ISSN No.: 2454- 2024 (online)

International Journal of Technical Research & Science

CALCULATING COHESION AND COPLING METRICES FOR OBJECT ORIENTED SYSTEM

Pooja Kumawat

E-Mail Id:poojakumawat.mit@gmail.com

Department of Computer Science and Engineering, SDBCE, Indore, Madhya Pradesh (India)

Abstract-In software engineering modularization is way to divide software project in to multiple independent and discrete modules. After complete module conquers make as software. Means modules design use divide and conquer rule. Cohesion and Coupling measure the quality of design of modules and interaction between modules. In this paper studies have been done to identify complexity between inheritance and interface by applying the cohesion and coupling metrics.Two program of c# implementing one with inheritance and other with interface are taken and measurement is done. The metrics value obtained is compared to prove which concept is good and beneficial for c# developer to use.

Keywords- cohesion, coupling, modularization, inheritance, interface

1. INTRODUCTION

When a software system program is modularized, its tasks area unit divided into many modules supported some characteristics. As we know, modules area unit set of directions place along so as to attain some tasks. They are although, thought-about as single entity however could confer with one another to figure along. There area unit measures by that the standard of a style of modules and their interaction among them is often measured. These measures area unit known as coupling and cohesion.

Cohesion is a very important attribute similar to the standard of the abstraction captured by the category into account. Sensible abstractions generally exhibit high cohesion. Cohesion refers to the degree of the relationships among the members during a category. A category is cohesive once its members area unit extremely related .A extremely cohesive module is one whose components have a detailed relationship among them so as to supply the only real practicality of the module. On the contrary, a coffee cohesive module has some components that have very little relation with others, that indicates that the module appears to supply many unrelated functionalities.

It is extensively accepted that the upper the cohesion of a module is, the simpler the module is to develop, maintain, and reuse, and therefore the less fault prone it's. it's a very important object-oriented software system quality attribute. The degree of sophistication cohesion offers a sign for the standard of sophistication style. In object-oriented paradigm, the category cohesion are often thought because the mensuration of connection among the members of sophistication. Metrics area unit suggests that for attaining a lot of correct estimations of project milestones, and developing a computer code that contains stripped-down faults. it's wide received that object bound development needs a unique manner of thinking than ancient structured development and software system comes area unit shifting to object bound code. Object bound metrics to measure properties of object bound Code.

1.1Cohesion

Cohesion may be a live that defines the degree of intra-dependability inside components of a module. The bigger the cohesion, the higher is that the program style.in below figure 1 describe how to determine cohesion module.

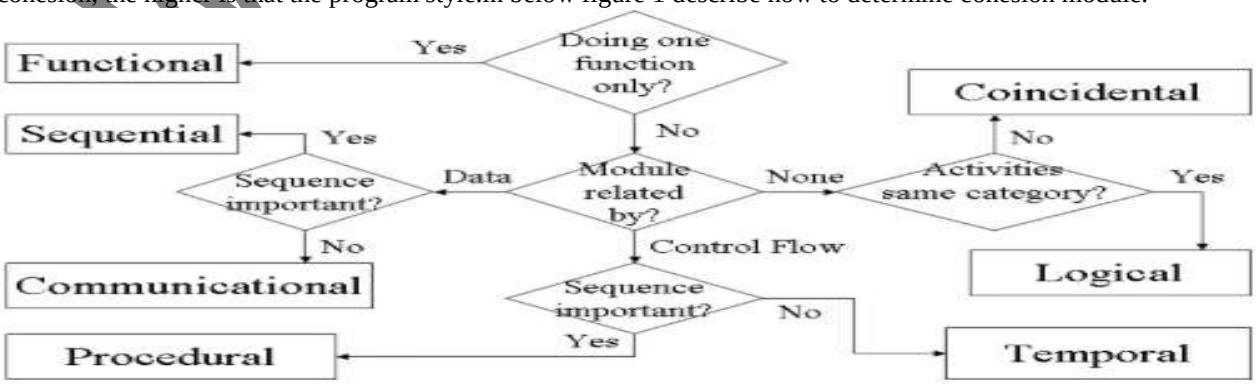

Fig. 1.1 Determine Cohesion Modules

1.1.1 Coincidental Cohesion

Its unplanned and random cohesion, which could be the results of breaking the program into smaller modules for the sake of modularization. as a result of it's unplanned, it's going to serve confusion to the programmers and is usually not accepted.

1.1.2 Logical Cohesion

Once logically classified components area unit place along into a module, it's known as logical cohesion.

1.1.3 Temporal Cohesion

Once components of module area unit organized such they're processed at the same purpose in time, it's known as temporal cohesion.

1.1.4 Procedural Cohesion

Once components of module area unit classified along, that area unit dead consecutive so as to perform a task, it's known as procedural cohesion.

1.1.5 Communicative Cohesion

Once components of module area unit classified along, that area unit dead consecutive and work on same information (information), it's known as communicative cohesion.

1.1.6 Successive Cohesion

Once components of module area unit classified as a result of the output of one part is input to a different and then on, it's known as successive cohesion.

1.1.7 Practical Cohesion

It's thought-about to be the best degree of cohesion, and it's extremely expected. Components of module in practical cohesion area unit classified as a result of all of them contributes to one well defined perform. It can even be reused.

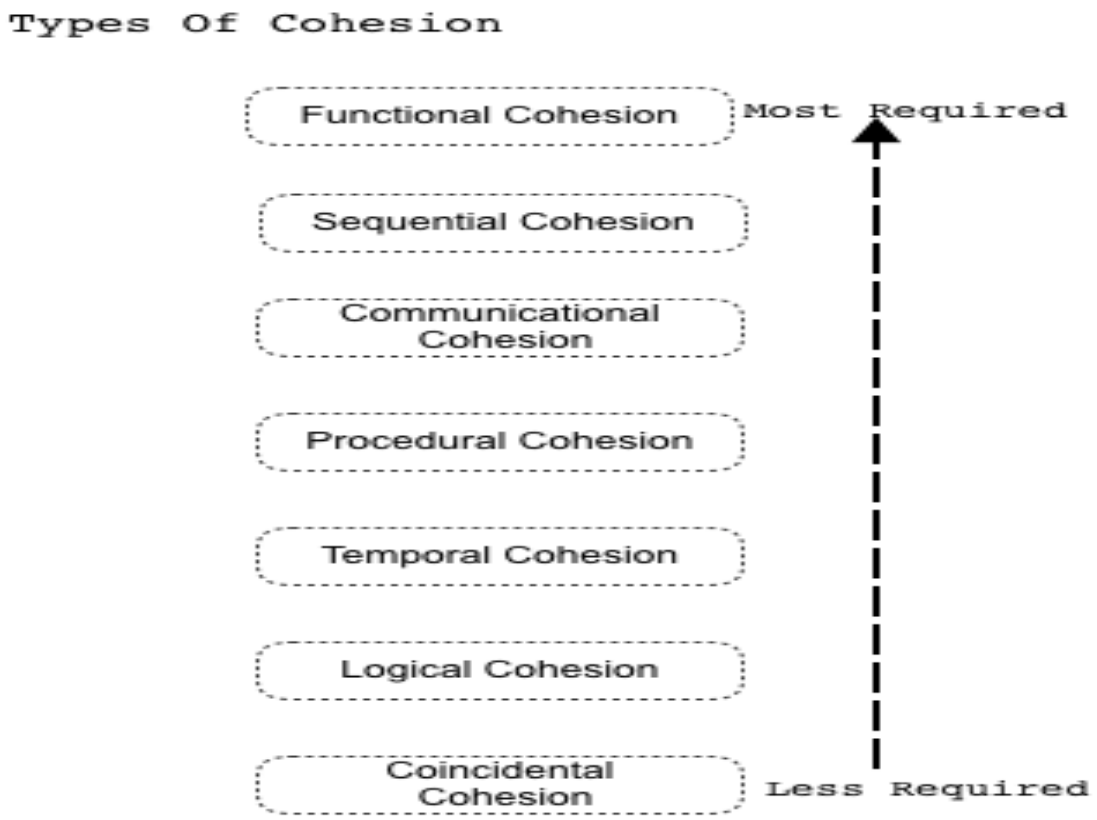

Fig. 1.2 Type of Cohesion and its Importance

1.2 Coupling

Coupling may be a live that defines the amount of inter-dependability among modules of a program. It tells at what level the modules interfere and act with one another. The lower the coupling, the higher the program.

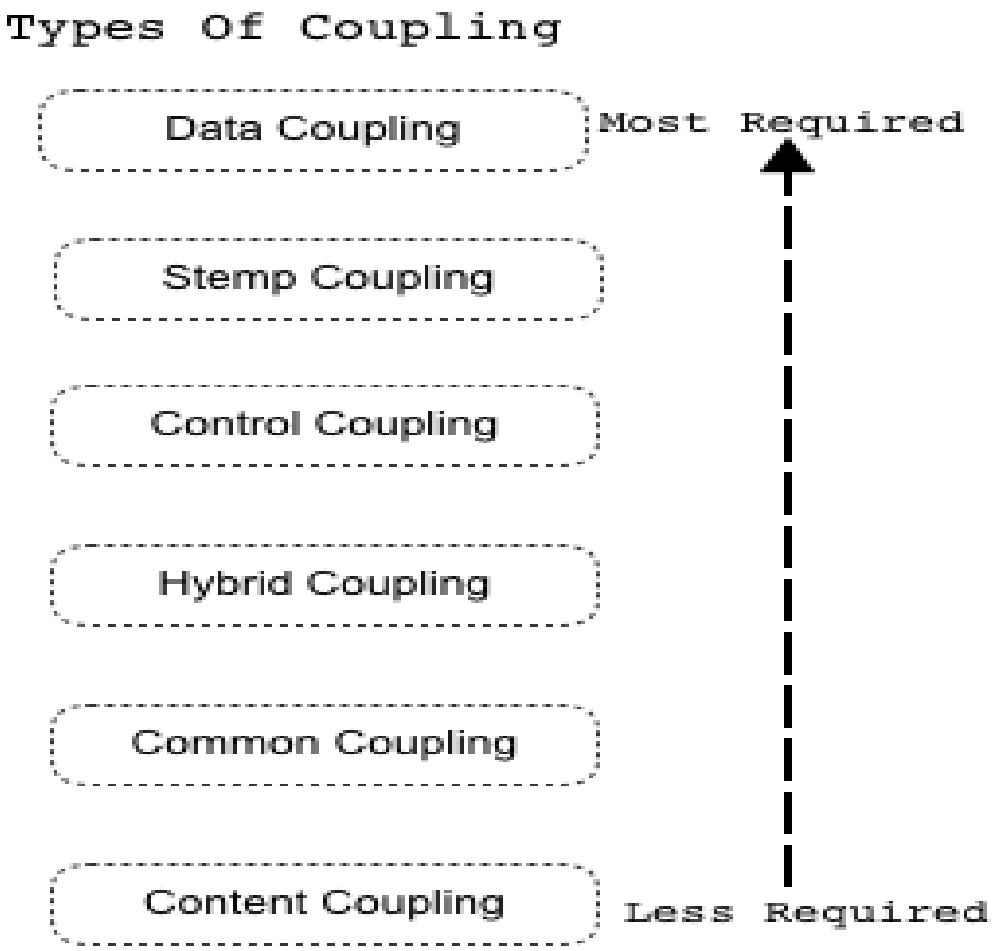

Fig. 1.3 Types of Coupling and its Importance

1.2.1 Content Coupling

Once a module will directly access or modify or confer with the content of another module, it's known as content level coupling.

1.2.2 Common Coupling

Once multiple modules have scan and write access to some international information, it's known as common or international coupling.

1.2.3 Control Coupling

Two modules area unit known as control-coupled if one among them decides the perform of the opposite module or changes its flow of execution.

1.2.4 Stamp Coupling

Once multiple modules share common system and work on completely different a part of it, it's known as stamp coupling.

1.2.5 Information Coupling

Data coupling is once two modules act with one another by suggests that of passing information (as parameter). If a module passes system as parameter, then the receiving module ought to use all its elements.

2. LITERATURE REVIEW

The concept of an interface in object-oriented programming is quite old. Software engineering has been using interfaces for more than 25 years. Many metrics are available to measure class, method, inheritance, polymorphism and system level. There is no significant work on the code of human computer interfaces. In literature, relatively little information has been published on interface metrics. Those metrics provide only little information about the quality and usability of the interfaces. Sue of interface leads to the high cohesion and make the code more reusable.
In Year -2010, V. Krishnapriya and Dr. K. Ramarhave measured interface concept by using coupling metrics on design based and have proved interface is more effective in use then inheritance to increase reusability of a code in object oriented programming [1]. In this paper, measurement of inheritance and interface is calculated using cohesion metrics using a example and prove the usage of interface increased the reusability James m. Bieman and byungkyookang published a Paper on Cohesion and reuse in object oriented system explaining the TCC and LCC on C++ Program [4].

2.1 What is Cohesion Metric and how will it be Measured

There are many metrics to find class cohesion but no standard metric or definition has been generally accepted, out of available [Fenton &Pfleeger 1998], [Counsell et al. 2002] and [Etzkorn et al. 2004]. A reasonable metric to measure class cohesion should give an insight to the relatedness among the methods of a class while considering the impacts of inheritance paradigm on local class cohesion.

2.2 What is the Coupling Metric and how will it be Measured

Like class cohesion, there is no standard metric or definition for class coupling [Fenton &Pfleeger 1998]. However, In OO design class coupling is a measurement of class dependence on other classes.We will attempt to measure a class coupling on the basis of UML relationships.

2.3 What are the possible relationships, which may Exist between Cohesion and Coupling Metric

Metrics for measuring class cohesion and class coupling are supposed to share same input data for their respective measurements. By the same set of input data we mean class member attributes, member methods, and usage of attributes by the methods. We will attempt to find mutual relationships between class cohesion and class coupling metrics by analyzing the results of experiment statistically.

2.4 How to Calculate Cohesion and Coupling Matrices During Software Evolution in Object Oriented

Literature on the subject of the software evolution clearly introduces the erosive trends in the software architecture while meeting the changes imposed by the software evolution. In this thesis, we will attempt to identify such erosive trends with the help of class cohesion and coupling metrics. Based on the literature review, we suppose that both class cohesion and coupling should follow deteriorating trends while evolution in the software architecture.

3. COHESION AND COUPLING METRICES IN OBJECT ORIENTED SYSTEM

Cohesion is the degree to which methods within a class are related to one another and work together to provide well-bounded behavior. Effective object oriented designs maximize cohesion because cohesion promotes encapsulation. Coupling is a measure of the strength of association established by a connection from one entity to another. Classes are coupled when a message is passed between objects; when methods declared in one class use methods or attributes of another class. Inheritance is the hierarchical relationship among classes that enables programmers to reuse previously defined objects including variables and operators.

3.1 Measure Cohesion for Class Using Inheritances and Interfaces

Inheritance is one of the fundamental concepts of Object Orientated programming, in which a class ”gains” all of the attributes and operations of the class it inherits from, and can override some of them, as well as add more attributes and operations of its own. In Object Oriented Programming, inheritance is a way to compartmentalize and reuse code by creating collections of attributes, thing and behaviors called objects that can be based on previously created objects.
Lack of Cohesion in Methods (LCOM) as the number of pairs of methods operating on disjoint sets of instance variables, reduced by the number of method pairs acting on at least one shared instance variable.

Implementation Using Inheritance:

```
using System;
classShape{
publicvoid Draw();
publicvoid Element();
}
classRegularPolygon:Shape
{
publicint Linesegment;
publicvoid Perimeter();
}
classEllipse: Shape{
publicint  curved;
publicint  Surface;
}
classTriangle : RegularPolygon
{
publicint sumofangles = 180;
publicvoid setsides();
publicvoid Area();
}
classRectangle : RegularPolygon
{
publicint sumofangles = 360;
publicvoid setsides();
publicvoid Area();
}
class Circle : Ellipse
{
   public int symmetrical;
   publicvoid Circumference();
}
classSalene : Triangle
{
publicint Nosidesequal;
}
classIsosceles : Triangle
{
publicint sideequal2;
publicint Anglesequal2;
}
classEquilateral : Triangle{
publicint sidesequal3;
publicint Anglesequal3;
}
classSquare : Rectangle
{
publicint oppositesidequal;
publicint angles4;
}
```

Implementation Using Interface:

```
using System;
interfaceShape
{
publicvoid Draw_Element();
}
interfaceRegularPolygon{
publicvoid Linesegment();
publicvoid Perimeter();
}
interfaceEllipse
{
publicvoid Circumference();
}
classTriangle : Shape
{
int Sumofangles = 180;
publicvoid Draw_Element();
publicvoid setsides();
publicvoid Area();
}
classRectangle : RegularPolygon
{
int sumofangles = 360;
publicvoid Perimeter();
publicvoid Linessegment();
publicvoid setsides();
publicvoid Area();
}
classCircle : Ellipse
{
int symmetricalpictur;
publicvoid Circumference();
}
classScalene : Triangle
{
int notequalsides;
}
classIsosceles : Triangle
{
int sidesequal;
int angleequal;
}
classEquilateral : Triangle{
int sidesequal;
int angleequal;
}
classsquare : Rectangle{
int opposite;
int sidesequal;
int anglesequal;
```

Fig. 3.1 Implementation Using Inheritance **Fig. 3.2Implementation Using Interface**

Class P is represented using C(P) and public method and attributes ate represented using M(P) and A(P).Any class is show that

$$C(A)=M(A)U\ A(P);$$

We are measuring the cohesion in the given program therefore considering the classes in short name

- Shape S

- Regular Polygon Rp
- Ellipse E
- Triangle T
- Rectangle R
- Circle C
- Selene Se
- Isosceles Is
- Equilateral Eq
- Square Sq

LCOM1 =|P|-|Q| …….(II)
Where P=No of pairs in Disjoint Set
Q=No of Pairs in Joint Set

Table-3.1 Calculation of Classes in Disjoint Sets (Using Inheritance)

<S,T>,<S,R>,<S,C>,<S,Se>,<S,Is>,<S,Eq>,<S,Sq>	7
<Rp,E>,<Rp,C>,<Rp,Se>,<Rp,Is>,<Rp,Eq>,<Rp,Sq>	6
<E,T>,<E,R>,<E,Se>,<E,Is>,<E,Eq>,<E,Sq>	6
<T,R>,<T,C>,<T,Sq>	3
<R,C>,<R,Se>,<R,Is>,<REq>	4
<C,Se>,<C,Is>,<C,Eq>,<CSq>	4
<Se,Is>,<Se,Eq>,<Se,Sq>	3
<Is,Eq>,<Is,Sq> ,<Eq,Sq>	3
Total no of pairs in P is=	36

Table-3.2 Calculation of Classes in Joint Sets (Using Inheritance)

<S,Rp>,<S,E>,<Rp,T>,<Rp,R>,<E,C>,<T,Se>,<T,Is>,<T,Eq>,<R,Sq>	9
Q=	9

LCOM1=|9|-|36|= -27

Formula of LCOM2 =| [4|p| - n(n-1)] /2 |$^{+}$ Where the
P=No of Pairs in Disjoint set
Q=No of classes are: n=10
Then LCOM2 is =| [4|36|-10*9]/2 | =
|54/2|= 27
LCOM2=27
Conventions : We are measuring the cohesion in the given program therefore considering the classes in short name

- Triangle T
- Rectangle R
- Circle C
- Scalene Sc
- Isosceles Is
- Equilateral Eq
- Square Sq

LCOM1 =|Q|-|P|
Where P=No of pairs in Disjoint Set
Q=No of Pairs in Joint Set

Table-3.3 Calculation of Classes in Disjoint Sets (Using Interface)

<T,R>,<T,C>,<T,Sq>,	3
<R,C>,<R,Sc>,<R,Is>,<R,Eq>	4
<C,Sc>,<C,Is>,<C,Eq>,<C,Sq>	4
<Sc,Is>,<Sc,Eq><Sc,Sq>	3
<Is,Eq>,<Is,Sq>	2
<Eq,Sq>	1
Total no of pairs in P is=	17

Table-3.4 Calculation of Classes in Disjoint Sets (Using Interface)

<T,Sc>,<T,I>,<T,Eq>,<R,Sq>	4
Q=	4

LCOM1=|4|-|17|= -13
Formula of LCOM2 =| [4|p| - n(n-1)] /2 |$^{+}$ Where the 'P': No of Pairs in Disjoint set
No of classes are : n=7

3.2measure Coupling for Class Using Inheritance and Interface

Coupling of a class means the measurement of the interdependency of class with other class's.A class as a parameter in one of its member methods.

CONCLUSION

The purpose of this thesis is to finding the approach and way to identify complexity between inheritance and interface programming through cohesion metrics in object oriented programs. Metrics measure certain properties of software system by mapping them to numbers (or to other symbols) according to well-defined, objective measurement rules. Code Metrics are measurements of the static state of the project's Code and also used for assessing the size and in some cases the quality and complexity of software. Analysis and maintenance of Object-Oriented (OO) software is expensive and difficult. Thus, measuring the relationships has become a prerequisite to develop efficient techniques for analysis and maintenance. Various cohesion metrics have been proposed and used in past empirical investigations; however none of these have taken the run-time properties of a program into account. As program behavior is a function of its operational environment as well as the complexity of the source code, static metrics may fail to quantify all the underlying dimensions of coupling and cohesion. In our future work, We can apply various other cohesion metrics to identify better complexity between inheritance and interface programming.

REFERENCES

[1] V. Krishnapriya, K. Ramar, "Exploring the Difference Between Object Oriented Class Inheritance and Interfaces Using Coupling Measures," ace, pp.207-211, 2010 International Conference on Advances in Computer Engineering, 2010.
[2] K.K. Aggarwal, Yogesh Singh, ArvinderKaur, Ruchika Malhotra. "Empirical Study of Object- Oriented Metrics",2006.
[3] Martin Hitz, Behzad Montazeri. "Measuring Coupling and Cohesion. In Object-Oriented Systems" in Angewandte Informatik (1995).
[4] James M. Bieman and Byung-kyookang."Cohesion and Reuse in Object Oriented System" Department of Computer Science, Colorado State University Fort Collins, Colorado,1995.
[5] Shyam R. Chidamberand Chris F. Kemerer" A Metrics Suite For object Oriented Design" IEEE Transactions on software Engineering, Vol. 20, No. 6, June 1994.
[6] Krishnaprasad Thirunarayan."Inheritance in Programming Languages" Department of Computer Science and Engineering, Wright State University, Dayton, OH-45435.