, ICWCC metric does not fulfill this property.

Property 8: No change on renaming
(\( (\text{If } P \text{ is a renaming of } Q, \text{ then } |P| = |Q|). \))

The name of a class is modified, it will not disturb the complexity of the class. The proposed complexity measure ICWCC, there is no consequence in the complexity values while renaming the. Hence the property.

Property 9: Interaction increases Complexity

This property shows complexity is increased due to interaction of classes. The complexity of two interacting classes will be at least equal or less than the sum of the measures of individual classes. Hence this property 9 is fulfilled.

The suggested ICWCC metric has been validated by using Weyuker’s properties and found that the proposed measures satisfy seven Weyuker’s properties out of nine as shown in Table 5. As a result, this measure is established as a well-structured one.

(Note: Yes denotes property is satisfied or fulfilled and No denotes property is not satisfied or fulfilled)

<table>
<thead>
<tr>
<th>Proposed Metric</th>
<th>Weyuker’s properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICWCC</td>
<td>Yes</td>
</tr>
<tr>
<td>CWICC</td>
<td>Yes</td>
</tr>
<tr>
<td>CWICC</td>
<td>Yes</td>
</tr>
<tr>
<td>ICWCC</td>
<td>Yes</td>
</tr>
<tr>
<td>ICWCC</td>
<td>Yes</td>
</tr>
<tr>
<td>ICWCC</td>
<td>No</td>
</tr>
<tr>
<td>ICWCC</td>
<td>No</td>
</tr>
<tr>
<td>ICWCC</td>
<td>Yes</td>
</tr>
<tr>
<td>ICWCC</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 5. Weyuker properties against proposed ICWCC Metric

VI. CONCLUSION AND FUTURE WORK

The complexity of a class is significantly affected by the different category of interface when it is implemented. Hence an Interface based Cognitive Weighted Class Complexity metric (ICWCC) for object oriented design to measure class complexity has been formulated. The metric value is calculated through a case study program for the proposed and the existing metrics. The comparative study and correlation analysis has been performed, which shows that the proposed metric is an improved indicator of predicting a complexity of a class. In future, the proposed metric ICWCC needs to validate more empirically with real time projects and also extended to include other OO features which are not concentrated in this paper like polymorphism to predict class complexity more precisely.
REFERENCES
