New Cognitive weighted complexity measurement: object-oriented factors

First author – D. I. De Silva
Second author – Samitha Vidhanaarachchi
Third author – Nethu Nipuna M. IT20204648
Fourth author – Abeywardhanage S.R.D IT20046552
Fifth author – Tharuka Gayashan F. IT20186906
Sixth author – V.G.A.P. Kumara IT20068578

Abstract - Software metrics should be utilized to improve software productivity and quality since they give essential information about the system's stability and maintainability. We offer a cognitive complexity metric for evaluating the design of object-oriented programs in this research. Based on the knowledge gained from the Cyclomatic Complexity (CC) and Weighted Composite Complexity (WCC) metrics, the following proposed new object-oriented (OO) complexity measures are invoked. The research group has chosen the following four factors to propose a new complexity metric. Encapsulation, Abstraction, Polymorphism override, Polymorphism overload. Based on the above-mentioned parameters, the newly proposed complexity metric can be used to measure the difficulty of each program statement, line by line. Furthermore, this metric can be used to quickly create a complexity measurement tool.

I. INTRODUCTION

Software complexity can be defined as the basic driver of software cost, reliability, and functionality. Software complexity can be classified as cognitive complexity, cyclomatic complexity, Inheritance complexity, Control flow complexity, Functional complexity etc. Software measurement complexity makes it possible to understand what computer programs are difficult to understand. Software complexity measurements provides opportunity to improve code quality, reduce maintenance cost, improve high productivity, meet architecture stands, increase robustness. A software complexity measurement regime should be implemented for any organization that seeks to increase the speed of software distribution. Knowing the level of complexity of the code while maintaining greater predictability makes it easier to know how much maintenance a program need. Software Risk Minimization Software complexity management reduces the risk of introducing errors into a product. The most important advantage of measurement is an unbiased and accurate understanding of reality and reduce complexity of code.

Cognitive Informatics is a emerging topic in research today. These cognitive informatics are used in research fields such as science, informatics, mathematics, computer science neurobiology, physiology and software engineering. The significance of the Cognitive Informatics experiment is that it seeks to solve common problems in two areas in a two - pronged and multidisciplinary approach. Cognitive Informatics is also used to solve problems in the medical and computer fields. Also measurements in the field of software engineering are still being created. Over the past few years, researchers have been identifying problems in software engineering and presenting principles in cognitive science and measurements. This is why several object oriented cognitive complexity measurements took effort.[23] The importance of cognitive informatics to software engineering is that software can take measurements and create less complex programs.

Literature of Cognitive Code Level Complexity

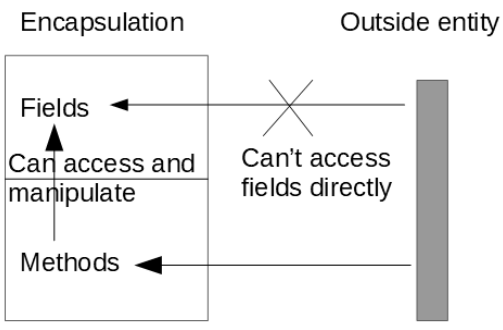
A variety of cognitive code-level (CCL) object-oriented (OO) complexity measures have been proposed in the literature [1], [2], [3],[4]. The cognitive complexity of a unit of code is a measure of how difficult it is to grasp intuitively. Unlike Cyclomatic Complexity, which defines how difficult it will be to test your code, Cognitive Complexity determines how difficult it will be to read and understand source code. A survey on existing CCL OO complexity measures can be found in to locate complicated code, code complexity measurements are utilized. The code complexity should be determined as early as feasible in the coding process in order to obtain high-quality software with cheap testing and maintenance costs. When recommended values are surpassed, the developer might adapt his code [5]. The code-based complexity metric includes Mac and the Halstead Complexity Measure Cabe's Metrics for Cyclomatic Complexity and Lines of Code. They also cover some OO language characteristics and quality aspects, including as efficiency, correctness, integrity, flexibility, maintainability, interoperability, reliability, portability, reusability, usability, and testability [6]. Maintainability is regarded as the most important attribute of software products [7] among the aforementioned attributes.

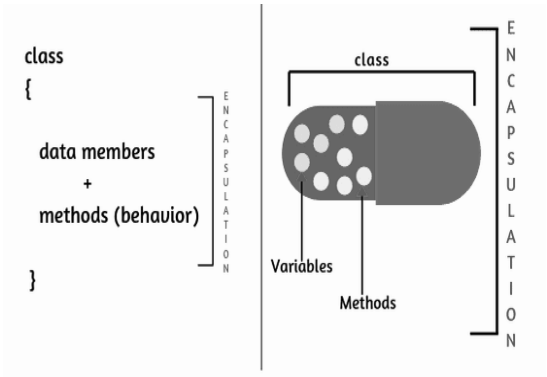
Cognitive code level complexity measures can be used to anticipate key information regarding the dependability and maintainability of software systems from computerized source code analysis [8]. Due to the maintainability of OO software, Object-Oriented (OO) techniques have dominated the software industry for the past two decades. Design quality aids researchers in evaluating the software's maintainability through the use of software metrics based on quantitative means. Any changes to the design after it has been completed and implemented result in increased complexity and costs at the conclusion of the software development. To address the aforementioned issues, it is vital to thoroughly study the design in order to determine its efficacy before finishing it [9]. In the literature, there are a number of software metrics that can be used to calculate complexity from various perspectives. Chidamber & Kermerer (CK) metrics suite [9], metrics of OO design (MOOD) [10], Lorenz and Kidd metrics [11], modified CK metrics [12], product metrics for OO design [13], [14], weighted class complexity metric [15], cognitive code complexity of inheritance for OO software [16], and an OO cognitive complexity metric [17] are a few of them. The aforementioned software metrics are related to OO design software, which indicates some software quality attributes, and each of these metrics has its own set of advantages and disadvantages. Furthermore, providing new software complexity measurements or improving the performance of existing ones is always welcome in order to attain improved software quality.

"New Cognitive weighted complexity measurement: object-oriented factors" research topic included about how to measure cognitive weight complexity using new mathematical equation. We used four object-oriented factors to calculate the cognitive weight complexity. We used static polymorphism factor [overloading], dynamic polymorphism factor [overriding], abstraction factor, encapsulation factor. All the factors' conditions used for creating new equation. We could improve the cognitive weight complexity measurements under this research topic.

Encapsulation in OOP with example

Encapsulation is one of the four important OOP concepts. Encapsulation is a mechanism for combining variables(data) and methods (code) into a single unit. It is better illustrated by the picture below.





Encapsulation is used to hide the values or status of a structured data object in a class, to prevent unauthorized access to it. It is the process of hiding information and protecting the data and behavior of the object. Publicly accessible methods are usually provided for access to class (so-called getters and setters) values, and these methods are used to obtain and modify values within other client class objects. Encapsulation can be achieved by declaring all the variables in the class as private and writing public methods in the class to set and get the values of the variable. It is more defined with setter and getter methods. Encapsulations perform very important role in OOP concepts. It controls the way of data accessibility, Modifies the code based on needs, helps us to achieve a loose couple, Gets the simplicity of our application and it also allows to modify a portion of the code without interrupting other functions or code in the program. Data hiding, increased flexibility, reusability Testing code is easy are benefits of encapsulation. In data hiding, user has no idea about the internal implementation of the class. The user does not even know how to store values in class variables. User only knows that we are sending a value processing system and with that value the variables start. Encapsulate class is easy to test, so it is also better to test units. Encapsulation also improves the re-usability and is easy to change with new requirements.

Abstraction in OOP

Object-oriented programming (OOP) languages use abstraction as one of its main ideas. Its major purpose is to deal with complexity by obfuscating superfluous information from the user. This allows the user to build more complicated logic on top of the offered abstraction without having to comprehend or consider all of the underlying complexity.

Interfaces and abstract classes are used in Java to accomplish abstraction. Using interfaces, we can accomplish complete abstraction.

Abstract methods and abstract classes:

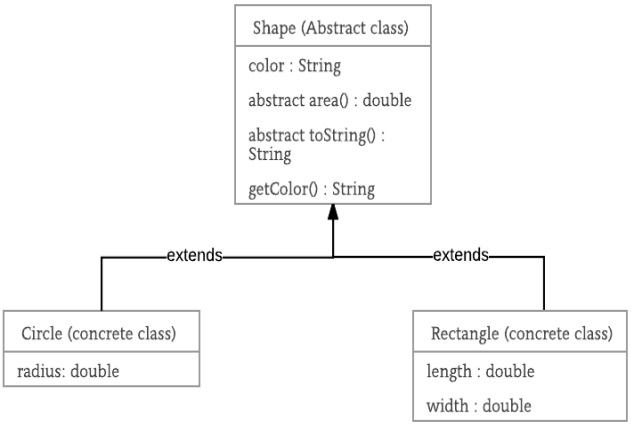
- An abstract class is one that has the abstract keyword in its name.

- The term "abstract method" refers to a method that is defined but not implemented.
- All abstract methods may or may not be present in an abstract class. Some of these might be concrete procedures.
- A method that is specified as abstract must always be redefined in the subclass, forcing overriding OR making the subclass abstract.
- Any class having one or more abstract methods must also include the abstract keyword in its declaration.
- An abstract class can't have any objects. An abstract class, on the other hand, cannot be directly created with the new operator.
- The default constructor is always present in an abstract class, and it can contain parameterized constructors.

An example of when to utilize abstract classes and abstract methods

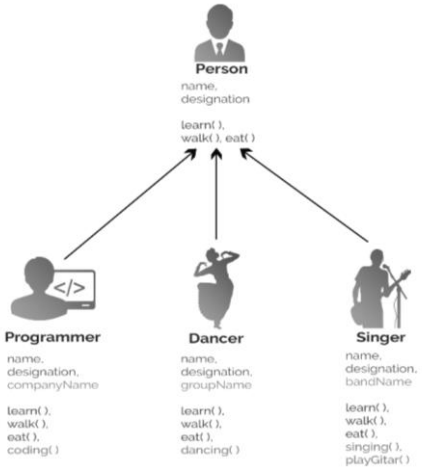
In certain cases, we'll wish to construct a superclass that specifies the structure of an abstraction without giving a complete implementation of all methods. That is, there will be occasions when we wish to construct a superclass that just defines a generalization form that all of its subclasses will use, allowing each subclass to fill in the details.

Consider the famous "shape" example, which may be used to a computer-aided design system or a gaming simulation. "Shape" is the base kind, and each shape has its own color, size, and other characteristics. Specific sorts of shapes—circle, square, triangle, and so on—are derived(inherited) from this, each with its own set of properties and behaviors. Certain forms, for example, may be turned inside out. When you wish to determine the area of a form, for example, some behaviors may change. The type hierarchy encapsulates the forms' similarities and variances.



Inheritance in OOP

Inheritance is a mechanism that allows one object to inherit all of the characteristics and actions of its parent object. Inheritance is one of the most significant characteristics of Object-Oriented Programming. The IS-A relationship, often known as a parent-child relationship, is represented by inheritance.



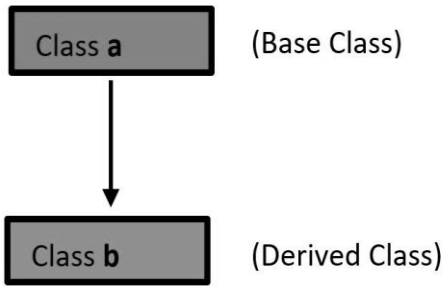
Inheritance has two types of Classes;

- Subclass or Derived Class: A Subclass or Derived Class is a class that inherits properties from another class.

- Base Class or Superclass: A Base Class or Superclass is a class whose properties are inherited by a subclass.

Inheritance can be divided in to five parts according to way that inherit the sub classes from parent class

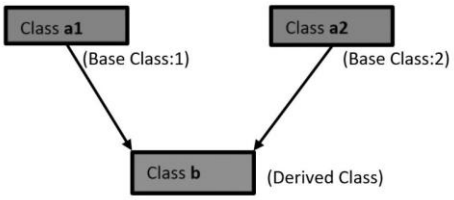
1) Single inheritance : A class can only inherit from one other class in single inheritance. Only one base class inherits one subclass.



Syntax:

```
class subclass_name : access_mode base_class
{
  // body of subclass
};
```

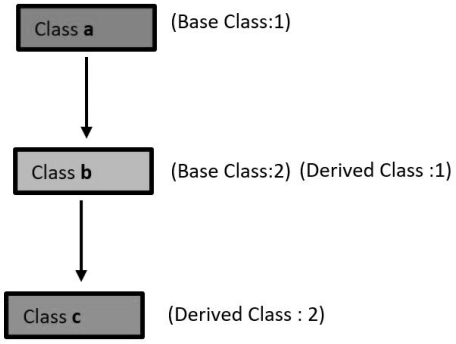
2) Multiple inheritance: Multiple Inheritance is a C++ feature that allows a class to inherit from multiple classes. More than one base class is inherited by a single subclass.



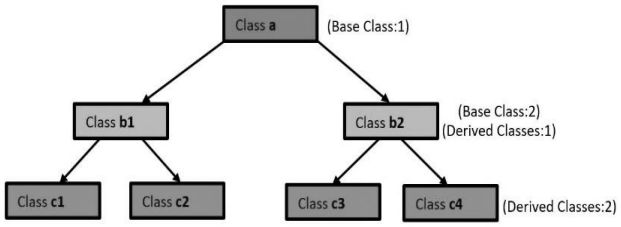
Syntax:

```
class subclass_name : access_mode base_class1, access_mode
base_class2, ....
{
  // body of subclass
};
```

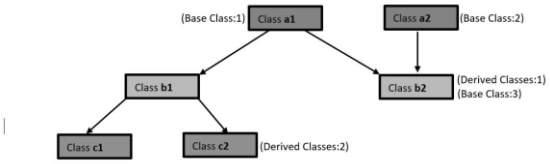
3) Multi-level inheritance: A derived class is produced from another derived class in this type of inheritance.



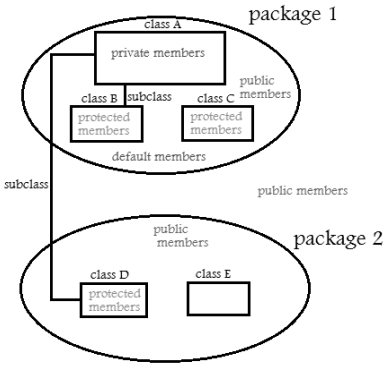
4)Hierarchical Inheritance: More than one subclass is inherited from a single base class in this style of inheritance. From a single base class, several derived classes are produced.



5) Hybrid Inheritance: Hybrid inheritance is created by merging multiple inheritance types. Combining Hierarchical Inheritance and Multiple Inheritance, for example.



Access Modifiers which use in Inheritance
Access modifiers specify which classes are allowed to use a particular attribute or method. There are 3 mode of access modifiers which using in Inheritance.



Public Method:
If a subclass is derived from a public base class. The base class's public members will become public in the derived class, and the base class's protected members will become protected in the derived class.

Protected Method:
If a subclass is derived from a Protected base class. The base class's public and protected members will then become protected in the derived class.

Private Method:
If a subclass is derived from a Private base class. The base class's public and protected members will then become Private in the derived class.

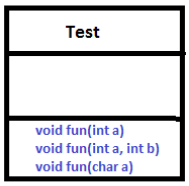
Default Method:
Package-private is the default access modifier, which means that all members are viewable within the same package but not available from other packages.

<u>Polymorphism in OOP</u>

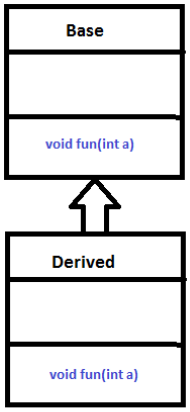
polymorphism is a single object-oriented concept. This describes how something happens in several different ways. There is a test to see if an object is polymorphism. If the object is multiple is a or instance of tests successfully passed, it is a polymorphism object. java When it comes to polymorphism, every java class is extended by a class object. For this reason, every java object passes more than two Instance Of checks. Therefore, all objects in Java are polymorphism.

There are two types of polymorphism

- static polymorphism (overloading)
- dynamic polymorphism (overriding)



Overloading



Overriding

Static polymorphism

Many programming languages allow you to implement several methods with the same name in the same class. But have to use different parameters. This is a static polymorphism.

There are a few types that need to change these parameters.

- They should have different sizes of parameters. That is, if one method has a parameter, the other method must have more than 1 parameter. That is, two or three.
- The type of parameter must be different. That is, if the type of one method is a string, then the parameter of the other method must be of a different type.
- The order of the parameters can be changed. That is, if the parameters of the first method are taken as string and long, then the second method can be taken as long and string.
- When using methods with different parameters, the compiler decides which method to call. This is a static polymorphism.

Dynamic Polymorphism

This type of polymorphism is not activated by the compiler. These are activated by the JVM (Java Virtual Machine) during run time.

This polymorphism is the inheritance of a class by a super class so that the methods of that super class can override the child class and customize the behavior of those methods or replace the behavior completely.

Here both these methods come from the super class. Also, the method name and the parameter are the same. But they offer different functionalities.

II. NEW FACTORS

a) Encapsulation and how to affect complexity

Encapsulation restricts the free flow of data from one object to another. The data and functions in an object are wrapped together in such a way that the data of an object can only be used in the associated functions. Encapsulation helps protect data from unauthorized access. The encapsulation hides some parts of the system, so they cannot be connected, so less complexity. Encapsulation is a method of reducing interdependencies between separate written modules through the definition of strict foreign interfaces. Getting more classes does not mean being complicated, because the Number of the classes and the methods depend on the complexity of the class architecture. It is difficult to understand a more complex form with more qualities and methods. As a result, increasing the code can reduce the complexity of the code [18].

Complexity introduced by encapsulation.

The encapsulated classes can be identified by the getter and setter methods. Also, by appealing to the public methods of the encapsulate class. If the getters and setters are used in the class, the weight value is -1 (E=-1). We considered a negative value for this parameter because it reduces the complexity of program. If not setters and getters methods in the program, weight value is +1 (E=+1). We considered a positive value for this parameter because it increases the complexity of program. If the private attributes are used in the class, the weight value is -2. It reduces the complexity of the program. If not, private attributes are used in the class, weight value is set to 2.it increase Complexity of the program [19].

- E-Encapsulation Factors
  - If a class has included set method or get method E = -1
  - Else E = 1
  - If the code has private attributes EA = -2
  - Else EA = 2

b) Abstraction and how to affect complexity

Those most important details are shown to the user as a result of data abstraction. The user is not shown the trivial or non-essential units. For example, a car is seen as a whole rather than its distinct parts. Data abstraction may also be described as the process of identifying just the necessary

properties of an item while discarding the unnecessary aspects. The features and behaviors of an item assist to distinguish it from other objects of the same type, as well as classify and organize the objects. Abstraction aids in the creation of well-designed code. Abstraction aids in the development of loosely connected programming. The reason for this is that abstraction behaves like a type, encapsulating the future changes hidden behind the abstract class/interface. Data abstraction aids in the transformation of a complex data structure into one that is rapid and simple to utilize. As a result, a program with a high level of code complexity can be turned into a program that appears to be written in human-readable language.[20]
There isn't a body in abstract techniques. If a class has an abstract form, it is regarded abstract; however, the reverse is not true, which implies that an abstract class does not need to have an abstract form. If a regular class extends an abstract class, the extended class must either follow the abstract parent class's abstract methods or be deemed abstract itself. By default, all interface methods are open to the public. In the interface, you can't have any concrete methods.[21]

o If a class is not an abstract class, the A value is considered as 0.
o If a class is an abstract class, the A value is considered as -1.
o Interfaces are pure abstract classes. Therefore, the A value for interfaces is also considered as -1

Polymorphism
Polymorphism and how to affect complexity.

Polymorphism is one of the main concepts of object-oriented. This allows the code to be reused, thus reducing the complexity of the code. Polymorphism is the ability of a programming language to display an object in various forms. The main advantage here is that it does a lot of work behind an interface. Therefore it is hidden from the client. So there is freedom and decoupling. polymorphism can be divided into two main parts. Namely dynamic and static. The easiest way to identify these dynamic polymorphisms is to include override methods. Sub inherited according to the inheritance hierarch can be used by overriding the existing methods in the super class. In this sub class it is possible to change or completely replace the structure of these methods.[22]

c) Static Polymorphism

Static polymorphism is a type of polymorphism that operates at the level of the class hierarchical tree. Static polymorphism based on methods overloading or function overloading. In here for Cognitive Code level Complexity measuring, we introduced new metric as Static Polymorphism Method Overloading. Method overloading is an Object-Oriented feature in which a class contains many methods with the same name but distinct arguments. It is accomplished by creating member functions with the same name but different signatures in separate classes. Method binding occurs at build time for both pure and static polymorphisms. Static Polymorphism increases the readability of the program and makes code look clean. Because of that Static polymorphism reduce the Cognitive code level complexity. Also, static polymorphism has ability of reuse code and save memory. This feature helps to minimizes the code length and build up the flexibility of the source code for the programmers.

According to newly proposed cognitive code complexity, the complexity
metric identifying structure for Static polymorphism as follows:

- If polymorphism has not occurred in a line of code of a program, then the value for ‘P’ is considered as = 1
- If polymorphism has occurred in a line of code of a program, then the value for ‘P’ is considered as = -1

d) Complexity introduced by polymorphism.

Polymorphism is mainly identified by override methods and overloading methods. A program containing override methods means that the complexity of the program is somewhat reduced of the ability to reuse the code. Let us now look at the new factor that this document points out. If a class contains override methods it is taken as PO = -1 and if it does not contain P = 1.

- P-polymorphism Factor
  - If a class has Override methods P=-1
  - Else P = 1

Proposed new metric equation

E - Encapsulation factor
P - Polymorphism overload factor
PO – Polymorphism override factor
A - Abstraction factor

$S_n=E+P+PO+A$

T_n= Size of the n^{th} executable line of code in terms of the token count
a = Total number of executable code lines of a program

We need to multiply the S value with the Zn value [S * T] to get the complexity measure value of a specific code statement(line). As a result, in order to determine the total complexity measure value of the entire program, we must add the values of each [S* T] code line.

$$WCC = \sum_{n=1}^{a} Sn * Tn$$

III. Conclusion

The metrics suite given here may be used to assess the Cognitive Code Level complexity of Object Oriented Factors based on Encapsulation, Static Polymorphism, Dynamic Polymorphism and Abstraction. This is significant because as projects get more complicated, the work required to sustain them tends to rise in tandem. We not only illustrated each metric, but we also used theoretical and empirical methods to validate them. We started by examining each measure based on Object Oriented characteristics for theoretical validation. Our measures are found to be satisfactory, as evidenced by the outcomes. It has been suggested that a measure does not have to meet all of these criteria in order to be considered a good measure [24].Then, using the paradigm provided by Briand et al. [25], we compared each to measurement theory. Also we used and analyzed nearly decades of past research to find the information which we need to do the research. From there, we noticed that each of our suggested metrics fit the source code qualities, putting all of the metrics in the suite on a ratio scale as is the case with any good measure. To give the metrics suite a practical viewpoint, we used some researchers methodologies to validate the usefulness of each of the measurements. As a result, the following capabilities are included in our metrics. At the class level, cognitive complexity metric for OO systems has been developed. Class cognition necessitates a thorough understanding of the class, which is accomplished through methods and characteristics. It is the primary rationale for developing a measure that may calculate the cognitive complexity of a class by taking into account the internal architecture of the methods as well as characteristics. The metric is assessed both theoretically and practically using measurement theory and a framework. The suggested measure is determined to be on a ratio scale and to satisfy the majority of the parameters required by a realistic evaluation system.

Calculation of Code Complexity

Executable Java Program 1

```
public abstract interface Pharmacy {

  //Abstraction
  abstract void giveMedicine();

  //abstraction
  abstract void calculateBill();

}


public class Doctor implements Pharmacy{

  public void discharge() {
    System.out.println("Discharge the patient");
  }

  public void testResult(String testName) {
    System.out.println(testName + " passed!");
  }

  //Static polymorphism - Method overloading
  public void checkPatient() {
    System.out.println("Check the patient");
  }

  //Static polymorphism - Method overloading
  public void checkPatient(String name) {
    System.out.println("Check the patient " + name);
  }

  //Static polymorphism - Method overloading
  public void checkPatient(String name, String
disease) {
    System.out.println("Check the patient " + name);
    System.out.println("Disease : " + disease);
  }

  //Dynamic polymorphism - Method overriding
  //Abstract Method call
  @Override
  public void giveMedicine() {
    System.out.println("Write the list of medicine");
  }

  //Dynamic polymorphism - Method overriding
  //Abstract method call
  @Override
  public void calculateBill() {
    System.out.println("Calculate Full bills");
  }

}


public class Patient extends Doctor implements
Pharmacy{

  //Encapsulation
  private String name;
  private int age;
  private String address;
  private String ward;
  private String guardian;
  private String telephone;

  //Default Constructor
  public Patient() {
    this.name = "";
    this.age = 0;
    this.address = "";
    this.ward = "";
    this.guardian = "";
    this.telephone = "";
  }

  //Overloaded Constructor
  public Patient(String name, int age, String address,
String ward, String guardian, String telephone) {
    this.name = name;
    this.age = age;
```

```
    this.address = address;
    this.ward = ward;
    this.guardian = guardian;
    this.telephone = telephone;
  }

  //Encapsulation
  public String getName() {
    return name;
  }
  public int getAge() {
    return age;
  }
  public String getAddress() {
    return address;
  }
  public String getWard() {
    return ward;
  }
  public String getGuardian() {
    return guardian;
  }
  public String getTelephone() {
    return telephone;
  }

  //Encapsulation
  public void setName(String name) {
    this.name = name;
  }
  public void setAge(int age) {
    this.age = age;
  }
  public void setAddress(String address) {
    this.address = address;
  }
  public void setWard(String ward) {
    this.ward = ward;
  }
  public void setGuardian(String guardian) {
    this.guardian = guardian;
  }
  public void setTelephone(String telephone) {
    this.telephone = telephone;
  }


  //Dynamic polymorphism - Method overriding
  @Override
  public void giveMedicine() {

    System.out.println("Give medicines by doctor");

  }

  //Dynamic polymorphism - Method overriding
  @Override
  public void calculateBill() {

    System.out.println("Calculate Full bills");

  }

  //Dynamic polymorphism - Method overriding
  @Override
  public void checkPatient() {
    // TODO Auto-generated method stub
    super.checkPatient();
  }

}
```

```
public class Main {

  public static void main(String[] args) {

    Patient patient1 = new Patient("Saman",32,"Hambantota","Priya","199","NO 5");
    Patient patient2 = new Patient("Nuwan",28,"Matara","Shashi","118","NO 8");

    Doctor doctor1 = new Patient();  //Polymorphism

    System.out.println("Patient Details");
    System.out.println("======================");
    System.out.println("Name                 :   "   + patient1.getName());  //Encapsulation
    System.out.println("Age                  :   "   + patient1.getAge());
    System.out.println("Address              :    "    + patient1.getAddress());
    System.out.println("Guardian          :     "     + patient1.getGuardian());
    System.out.println("Telephone      :      "      + patient1.getTelephone());
    System.out.println("Ward    No          :    "    + patient1.getWard());

    System.out.println("");
    System.out.println("Doctor Details");
    System.out.println("======================");
    doctor1.checkPatient("Saman",  "Fever");  //Static polymorphism - method overloading
    doctor1.testResult("Blood Test");
    doctor1.giveMedicine();      //override      method (Abstraction and polymorphism)
    doctor1.discharge();

  }

}
```

a

Program Statement	Tokens	Encapsulation (E)	Static Polymorphism (P)	Abstraction (A)	Dynamic Polymorphism (PO)	Size (S)	Total (T)	Sum (S * T)
public abstract interface Pharmacy {								
abstract void giveMedicine();	Abstract,void, giveMedicine()	1	1	-1	1	3	2	6
abstract void calculateBill();	Abstract, void, calculateBill()	1	1	-1	1	3	2	6
}								
public class Doctor implements Pharmacy{								
public void discharge() {	void, discharge()	1	1	0	1	2	3	6
System.out.println("Discharge the patient");	System,.,out,.,println(),"Discharge the patient"	1	1	0	1	6	3	18
}								
public void testResult(String testName) {	void, testResult()	1	1	0	1	2	3	6
System.out.println(testName + " passed!");	System,.,out,.,println(),testName, +, " passed!"	1	1	0	1	8	3	24
}								
public void checkPatient() {	void, checkPatient()	1	-1	0	1	2	1	2
System.out.println("Check the patient");	System,.,out,.,println(),"Check the patient"	1	-1	0	1	6	1	6
}								
public void checkPatient(String name) {	void, checkPatient()	1	-1	0	1	2	1	2
System.out.println("Check the patient " + name);	System,.,out,.,println(),"Check the patient ", +, name	1	-1	0	1	8	1	8
}								

public void checkPatient(String name, String disease) {	void , checkPatient()	1	-1	0	1	2	1	2
System.out.println("Check the patient " + name);	System,.,out,.,println(),"Check the patient " ,+, name	1	-1	0	1	8	1	8
System.out.println("Disease : " + disease);	System,.,out,.,println(),"Disease : " ,+, disease	1	-1	0	1	8	1	8
}								
@Override public void giveMedicine() {	void, giveMedicine()	1	1	-1	-1	2	0	0
System.out.println("Write the list of medicine");	System,.,out,.,println(),"Write the list of medicine"	1	1	-1	-1	6	0	0
}								
@Override public void calculateBill() {	void, calculateBill()	1	1	-1	-1	2	0	0
System.out.println("Calculate Full bills");	System,.,out,.,println(),"Calculate Full bills"	1	1	-1	-1	6	0	0
}								
}								
public class Patient extends Doctor implements Pharmacy{								
private String name;	String ,name	-2	1	0	1	2	0	0
private int age;	int,age	-2	1	0	1	2	0	0
private String address;	String, address	-2	1	0	1	2	0	0
private String ward;	String, ward	-2	1	0	1	2	0	0
private String guardian;	String ,guardian	-2	1	0	1	2	0	0
private String telephone;	String, telephone	-2	1	0	1	2	0	0
public Patient() {	Patient()	1	1	0	1	1	3	3
this.name = "";	this,.,name, =, ""	-2	1	0	1	5	0	0
this.age = 0;	this,.,age, =, 0	-2	1	0	1	5	0	0
this.address = "";	this,.,address,=, ""	-2	1	0	1	5	0	0
this.ward = "";	this,.,ward, = ,""	-2	1	0	1	5	0	0

this.guardian = "";	this,.,guardian, =,""	-2	1	0	1	5	0	0
this.telephone ="";	this,.,telephone, = ,"";	-2	1	0	1	5	0	0
}								
public Patient(String name, int age, String address, String ward, String guardian, String telephone) {	Patient()	1	1	0	1	1	3	3
this.name = name;	this,.,name ,=, name	-2	1	0	1	5	0	0
this.age = age;	this,.,age, = ,age	-2	1	0	1	5	0	0
this.address = address;	this,.,address, =, address	-2	1	0	1	5	0	0
this.ward = ward;	this,.,ward, = ,ward,	-2	1	0	1	5	0	0
this.guardian = guardian;	this,.,guardian ,= ,guardian	-2	1	0	1	5	0	0
this.telephone = telephone;	this,.,telephone, =, telephone;	-2	1	0	1	5	0	0
}								
public String getName() {	String, getName()	-1	1	0	1	2	1	2
return name;	Name	-2	1	0	1	1	0	0
}				0				
public int getAge() {	Int, getAge()	-1	1	0	1	2	1	2
return age;	Age	-2	1	0	1	1	0	0
}								
public String getAddress() {	String, getAddress()	-1	1	0	1	2	1	2
return address;	Address	-2	1	0	1	1	0	0
}								
public String getWard() {	String, getWard()	-1	1	0	1	2	1	2
return ward;	Ward	-2	1	0	1	1	0	0
}								
public String getGuardian() {	String, getGuardian()	-1	1	0	1	2	1	2
return guardian;	Guardian	-2	1	0	1	1	0	0
}								

public String getTelephone() {	String, getTelephone()	-1	1	0	1	2	1	2
return telephone;	Telephone	-2	1	0	1	1	0	0
}								
public void setName(String name) {	void ,setName()	-1	1	0	1	2	1	2
this.name = name;	this,.,name, =, name	-2	1	0	1	5	0	0
}								
public void setAge(int age) {	void, setAge()	-1	1	0	1	2	1	2
this.age = age;	this,.,age, = ,age	-2	1	0	1	5	0	0
}								
public void setAddress(String address) {	void ,setAddress()	-1	1	0	1	2	1	2
this.address = address;	this,.,address, = ,address	-2	1	0	1	5	0	0
}								
public void setWard(String ward) {	void, setWard()	-1	1	0	1	2	1	2
this.ward = ward;	this,.,ward, = ,ward	-2	1	0	1	5	0	0
}								
public void setGuardian(String guardian) {	void, setGuardian()	-1	1	0	1	2	1	2
this.guardian = guardian;	this,.,guardian, =, guardian	-2	1	0	1	5	0	0
}								
public void setTelephone(String telephone) {	void, setTelephone()	-1	1	0	1	2	1	2
this.telephone = telephone;	this,.,telephone, = ,telephone	-2	1	0	1	5	0	0
}								
@Override public void giveMedicine() {	void, giveMedicine()	1	1	-1	-1	2	0	0
System.out.println("Give medicines by doctor");	System,.,out,.,println(),"Give medicines by doctor"	1	1	-1	-1	6	0	0
}								

@Override public void calculateBill() {	void, calculateBill()	1	1	-1	-1	2	0	0
System.out.println("Calculate Full bills");	System,.,out,.,println(),"Calculate Full bills"	1	1	-1	-1	6	0	0
}								
@Override public void checkPatient() {	void, checkPatient()	1	-1	0	-1	2	-1	-2
super.checkPatient();	super,.,checkPatient()	1	-1	0	-1	3	-1	-3
}								
}								
public class Main {								
public static void main(String[] args) {	void, main()	1	1	0	1	2	3	6
Patient patient1 = new Patient("Saman",32,"Hambantota","Priya","199","NO 5");	Patient, patient1, =, new, Patient()	1	1	0	1	5	3	15
Patient patient2 = new Patient("Nuwan",28,"Matara","Shashi","118","NO 8");	Patient,patient2 ,=,new ,Patient()	1	1	0	1	5	3	15
Doctor doctor1 = new Patient();	Doctor, doctor1, =, new, Patient()	1	1	0	1	5	3	15
System.out.println("Patient Details");	System,.,out,.,println(),"Patient Details"	1	1	0	1	6	3	18
System.out.println("==================");	System,.,out,.,println(),"================"	1	1	0	1	6	3	18
System.out.println("Name : " + patient1.getName());	System,.,out,.,println(),"Name : ",+, patient1,.,getName()	-1	1	0	1	10	1	10
System.out.println("Age : " + patient1.getAge());	System,.,out,.,println(),"Age : ",+, patient1,.,getAge()	-1	1	0	1	10	1	10

System.out.println("Address : " + patient1.getAddress());	System,.,out,.,println(),"Address : ",+, patient1,.,getAddress()	-1	1	0	1	10	1	10
System.out.println("Guardian : " + patient1.getGuardian());	System,.,out,.,println(),"Guardian : ",+ ,patient1,.,getGuardian()	-1	1	0	1	10	1	10
System.out.println("Telephone : " + patient1.getTelephone());	System,.,out,.,println(),"Telephone : ",+, patient1,.,getTelephone()	-1	1	0	1	10	1	10
System.out.println("Ward No : " + patient1.getWard());	System,.,out,.,println(),"Ward No : ",+, patient1,.,getWard()	-1	1	0	1	10	1	10
System.out.println("");	System,.,out,.,println(),""	1	1	0	1	6	3	18
System.out.println("Doctor Details");	System,.,out,.,println(),"Doctor Details"	1	1	0	1	6	3	18
System.out.println("==================");	System,.,out,.,println(),"================"	1	1	0	1	6	3	18
doctor1.checkPatient("Saman", "Fever");	doctor1,.,checkPatient(),"Saman", "Fever"	1	-1	0	1	5	1	5
doctor1.testResult("Blood Test");	doctor1,.,testResult(),"Blood Test"	1	1	0	1	4	3	12
doctor1.giveMedicine();	doctor1,.,giveMedicine()	1	1	-1	-1	3	0	0
doctor1.discharge();	doctor1,.,discharge()	1	1	0	1	3	3	9
}								
}								
WCC								**354**

References

[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] M. Lorenz and J. Kidd, Object-Oriented Software Metrics: A Practical Guide. Upper Saddle River, NJ, USA: Prentice-Hall, 1994.

[4] J. Shao and Y. Wang, ‘‘A new measure of software complexity based on cognitive weights,’’ Electr. Comput. Eng., Can. J., vol. 28, no. 2, pp. 69–74, Apr. 2003.

[5] S. Misra and A. Adewumi, ‘‘Object-oriented cognitive complexity measures: An analysis,’’ in Proc. Handbook Res. Innov. Syst. Softw. Eng., 2014, pp. 150–169.

[6] 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.

[7] 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.

[8] T. J. McCabe and A. H. Watson, ‘‘Software complexity,’’ Crosstalk, vol. 7, no. 12, pp. 5–9, 1994.

[9] R. Reißing, “Towards a model for object-oriented design measurement”, In 5th International ECOOP workshop on quantitative approaches in object-oriented software engineering, 2001, pp. 71-84.

[10]S.R. Chidamber and C.F. Kemerer, “A metrics suite for object oriented design”, IEEE Transactions on Software Engineering, vol. 20, no. 6, 1994, pp. 476-493.

[11] R. Harrison, S.J. Counsell and R.V. Nithi, “An evaluation of the MOOD set of object-oriented software metrics”, IEEE Transactions on Software Engineering, vol. 24, no. 6, 1998, pp. 491-496.

[12] M. Lorenz, J. Kidd, Object-oriented software metrics, Englewood Cliffs, New Jersey: Prentice Hall, 1994.

[13] V.R. Basili, L.C. Briand, and W.L. Melo, “A validation of objectoriented design metrics as quality indicators”, IEEE Transactions on Software Engineering, vol. 22, no. 10, 1996, pp. 751-761.

[14] S. Purao, and V. Vaishnavi, “Product metrics for object-oriented systems”. ACM Computing Surveys (CSUR), vol. 35, no. 2, pp. 191-221.

[15] V.K. Vaishnavi, S. Purao, and J. Liegle, “Object-oriented product metrics: A generic framework”, Information Sciences, vol. 177 no. 2, 2007, pp. 587-606.

[16] S. Misra and I. Akman, “Weighted class complexity: A measure of complexity for object –oriented system” Journal of Information Science and Engineering, vol. 24, 2008, pp. 1689-1708.

[17] S. Misra, I. Akman and M. Koyuncu, “An inheritance complexity metric for object-oriented code: A cognitive approach”, Sadhana, vol. 36, no. 3, 2007, pp. 317-337.

[18] Raees Ahmad Khan, A. Yadav, “Development of Encapsulated Class Complexity Metric,” in Procedia Technology 4 (December 2012), 754 – 760

[19] Francis Thamburaj , A. Aloysius,”On Validating Cognitive Weighted Attribute Hiding Factor Complexity Metric ”,in I J C T A, 9(27), 2016, pp. 575-58

[20] FreeCodeCamp, 5 September 2018 [Online]. Available: https://www.freecodecamp.org/news/make-your-code-understandable-by-using-abstraction 4b522307130c/

[21] BeginnersBook, [Online]. Available: https://beginnersbook.com/2014/01/abstract-method with-examples-in-java

[22] 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

[23] Kushwaha & Misra, 2006; Misra & Akman, 2008; Arockiam et al., 2009; Misra et al., 2011; Misra & Cafer, 2011; Arockiam & Aloysius, 2011; Misra et al., 2012; Aloysius & Arockiam, 2012

[24]D. Baski and S. Misra, ‘‘Metrics suite for maintainability of eXtensible Markup Language Web services,’’ IET Softw., vol. 5, no. 3, pp. 320–341, Jun. 2011.

[25] L. C. Briand, S. Morasca, and V. R. Basily, ‘‘Property based software engineering measurement,’’ IEEE Trans. Softw. Eng., vol. 22, no. 1, pp. 68– 86, Jan. 199