Grp_103.pdf

by

Submission date: 22-Apr-2022 08:53PM (UTC+0530)
Submission ID: 1817354903
File name: Grp_103.pdf (322.5K)
Word count: 5067
Character count: 28517

Coupling Complexity Metric: A Cognitive Weight of Polymorphism, Encapsulation, Method Hiding and Attribute Hiding

IT20272104 -Dissanayaka D.D.D
IT20116606 -Fernando W.H.K
IT20136710 -K.M.K.R Kariyapperuma
IT20258030 -Gunawardhana H.P.M.N

Abstract – As the paradigm grows in popularity, analyzing object-oriented systems in order to determine their quality becomes extremely important. As a result, many object-oriented metrics for analyzing various aspects of these systems, such as class coupling, have been introduced. This research provides a new cognitive complexity metric as cognitive weighted coupling within objects for analyzing coupling in object-oriented system and new 4 factors which are related to object-oriented concepts. In computing CWCBO, five types of coupling between classes are considered: control coupling, global data coupling, internal data coupling, data coupling, and lexical content coupling. Polymorphism, method hiding, and attribute hiding are the other three couplings. Another object-oriented strategy is encapsulation. Here we introduce calculations to calculate the complexity based on 4 new factors in addition to the existing ones, which are given below in detail. These four factors are the cognitive weight of polymorphism factor (CWOPF), the Cognitive weight of Encapsulation complexity Metrix(CWECM), the cognitive weight of method hiding factor (CWOMHF), and Cognitive Weight Attribute hiding complexity Metrix(CWAHCM). Of these, CWOPF and CWECM are derived as object-oriented factors and CWOMHF and CWAHCM as non-object object-oriented factors.

1.Introduction

The functional complexity that the code is attempting to enable is a natural byproduct of software complexity. The complexity of software systems can sometimes grow out of control due to multiple system interfaces and complex requirements, making applications and portfolios overly expensive to maintain and risky to enhance. If left unchecked, software complexity can spiral out of control in completed projects, resulting in bloated, inefficient applications. As for the benefits, it can be difficult and time consuming to identify the architectural hotspots where risk and cost originate without the use of reliable software complexity metrics. More importantly, continuous software complexity analysis allows project teams and technology management to stay ahead of the problem and avoid the emergence of excessive complexity.

Based on the cognitive informatics foundation that "cognitive complexity of software is dependent on three fundamental factors: inputs, outputs, and internal processing," cognitive complexity measures attempt to quantify the effort or degree of difficulty in comprehending the software. A mathematical model of a program is a model that depicts the program's executive step. The number of variables with high information complexity is determined by the value change of the variable, and the ability to recognize the value change is determined by the scope of the variable. Based on the assumption that variables and operators contained information, they defined information complexity as the number of

variables and information of the program. However, cognition of a program is cognition of a function, class, module, and file in cognition and understanding of a program; they are implemented with source code, and it is ultimately cognition of source code.

Because software developed using this technique is usually easier to maintain, object orientation is now a widely used approach in software engineering. Dynamic binding, encapsulation, inheritance, interaction, polymorphism, and reusability are all powerful features of object-oriented (OO) software development. "Chidamber and Kemerer "(CK) metrics suite [1], MOOD metrics for OO Design [2], design metrics for testing [3], product metrics for object-oriented design [4], Lorenz and Kidd metrics [5, Henderson–Seller et al. metrics [6, (slightly) modified CK metrics [7], and size estimation of OO systems [8]" are some of the metrics that have been proposed for evaluating OO software. Specific phases of software development, such as design, implementation, and testing, are frequently the focus of these metrics. They also cover some OO language characteristics and quality attributes, including efficiency, correctness, integrity, flexibility, maintainability, interoperability, reliability, portability, reusability, usability, and testability [9]. Maintainability is regarded as the most important attribute of software products across all attributes [10]. Complexity metrics can be used to predict critical information about the reliability and maintainability of software systems from automatic source code analysis [11]. Here we have studied two object-oriented concepts polymorphism and encapsulation and two non-object-oriented factors methods hiding and attribute hiding through the concept of cognitive weight. Based on the data obtained, we presented four new equations separately. For some equations, certain constants were used based on existing data. We have shown the relevant variables in the equations separately.

The aim of this research is proposing improvements to AA CWCBO measure (Cognitive Code-Level OO Complexity Measure). This includes both theoretical and empirical validation, as well as a comprehensive comparative analysis. The rest of this paper is structured as follows: Body consists with six sections. They are CBO (coupling between object), Cognitive Weighted Coupling Between Objects (CWCBO), Cognitive Weight Of Polymorphism Factor (CWOPF), Cognitive weight of Encapsulation complexity Metrix(CWECM), Cognitive Weight Of Method Hiding Factor (CWOMHF), Cognitive weight Attribute hiding complexity Metrix(CWAHCM).After the body section there is an analytical description of the conclusion. Finally, under the references, we stated quotations we use to find the information.

2.BODY

CBO(coupling between object)

A class's Coupling Between Objects (CBO) is a tally of how many other classes it is linked to. In three ways, this term is adaptable. Which way a class is connected to another The way in which one class is linked to another The value assigned to a coupling connection to distinguish it from other coupling relationships. CBO calculate only outward coupling to foreign classes. The manner two classes are connected will be defined in the same way as it was in the previous section. The coupling will be assigned a value of one by default, but this study will experiment with different alternative values as well. The suggested metric's novel feature will be these variations.Various varieties of Coupling have been proposed by Edward Berard [12], which are defined as follows: The motivation acquired from the literature review is discussed in the next section.

1.Control Coupling: This is the process of passing control flags between modules such that one module can control the sequence of processing steps in another.

2.Global Data Coupling: When two or more modules share the same global data structures, this is known as global data coupling.

3.Internal Data Coupling: One module changes the local data of another module directly.

4.Data Coupling: One module's output becomes the input for another. Parameter lists are used to convey items between functions.

5.Lexical Content Coupling: The contents of 1 module are partially or completely included within the contents of another.

Cognitive Weighted Coupling Between Objects (CWCBO)

The suggested measure, dubbed Cognitive Weighted Coupling Between Objects (CWCBO), takes into account the cognitive complexity of several types of coupling, such as data, control, global, and interface coupling. "Unnecessary object coupling reduces the reusability of connected objects," and "unnecessary object coupling raises the possibilities of system corruption when one or more related objects are changed. "The current CBO metric proposed by C.K counts the number of items associated with the present classes. A weight of one is assigned to each couple. The various types of coupling are not taken into account in this metric. CWCBO can be calculated using the following equation:

$$CBO = CC + GDC + IDC + DC + LCC$$

$$CWCBO = (CC * WFCC) + (GDC * WFGDC) + (IDC * WFIDC) + (DC * WFDC) + (LCC * WFLCC)$$

Control Coupling is found in a total of modules(CC).
Global Data Coupling Count (GDC)
Internal Data Coupling Count (IDC)
Data Coupling Count (DC)
Lexical Content Coupling (LCC) count

Each type of coupling's Weighting Factor is calibrated using the procedure described in the preceding section, and the values are as follows:

Control Coupling Weighting Factor (WFCC)
WFGDC Weighting factor of global data coupling
WFIDC Weighting Factor of internal data coupling
WFDC weighting factor of data coupling
WFLCC weighting factor of lexical content coupling

Using an existing comparison test some weighting factors values can be taken as follows,

WFGDC	1
WFIDC	2
WFDC	3
WFLCC	4

```
import java.io.*;

class Finalexam
{
  DataInputStream in=new DataInputStream(System.in);
  int studentno,totalmark;
  String name,examtype;

                        //GLOBAL DATA  COUPLING (Sharing global variables)

void getdata()throws IOException
{
  System.out.println("Enter the Student No:");
  studentno=Integer.parseInt(in.readLine());
  System.out.println("Enter the Exam Type:");
  examtype=in.readLine();
  System.out.println("Enter the Student Name:");
  name=in.readLine();
  System.out.println("Enter the Final Exam mark:");
  totalmark=Integer.parseInt(in.readLine());
}
void display()
{
  System.out.println("******************");
  System.out.println(" Student No:"+studentno);
  System.out.println("Exam Type:"+examtype);
  System.out.println("Student Name:"+name);
  System.out.println("Total exam mark:"+totalmark);
}

class repeatMark extends Finalexam {
  int repeatmark;
void getin()throws IOException
{
```

```
  System.out.println("Enter the mark of repeat subjec:");
  repeatmark=Integer.parseInt(in.readLine());
}
  void show() {
if(repeatmark<=totalmark)  //DATA COUPLING (passing variables again for use )
 {
totalmark-=repeatmark;
System.out.println("Total pass marks of  after decrese repete subject mark:"+totalmark);
System.out.println(" ******************");
}
else   //CONTROL COUPLING//(using true or false values)
{
System.out.println("You cannot decrese this mark"); System.out.println(" **********************");
}}}

class repeatexam extends Finalexam
{
int repeatpassmark;
void get()throws IOException
{
System.out.println("Enter the mark of repeat exam subject:"); repeatpassmark=Integer.parseInt(in.readLine());
}
void print() { totalmark+=repeatpassmark;                  //LEXICAL CONTENT COUPLING(Same content)
System.out.println("Total marks after calculate:"+totalmark;

  System.out.println("**********************"
);
}}

class MainExam {

public static void main(String args[])throws
IOException
{
DataInputStream in=new
DataInputStream(System.in);
System.out.println("Enter the choice:");
int op=Integer.parseInt(in.readLine());
      switch(op)
      {
        case 1:
        repeatMark r=new repeatMark();
          w.getdata();
          w.getin();
          w.display();
          w.show();
       break;

          case 2:
          repeatexam d=new repeatexam();
          d.getdata();   // INTERNAL DATA COUPLING(Modifying same method with   different     objects)
          d.get();
          d.display();
         d.print();
       break;
       default:

        System.out.println("Enter the choice 1 or 2");
        break;
}}}
```

CBO=1+1+1+1+1
=5

CWCBO=(1*1)+(1*1)+(1*2)+(1*3)+(1*4)
=1+1+2+3+4
=11

Let's discuss the four new factors that we have introduced.

Cognitive Weight Of Polymorphism Factor (CWOPF)

There are several object-oriented concepts. They are encapsulation, polymorphism, aggregation, abstraction so on. Among them, polymorphism is one of the important object-oriented concept. Polymorphism in object-oriented programming refers to a programming language's ability to process objects differently based on their data type or class.
In other words, the ability to redefine or implement methods or interfaces in derived classes is defined as polymorphism. This has been shown to improve extensibility and reusability [13]. Polymorphism's importance arises from the several advantages it provides in the software development life cycle, allowing for the creation of qualitatively efficient and successful software. Polymorphism's key benefit is that it provides a great level of freedom and decoupling because implementation behind an interface is concealed from clients. You only have to worry about programming the interface when it comes to coding. You can naively swap a mock object during testing and independently validate your unit of code.

Pure Polymorphism: It is accomplished by calling the same member function with different signatures within the same class scope. The number, type, and order of the arguments make the signature. It's the same as overloading methods within a class. Parametric overloading is another name for it.

2) Static Polymorphism: In a class hierarchical tree, it is composed level polymorphism. It's the same as overriding a method. It is accomplished by defining member functions with the same name but distinct signatures in separate types. Inheritance relations may or may not connect these classes. Method binding occurs at binding period for both pure and static polymorphisms.

3) Dynamic Polymorphism: It's also known as composition level polymorphism or method overriding. In contrast to static polymorphism, it is accomplished by defining member functions in child classes with the same name and signature. it's the possibility to use the same name and signature used in an overridden method [14].

The following formula can be used to calculate the complexity metrics of the polymorphism factor,

$$PF = \frac{\sum_{i=1}^{TC} M_0(C_i)}{\sum_{i=1}^{TC}[M_n(C_i) * DC(C_i)]}$$

$M_0(C_i)$:number of overriding methods in class (C_i),
$M_n(C_i)$= number of new methods defined in class (C_i),
$DC(C_i)$= number of children for class (C_i),
TC = Total number of Classes in the whole software system.

Using the above formula, we propose the following formula to calculate the complexity based on the cognitive weight of the polymorphism factor. The suggested CWPC calculates not only the architectural complexity of the polymorphism, but also the cognitive complexity resulting from the effort required to comprehend various types of polymorphism in the software system in question.

$$CWOPF = \frac{\sum_{i=1}^{TC} CWM_0(C_i)}{\sum_{i=1}^{TC}[M_n(C_i) * DC(C_i)] * ACW}$$

$CWM_0(C_i) = N_{pp} * CW_{pp} + N_{sp} * CW_{sp} + N_{dp} * CW_{dp}$
$M_n(C_i)$= number of overriding methods in class C_i
$DC(C_i)$ = number of children for class C_i
TC = Total number of Classes.
$ACW = (CW_{pp} + CW_{sp} + CW_{dp}) / 3$
N_{pp} = number of pure polymorphism
N_{sp}= number of static polymorphism
N_{dp}= number of dynamic polymorphism
CW_{pp}= cognitive weight of pure polymorphism
CW_{sp} = cognitive weight of static polymorphism
CW_{dp}= cognitive weight of dynamic polymorphism

Using an existing comparison test some cognitive values can be taken as follows,

Pure Polymorphism (PP)	3
Static Polymorphism (SP)	5
Dynamic Polymorphism (DP)	7

Considering the following code snippet let's calculate the value of CWOPF,

```
class Student{
        float v1 = 3;
        void m1(int i){ }
        void m2(char ch){ }
        float m3(){
            return 4 * v1;}
        float m4(){
            return v1 * v1}
}
 class StudentProfile extends Student {
        void m1(float f) { }
        void m2(String s) { }
        }
 class StudentMark extends Student {
        float m3(){
            return 2 * 3.14 * v1;}
        float m4(){
            return v1 * v1 * v1;}
        int m5(int a, int b) {
            return a+b;}
        int m6(int a, int b) {
```

```
        return a * b;}
    }
class StudentReport extends StudentMark {
    }
class Subject  extends StudentMark {
    }
```

$$CWOPF = \frac{Mo(C1) + Mo(C2) * CWsp + Mo(C3) * CWdp + Mo(C4) + Mo(C5)}{(Mn(C1) * 4 + Mn(C2) * 0 + Mn(C3) * 2 + Mn(C4) * 0 + Mn(C5) * 0) * ACW}$$

$$CWOPF = \frac{0 + 2 * 5 + 2 * 7 + 0 + 0}{(4 * 4 + 0 + 2 * 2 + 0 + 0) * 5} = 0.24$$

Cognitive weight Of Encapsulation complexity Metrix(CWECM)

The object-oriented language was used as a successful language among other languages of the 20th century. Facilitates object-oriented language heritage, abstraction, closure, linking, polymorphism, and integration, enhancing the ability to reuse, test, maintain and extend the software. If the capsule is maximized during the design phase of the software development life cycle, encapsulation will become very popular among other OO developers. Heritage and Contact Software Development Lifecycle reduces the impact of coverage on software. Based on a literature survey of software engineering [15-16] it can be suggested that prompt debugging is essential for the success of a reliable software system. Software Development Lifecycle and Reliable Engineering Together allow designers to assess, test and verify reliable requirements early in the lifecycle. Table 1 of this paper illustrates the design effect on complexity and reliability. The arrow with the highest score shows more impact and the lower arrow shows less impact. If the inheritance and relevance of the class design are high, the complexity will inevitably be a high and less reliable system. On the other hand, if there is strong integration and coverage, the complexity is reduced and the software is very reliable. A class plan can be maintained by optimizing it to make the class plan system credible.

Reliability is the probability of a task failing within a specific time frame. Reliability can be described as a function of complexity. Decreased reliability increases system complexity. It can be said that reliability is inversely proportional to the complexity of the software as its reliability decreases with increasing complexity.Increasing demand for anything makes the system more buggy, unreliable, and difficult to understand. Increasing the existing complexity of the system into a more faulty system reduces the reliability of the system. Focusing on object-oriented design structures such as proposed access, legacy, connection, integration, and integration contributes to minimizing system complexity. [17]. The positive and negative effects of object-oriented structures on the complexity of the system are examined as shown in Table 1. In the previous work, it can be shown that inheritance increases the complexity of the relationship and integration decreases the complexity . Object-Oriented Design Conduct Conducted Object-Oriented Software has already been proven to improve quality. Complexity increases as design construction decreases. The complexity is therefore inversely proportional to the combination. Reliability increases when the design capsule is maximized. Reliability is a qualitative quality that is closely related to object-oriented design construction. Accordingly, an assumption can be suggested based on the following discussion: As the capsule increases, the complexity decreases.

The object-oriented design supports the encapsule. Encapsulation is the hiding of data by unauthorized access. Encapsulation can be introduced as a technology that minimizes interdependence between separate written modules by defining rigid external interfaces. [18]. Class complexity can be controlled by maximizing class design. The complexity of a closed class can be measured in two metric parts, one capsule part, and the other complex part. Here, CWECM is calculated based on the method and properties.

The CWECM calculation can be done using the following metric.

$$CWECM = \frac{\{\sum En(EM \ \& \ EA) + \sum DA(M \ \& \ A)\}}{\sum S(M\&A)} * Cw$$

E (m & a) (Encapsulated methods and attributes): - Contains methods and attributes not used by any other class.
DA (m & a) (Direct access methods and properties): - A given layout contains methods and attributes used by other classes.
T (m & a) (complete methods and properties): - It considers all the methods and properties and properties of a closed system or the access system and properties class.
CWECM (Encapsulated class complexity metric): - Encapsulated class complexity metric consists of closed method and properties and accessible methods and properties.
Increasing the number of classes will not be complicated. It can be seen that the complexity of a class plan depends on the qualities and methods of the class. A class with a large number of attributes and methods is more complex and difficult to understand. The researcher tried to control the complexity of a design by increasing the capsule of a class design. The design was taken to calculate. CWECM. The metric calculation plan has been redesigned. Both of these designs are more comparable to the proposed Metric CWECM estimate.

```
public class Marks {
  private String subject;
  private String Id;
  private int mark;

  public int getMark() {
    return mark;
  }

  public String getSubject() {
    return subject;
  }

  public String getIdNum() {
    return idNum;
  }

  public void setMark( int newMark) {
    mark = newMark;
  }

  public void setSubject(String newSubject) {
    name = newName;
  }

  public void setIdNum( String newId) {
    idNum = newId;
  }
}
```

$$CWECM = \frac{\{\sum En(EM + EA) + \sum DA(M + A)\}}{\sum S(M + A)} * Cw$$

$$CWECM = \frac{\{(0+3)+(6+0)\}}{(6+3)} * 1$$

$$= 1$$

Cognitive Weight Of Method Hiding Factor (CWOMHF)

The method hiding factor is a non-object-oriented component that is included in the information hiding. Because information hiding and encapsulation are not the same thing [19]. The instance variables and instance methods may be encapsulated, but they may still be visible to other classes and packages in part or in full. The information hiding is achieved by a combination of class encapsulation and attribute and method scoping within the class. Method hiding must be done with caution, since less method hiding can result in an implementation that isn't properly abstracted, while excessive method hiding can result in very little functionality. As a result, in order to maintain the balance between hiddenness and visibility of methods in the system, the method hiding factor must be determined more precisely. Let's take this as Cognitive Weight Of Method Hiding Factor (CWOMHF) .

The following formula can be used to calculate the complexity metrics of the method hiding factor

$$MHF = \frac{\sum_{i=1}^{TC} M_h(C_i)}{\sum_{i=1}^{TC}[M_d(C_i)]}$$

$M_d(C_i) = M_v(C_i) + M_h(C_i)$
$M_d(C_i)$ = Total number of methods defined in class C_i
$M_v(C_i)$ = Number of visible methods in class C_i
$M_h(C_i)$ = Number of hidden methods in class C_i
TC = Total number of Classes in the whole system.

Using the above formula, we propose the following formula to calculate the complexity based on the cognitive weight of the method hiding factor. The CWOMHF complexity metrics consist of cognitive complexity by taking the various types of method visibility. Visibility might be completely hidden, slightly visible, or totally visible. In Java, there are different types of visibility methods. Such as 'Private,' 'protected,' 'public,' and the default,' for example. The method only visible within a relevant class can name as private, and a method visible to the entire system can be is defined as public. The function is visible both within the class and its inheriting classes, as well as within the package and other packages, which can be named as a protected method.
The derived formula for the CWOMHF can be mathematically defined as follows,

$$CWOMHF = \frac{\sum_{i=1}^{TC} M_h(C_i)}{\sum_{i=1}^{TC}[M_h(C_i)] + \sum_{i=1}^{TC}[M_v(C_i)]}$$

$$\sum_{i=1}^{TC} M_h(C_i) = \sum_{i=1}^{TC} M_p(C_i) * CW_{pm} + M_d(C_i) * CW_{dm} + M_t(C_i) * CW_{tm}$$

$$\sum_{i=1}^{TC} M_v(C_i) = \sum_{i=1}^{TC} M_u(C_i) * CW_{um}$$

$M_p(C_i)$ = Number of private methods in class C_i

$M_d(C_i)$ = Number of default methods in class C_i

$M_t(C_i)$ = Number of protected methods in class C_i

$M_u(C_i)$ = Number of public methods in class C_i

CW_{pm} = Cognitive Weight of private methods

CW_{dm} = Cognitive Weight of default methods

CW_{tm} = Cognitive Weight of protected methods

CW_{um} = Cognitive Weight of public methods

TC = Total number of Classes in the whole system

Using an existing comparison test some cognitive values can be taken as follows,

Private Method (PM)	4
Private Method (PM)	5
Protected Method (TM)	6

Considering the following code snippet let's calculate the value of CWOMHF,

```
class ITEM{
        protected String s1, s2;
        protected String getItem(String s1){
          this.s1 = s1;
          return s1;
        }
        protected String getBrand(String s2){
          this.s2 = s2;
          return s2;
        }
        private void putItem() {
          System.out.print( s1 +" ");        }
        private void putBrand()  {
          System.out.print( s2 +" ");        }
         }

 class ITEMBILLING extends ITEM{
        private int  num;
        private float price;
        protected void getNum(int i) {
          this.num = I;
        }

protected void getPrice(float f) {

          this.price = f;
        }
        private int putNum() {
          return num;
```

```
        }
        private float putPrice() {
          return price;
        }
        void putAmount(){
          System.out.print(num*price+"  ");
        }
}

class SHOP extends ITEMBILLING{
        private int num;
        private float price;
        public int putNum() {
          return num;
        }
        public float putPrice() {
          return price;
        }
}

class CART extends SHOP{
          private String item;
          private String brand;
        public String putItem() {
                    item  = s1;
          return item;
        }
        public String putBrand() {                    brand = s2;
          return brand;
        }
}
```

$$\text{CWOMHF} = \frac{((12+8)+(12+8+5)+0+0)}{\big((20+25)+(4*1)\big)}$$
$$= 0.9184$$

Cognitive weight Attribute hiding complexity Metrix(CWAHCM)

The success of a software product depends largely on the concealment of data protection properties against any random or unwanted changes. The ability to hide attributes is achieved by encapsulating data value-holding variables. Typically, the 'class' block is used to encapsulate. But capsulation and concealment of information are not the same things. The example variable may be closed but may still be fully or partially visible to other classes and packages. Also, information can be hidden without engraving, and vice versa [20]. Therefore, attribute concealment can actually be activated by a combination of class coverage and scope of attributes. AHCM: - The ratio of the sum of the invisibility of all the properties defined in all classes to the total number of properties defined in the software system. Validation Complex Metrics The CWAHCM is derived from the AHCM Complex Metrics, which considers only the structural mechanism

of the OO paradigm and does not include the cognitive component. By combining both operational and architectural complexities, interpretive complexity can thus add cognitive complexity to the CWAHCM AHCM metric. According to an experiment conducted by Francis Thamburaj and Aloysius, the cognitive load (CW) [21] has already been calibrated. Accordingly, the cognitive burden for a private attribute is 2; For the default attribute 3; 4 for the security attribute, and 1 for the general attribute. The complexity is calculated by substituting these CW loads for this derived formula.

$$AHCM = \frac{\sum_{i=1}^{n} AT(CL_n)}{\sum_{i=1}^{n} AT(CL_n) + AV(CL_n)}$$

AHCM: −Attribute hiding complexity Metrix
$\mathrm{AT(CL_n)}$: −Nuumber of hidden attribute in class$\mathrm{(CL_n)}$
$\mathrm{AV(CL_n)}$: −number of visible attribute in class $\mathrm{(CL_n)}$
n: −number of classes in the system

$$\mathrm{CW}AHCM = \frac{\sum_{i=1}^{n} AT(CL_n)}{\sum_{i=1}^{n} AT(CL_n)+AV(CL_n)}$$

$$\sum_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{AT(CL_n)} = \sum_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{AP(CL_n)} * \mathrm{CW_{pa}} + \mathrm{AD(CL_n)} * \mathrm{CW_{da}} + \mathrm{AR(CL_n)} * \mathrm{CW_{ta}}$$

$$\sum_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{AV(CL_n)} = \sum_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{AU(CL)} * \mathrm{CW_{ua}}$$

$\mathrm{AP(CL_n)}$: −Nuumber of private attribute in class$\mathrm{(CL_n)}$
$\mathrm{AD(CL_n)}$: −Nuumber of default attribute in class$\mathrm{(CL_n)}$
$\mathrm{AR(CL_n)}$: −Nuumber of protected attribute in class$\mathrm{(CL_n)}$
$\mathrm{AU(CL_n)}$: −Nuumber of public attribute in class$\mathrm{(CL_n)}$

$\mathrm{(CW_{pa})}$: −cognactive weight of private attribute
$\mathrm{(CW_{da})}$: −cognactive weight of default attribute
$\mathrm{(CW_{ta})}$: −cognactive weight of protected attribute
$\mathrm{(CW_{ua})}$: −cognactive weight of public attribute

```
class Midmark{
private int mark1=10;
private double mark2=10;
private float  mark3=10.4f;
}


class Repeatmark{
private int rmark1=20;
private float rmark2=7.7f;
private String rmark3="THOMAS";
private double rmark4=3.0,d3=2.0;

}
```

$$AHCM = \frac{(3+5)}{(3+5)}$$
$$= 1$$
$$CWAHCM = \frac{(1*2+1*4)+(2*2+2*4+1*3)}{(1*1)+(1*2+1*4)+(2*2+2*4+1*3)}$$
$$= 0.955$$

3.Conclusion

A CWCBO metric has been developed to assess class level complexity. Class complexity includes class complexity. CWCBO includes the cognitive complexity of different types of connections. CWCBO has shown that class complexity affects the cognitive weight of different types of relationships. The Cognitive Test was used to validate the cognitive loads given by the four coupling types, and it was found that the cognitive loads required to understand the LCC were higher than the CC, GDC and IDC. A case study and a comparative study were used to test the measure and found that it was a better indicator of class-level complexity. Calculated using a technology developed by CWCBO. A new complexity measurement, called cognitive weight of polymorphism factor (CWOPF), has been proposed to measure class-level complexity. The cognitive weight of polymorphism factor captures not only the structural complexity but also the cognitive complexity. The new polymorphism complexities were calibrated using a series of metric comprehension tests. CWECM is proposed to measure the complexity of the class plan. It has been introduced that high coverage reduces the complexity of class design and therefore concludes that the complexity of object-oriented software can be maintained by increasing the encapsulation of design. This paper proposes and mathematically defines a new complexity metric that is the concealing factor of the cognitive weight system for measuring class-level complexity. The hidden factor in the cognitive system is not only the structural complexity, but also the cognitive complexity that arises from the time and effort it takes to understand the software. The cognitive load was calibrated using a series of cognitive tests, and it was found that the cognitive load varies sequentially from the default, default and private domains to the different domain scope used to hide system visibility in other classes. The new CWOMHF complexities are meticulously broader in nature and more realistic than realistic. The CWAHCM Complexity Metric has shown a better and broader index. The calculations related to the equations obtained using case studies are performed and final conclusions are reached.

4.Reference

[1] S. R. Chidamber and C. F. Kemerer, ‘‘A metrics suite for object-oriented design,’’ IEEE Trans. Softw. Eng., vol. 20, no. 6, pp. 476–493, Jun. 1994.
[2] R. Harrison, S. J. Counsell, and R. V. Nithi, ‘‘An evaluation of the MOOD set of object-oriented software metrics,’’ IEEE Trans. Softw. Eng., vol. 24, no. 6, pp. 491–496, Jun. 1998.
[3] R. V. Binder, ‘‘Design for testability in object-oriented systems,’’ Comms. ACM, vol. 37, no. 9, pp.

87–101, Sep. 1994.
[4] S. Purao and V. Vaishnavi, ‘‘Product metrics for object-oriented systems,’’ ACM Comput. Surveys, vol. 35, no. 2, pp. 191–221, Jun. 2003.
[5] M. Lorenz and J. Kidd, Object-Oriented Software Metrics: A Practical Guide. Upper Saddle River, NJ, USA: Prentice-Hall, 1994.
[6] B. Henderson-Sellers, L. L. Constantine, and I. M. Graham, ‘‘Coupling and cohesion (towards a valid metrics suite for object-oriented analysis and design),’’ Object Oriented Syst., vol. 3, no. 3, pp. 143–158, Sep. 1996.
[7] V. R. Basili, L. C. Briand, and W. L. Melo, ‘‘A validation of object-oriented design metrics as quality indicators,’’ IEEE Trans. Softw. Eng., vol. 22, no. 10, pp. 751–761, Oct. 1996.
[8] G. Costagliola, F. Ferrucci, G. Tortora, and G. Vitiello, ‘‘Class point: An approach for the size estimation of object-oriented systems,’’ IEEE Trans. Softw. Eng., vol. 31, no. 1, pp. 52–74, Jan. 2005.
[9] S. L. Pfleeger and J. M. Atlee, Software Engineering: Theory and Practice. Chennai, India: Pearson Ed., 1998.
[10] I. Sommerville, Software Engineering. Reading, MA, USA: Addison-Wesley, 2004.
[11] T. J. McCabe and A. H. Watson, ‘‘Software complexity,’’ Crosstalk, vol. 7, no. 12, pp. 5–9, 1994.
[12] Edward Berard. V —Essays on object-orientedsoftware engineering (vol. 1)| Berard Software Engineering, Prentice-Hall, 1993, ISBN:0-13-288895-5, 1993
[13] T. G. Mayer, T. Hall, “Measuring OO systems: a critical analysis of the MOOD metrics,” Tools 29, (Procs. Technology of OO Languages & Systems, Europe’ 99), R. Mitchell, A. C. Wills, J. Bosch, B. Meyer (Eds.): Los Alamitos, Ca., USA, IEEE Computer Society, pp. 108–117, 1999.
[14] S. Benlarbi, and W. L. Melo, “Polymorphism measures for early risk prediction,” IEEE Software Engineering, 1999. Proceedings of the 1999 International Conference, pp. 334-344, 1999.
[15]. S.Singh, K.S. Kahlon, P.S. Sandhu, “Re-engineering to analyze and measure object oriented paradigms ”, The 2nd IEEE
International Conference on Information Management and Engineering , **ISBN:** 978-1-4244-5263-7,16-18 April 2010,p: 472 –
478,DOI: 10.1109/ICIME.2010.5478013 .
[16]. N.Sharygina, J. C. Browne, R. P. Kurshan, “A Formal Object-Oriented Analysis for Software Reliability: Design for
Verification”, Proceedings of the 4th International Conference on Fundamental Approaches to Software Engineering ISBN:3-
540-41863-6, 2011, p: 1-15.
[17] Yong Cao Qingxin Zhu, “Improved Metrics for Encapsulation Based on Information Hiding”, DOI: 10.1109/ICYCS.2008.76,
The 9th International Conference for Young Computer Scientists, IEEE computer society, 2008, p: 742-724.
[18] Nathanael Sch aril, Andrew P. Black,and St´ephane Ducasse, “ Object oriented Encapsulation for Dynamically Typed
Languages”, OOPSLA’04 Vancouver, BC, Canada, p. 130–139, Oct. 2428, 2004, ACM 1581138318/04/0010.
[19]Tahvildari, Ladan, and Ashutosh Singh, “Categorization of object-oriented software metrics,” Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering, Halifax, Nova Scotia, pp. 235–239, May 2000.
[20] Harrison, R., Counsel, S.J. and Nithi, R.V., “An Evaluation of the MOOD Set of Object-Oriented Software Metrics,”IEEE Tr.on Software Eng., 6, 491-496, 1998
[21] Francis Thamburaj and A. Aloysisus, “Cognitive Perspective of Attribute Hiding Factor Complexity Metric,”InternationalJournal of Engineering and Computer Science, ISSN: 2319-7242,11, 14973-14979, November 2015

Grp_103.pdf

ORIGINALITY REPORT

38%	32%	22%	9%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	docplayer.net Internet Source	9%
2	www.internationalscienceindex.org Internet Source	6%
3	cyberleninka.org Internet Source	4%
4	ijcsse.org Internet Source	3%
5	Submitted to Kingston University Student Paper	2%
6	arxiv.org Internet Source	1%
7	eprints.covenantuniversity.edu.ng Internet Source	1%
8	Yadav, A., and R.A. Khan. "Development of Encapsulated Class Complexity Metric", Procedia Technology, 2012. Publication	1%
	shcpub.edu.in	

9	Internet Source	1 %
10	Submitted to CTI Education Group Student Paper	1 %
11	Sanjay Misra, Adewole Adewumi. "chapter 57 Object-Oriented Cognitive Complexity Measures", IGI Global, 2018 Publication	1 %
12	Submitted to University of Bath Student Paper	1 %
13	fdocuments.in Internet Source	1 %
14	www.ijarcsse.com Internet Source	1 %
15	Submitted to Universiti Teknologi Petronas Student Paper	1 %
16	ijsrcseit.com Internet Source	1 %
17	Submitted to Sri Lanka Institute of Information Technology Student Paper	<1 %
18	documents.mx Internet Source	<1 %
19	Sanjay Misra, Adewole Adewumi, Luis Fernandez-Sanz, Robertas Damasevicius. "A	<1 %

	Suite of Object Oriented Cognitive Complexity Metrics", IEEE Access, 2018 Publication	
20	file.allitebooks.com Internet Source	<1 %
21	serialsjournals.com Internet Source	<1 %
22	Lecture Notes in Computer Science, 2012. Publication	<1 %
23	Matthias Tamm. "Chelate Complexes of Functionalized Cycloheptatrienyl Ligands: 17- and 18-Electron Molybdenum Complexes with Linked Cycloheptatrienyl–Phosphane Ligands and Their Use in Transition Metal Catalysis", European Journal of Inorganic Chemistry, 03/2002 Publication	<1 %
24	www.eli.sdsu.edu Internet Source	<1 %
25	Submitted to Florida Institute of Technology Student Paper	<1 %
26	Shiu Lun Tsang, S. Clarke, E. Baniassad. "An evaluation of aspect-oriented programming for java-based real-time systems development", Seventh IEEE International Symposium onObject-Oriented Real-Time	<1 %

	Distributed Computing, 2004. Proceedings., 2004 Publication	
27	thinkmind.org Internet Source	<1 %
28	www.cs.umu.se Internet Source	<1 %
29	www.lbd.dcc.ufmg.br Internet Source	<1 %
30	Sun-Jen Huang, R. Lai. "Measuring the maintainability of a communication protocol based on its formal specification", IEEE Transactions on Software Engineering, 2003 Publication	<1 %
31	es.scribd.com Internet Source	<1 %
32	"Advances in Computer Science, Engineering & Applications", Springer Science and Business Media LLC, 2012 Publication	<1 %
33	Ayaz Farooq, Rene Braungarten, Reiner R. Dumke. "An Empirical Analysis of Object-Oriented Metrics for Java Technologies", 2005 Pakistan Section Multitopic Conference, 2005 Publication	<1 %

34 Benjapol Auprasert, Yachai Limpiyakorn. "Towards Structured software Cognitive complexity measurement with Granular Computing strategies", 2009 8th IEEE International Conference on Cognitive Informatics, 2009
Publication

<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		

Grp_103.pdf

GRADEMARK REPORT

FINAL GRADE

/0

GENERAL COMMENTS

Instructor

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15