Cognitive Information Complexity Measure of Object-Oriented Program

DHARMENDER SINGH KUSHWAHA, R.K.SINGH and A.K.MISRA
Department of Computer Science and Engineering
Moti Lal Nehru National Institute Of Technology
Allahabad,
INDIA.

Abstract: - Object-oriented paradigm is built on the notion of real world entities. This paradigm has received higher acceptability among the programming community. This is because the programs written using an object-oriented language are easier to comprehend as compared to the procedural programs. Hence this paper makes an attempt to apply Cognitive Information Complexity Measure (CICM) on object-oriented program and proves that cognitive complexity for any procedural program is greater than the cognitive complexity of the same program written in object-oriented software.

Keywords: - Cognitive informatics, Cognitive information complexity measure, Inter-Object Complexity Measure.

1 Introduction
Object-oriented paradigm is built on the notion of real world entities. This paradigm has received higher acceptability among the programming community. This is because the programs written using an object-oriented language are easier to comprehend as compared to the procedural programs. Hence complexity for any procedural program should generally be greater than the complexity of the same program written in object-oriented software.

A fundamental finding in computer science is that software, as an artifact of human creativity is not constrained by the laws and properties discovered in the physical world. Information is the abstraction of real-world objects and their relations, which can be processed by brain [10]. It is now realized that a large part of things in software description are non-algorithmic and non-functional, such as system architecture, performance, quality and fault-tolerance mechanism. Based on the above findings, our work establishes that the same cognitive information complexity measure can be used to compute the complexity of object-oriented programs. Also it is able to demonstrate that for a given requirement, the complexity of program written in object-oriented language is lower than the same program written in procedural language.

Cognitive informatics is still exploring the solutions to a vital question “How does the human natural intelligence process information?” The natural intelligence is derived among many things by self-learning. The self-learning is based on the interaction with the real world. In real world, things around us are objects and the essence of the object-oriented design is based on decomposing the problem / system into objects. Hence there is a close linkage between the cognitive informatics and the object-oriented paradigm and we have made an attempt to establish the linkage. The next section makes an attempt to establish CICM on object-oriented paradigm along with the concept of entropy.

2 Cognitive Information Complexity Measure of Object-Oriented Software
As programs grow larger and more complex, even structured programming paradigm of procedural language begins to show sign of strain. No matter how well the structured programming approach is implemented, large programs become excessively complex.

The fundamental idea behind object-oriented language is to combine into a single unit both data and the functions that operate on the data. Such a unit is called an object. An objects functions, called member function in C++ typically provide the only way to
access its data. Data and its functions are encapsulated into a single entity. Member functions in C++ are called methods in some other object-oriented language such as Smalltalk. Also data items are referred to as instance variable. Calling an object member function is referred to as sending a message to the object. Therefore calculating the complexity of an object-oriented program can only considered being complete if it also encompasses the complexity associated with sending messages.

3 Procedure for Calculating the CICM of Object-Oriented Programs
In order to measure the cognitive information complexity of object-oriented software, the following procedure is adopted:

1. Calculate the cognitive information complexity of each class [2,5,6].
2. Calculate the cognitive information complexity of the main function.
3. Calculate the inter-object complexity.
4. Summing up the above three complexities gives us the cognitive information complexity measure of object-oriented software.

3.1 Cognitive Information Complexity Measure of Classes (CICMC)
Since the code inside a method is not distinguished as being procedural or object-oriented, we calculate the complexity of a class by calculating the cognitive complexity of each method in a class by cognitive information complexity measure (CICM) [2,3,4]. This measure is computationally simple and a robust one [3,8], since it adheres to all the nine Weyuker properties.

The CICM defines complexity as the product of weighted information count of the software (WICS) and sum of the cognitive weights of basic control structures (SBCS) of the software i.e. WICS * SBCS.

Since a class consists of number of methods, the complexity of a class is calculated by calculating the complexity of each method contained in a class. Let a class C contain $M_n$ number of methods where $n = 1,2,\ldots,k$ and let $CICM_1$, $CICM_2$,\ldots,$CICM_k$ be the CICM of each method. Then CICM of the class is defined as $\sum_{n=1}^{k} (CICM)_n$. Let a program P contain $C_i$ number of methods where $i = 1,2,\ldots,x$ and let $CICM_1$, $CICM_2$,\ldots,$CICM_x$ be the CICM of each class. Then CICMC of all classes is defined as $\sum_{i=1}^{x} (CICM)_i$.

The definitions of our metrics are based on the concept of object-orientations and hence are independent of the object-oriented programming language used. Thus CICMC will guide the developer in designing classes such that the class complexity is reduced and hence improving its reusability and maintainability.

3.2 Cognitive Information Complexity of the Main Function (CICMMF)
The code written in main function is also calculated using cognitive information complexity measure (CICM).

3.3 Inter-Object Complexity Measure (IOCM)
The inter-object complexity for a program measures the information flows between objects. It depends on the number of objects and the number of messages between objects. The information flow is measured from the collaboration diagram. A collaboration diagram shows the objects and relationships involved in an interaction, and the sequence of messages exchanged among the objects during the interaction. It describes both the static structure and dynamic behavior of a system.

The idea is that if a class has a large number of immediate children, there shall be increase in the message transfer among them. This will have a larger influence on system design and will make testing more complex [6]. It is also to be noted that if a class has a large number of children, it also indicates improper abstraction, but a greater amount of reuse in system. There are also instances when a class cannot respond to a message (i.e. it lacks a corresponding method of its own), then it will pass the message on to its parents and this fact should be captured by metrics concerned with the object-oriented development.

Wang [10] proposed one of the properties in informatics laws of software based on information entropy. The entropy theory has been applied to
measure the complexity of the software. Entropy has been used as a measure of uncertainty. According to Shannon’s definition, the higher the uncertainty associated with the signal, the greater is the amount of information conveyed by the signal. The entropy increases with the increase in the messages [1]. Value of entropy measure for the program consisting of two objects is higher than that of the complexity measure for the program consisting of one object.

Let a system contain $O_m$ number of objects where $m = 1, 2, \ldots, j$. Let any object $O_m$ send or receive $M_x$ number of messages where $x = 1, 2, \ldots, k$. Let ‘$n$’ be the number of total messages exchanged within the inheritance tree. The reference probability of object $O_m$ is $x/n$. The entropy $H$ for the system shall be $-\left[\frac{x}{n}\log\left(\frac{x}{n}\right)\right]$. Therefore inter-object complexity measure (IOCM) for all the objects/classes in the inheritance tree is:

$$\text{IOCM} = \sum_{m=1}^{j} \left\{-\frac{x}{n}\log\left(\frac{x}{n}\right)\right\}$$

Where $m = \text{number of objects } O_1, O_2, \ldots, O_m$

$x = \text{number of messages related to object } O_m$

$n = \text{total number of messages}$

Hence we can assert that the inter-object complexity measure depends on the number of objects and the number of messages exchanged between the objects.

### 3.4 Complexity of Object-Oriented Software (COOS)

Total complexity of an object-oriented software is the sum total of the cognitive information complexity of each class, cognitive information complexity of the main function and the inter-object complexity. This can be expressed as

$$\text{COOS} = \text{CICMC} + \text{CICMMF} + \text{IOCM}$$

Where:

- CICMC = Cognitive Information Complexity Measure of Classes,
- CICMMF = Cognitive Information Complexity Measure of Main Function and
- IOCM = Inter-Object Complexity Measure.

The above complexity measure is illustrated with the help of an example taken from the book titled “Object-Oriented Programming with C++” authored by E. Balagurusamy, is illustrated in figure 1.

```c
#include <iostream.h>
class student
{
protected:
    int roll_number;
public:
    void get_number(int);
    void put_number(void);
};
void student :: get_number(int a)
{
    roll_number = a;
}
void student :: put_number()
{
    cout << “Roll Number: “ << roll_number << “\n”;}
class test : public student
{
protected:
    float sub1;
    float sub2;
public:
    void get_marks(float x, float y)
    { sub1 = x; sub2 = y; }
    void test :: put_marks()
    {
        cout << “Marks in SUB1 = “ << sub1 << “\n”;;
        cout << “Marks in SUB2 = “ << sub2 << “\n”;;
    }
class result : public test
{
    float total;
public:
    void display(void)
    {
        total = sub1 + sub2;
        put_number();
        put_marks();
        cout << “Total = “ << total << “\n”;
    }
```
Fig. 1: Program in C++

4 Calculation of cognitive complexity for the above program

STEP 1: Calculate the cognitive information complexity of all the classes.

**CICM of Class Student**

LOC = 11
BCS (sequence) \( W_1 = 1 \)
SBCS = \( W_1 = 1 \)
WICS = \[1/8 + 1/6 + 1/5 + 1/3 + 2/2\] = 1.185
(The weight associated with identifier names is 1).
\[\text{CICM}_1 = \text{WICS} \times \text{SBCS} = 1.185 \times 1 = 1.185\]

**CICM of Class Test**

LOC = 16
BCS(sequence) \( W_1 = 1 \)
SBCS = \( W_1 = 1 \)
WICS = \[1/12 + 1/11 + 1/9 + 1/8 + 2/6 + 4/5\] = 1.535
(The weight associated with identifier names is 1).
\[\text{CICM}_2 = \text{WICS} \times \text{SBCS} = 1.535 \times 1 = 1.535\]

**CICM of Class Result**

LOC = 13
BCS(sequence) \( W_1 = 1 \)
SBCS = \( W_1 = 1 \)

\[\text{WICS} = [1/10 + 1/8 + 3/4] = 0.975\]
(The weight associated with identifier names is 1).
\[\text{CICM}_3 = \text{WICS} \times \text{SBCS} = 0.975 \times 1 = 0.975\]

\[\text{CICMC} = \text{CICM}_1 + \text{CICM}_2 + \text{CICM}_3 = 1.185 + 1.535 + 0.975 = 3.695\]

STEP 2: Calculate the cognitive information complexity of main function.

**CICM of Main Function**

LOC = 7
BCS (sequence) \( W_1 = 1 \)
BCS (function call 1) \( W_2 = 2 \)
BCS (function call 2) \( W_3 = 2 \)
BCS (function call 3) \( W_4 = 2 \)
SBCS = \( (W_1 + W_2 + W_3 + W_4) = 7 \)
WICS = \[1/8 + 1/6 + 1/5 + 1/3 + 2/2\] = 1.158
(The weight associated with identifier names is 1).
\[\text{CICMF} = \text{WICS} \times \text{SBCS} = 1.158 \times 7 = 11.06\]

STEP 3: Calculate the inter-object complexity.

For calculating the inter-object complexity, we first construct the collaboration diagram as shown in figure 2.

Once we have the collaboration diagram, the IOCM is measured.

\[\text{IOCM} = -[3/5\log_2(3/5) + 3/5\log_2(3/5) + 2/5\log_2(2/5) + 2/5\log_2(2/5)]\]
\[= 1.345\]
Cognitive Information Complexity Measure of the above object-oriented software is therefore sum of the complexities as calculated in step 1, 2 and 3.

\[
\text{CICM} = 3.695 + 11.06 + 1.345 \\
= 15.385
\]

The complexity measure of the above object-oriented program can only be termed as reliable and robust only if it can be shown that if the same program written in a procedural language has a higher complexity measure. To verify this, the same program is illustrated in figure 3.

**Calculation of CICM for the above Program**

- \( \text{LOC} = 17 \)
- \( \text{BCS(sequence)} = W_1 = 1 \)
- \( \text{BCS(function call)} = W_2 = 2 \)
- \( \text{SBCS} = (W_1 + W_2) = 3 \)
- \( \text{WICS} = \frac{4}{14} + \frac{1}{12} + \frac{2}{10} + \frac{3}{9} + \frac{2}{5} + (\frac{3}{4})*4 + \frac{2}{2} + (\frac{2}{2})*4 + \frac{1}{1} \)
  \[
  = 10.3
  \]

\[
\text{CICM} = \text{WICS} \times \text{SBCS} \\
= 10.3 \times 3 \\
= 30.9
\]

For the same problem program written in object-oriented language carries lower cognitive complexity as compared to the program written in a procedural language. This establishes the cognitive information complexity measure proposed in this work as a reliable measure.

**6 Conclusion**

Our work applies cognitive information complexity measure on the object-oriented program to calculate its complexity. Our proposed cognitive complexity measure adheres to the literature and practice that programs written in object-oriented languages are easy to comprehend and are best suited to model the real world objects. The work establishes that the cognitive complexity measure of a program written in object-oriented language is lower than the complexity of the same written for a procedural language thus establishing itself as a robust one.
main()
{
    int roll_number, marks1, marks2, total;
    printf("Enter the roll number");
    scanf("%d", &roll_number);
    printf("Enter the marks");
    scanf("%d%d", &marks1, &marks2);
    total = addmarks(marks1, marks2);
    printf("Roll Number = %d marks1 = %d marks2 %d", roll_number, marks1, marks2);
    printf("Total = %d", total);
}
addmarks(m1,m2)

int m1, m2, sum;
{
    sum = m1 + m1;
    return(sum);
}

Fig.3: Same Program written in ‘C’

References: