Complexity Identification of Inheritance and Interface based on Cohesion and Coupling Metrics to Increase Reusability

Maya Yadav
R.K.D.F, IST Bhopal

Jasvinder Pal Singh
R.K.D.F, IST Bhopal

Pradeep Baniya
MITM, Indore

ABSTRACT

Measurement is an essential component of software engineering the aim of this paper to identify and analyze complexity of object oriented programming. In this we have applied Cohesion and Coupling metrics on programs of inheritance and interface and evaluate the metrics values. The cohesion and Coupling metrics presented identifies complexity between inheritance and interface programming. In this paper we want to show which concept is good to use and beneficial for software developer. This paper focuseson an empirical evaluation of object oriented metrics in C#. The resulting valueshave been analyzed to provide significant insight about theobject oriented characteristics of reusability programs.

Keywords: Object Oriented Metrics, C#, Cohesion, Coupling, Inheritance, Interface.

1. INTRODUCTION

Effective reuse of software products is reportedly increasing productivity, saving time, and reducing cost of software development. Historically, software reuse focused on repackaging and reapplying of code modules, data structures or entire applications in the new software projects.

"Software quality is the degree to which software possesses a desired combination of attributes such as maintainability, testability, reusability, complexity, reliability, interoperabilityetc." – IEEE 1992.

Today everyone is interested to increase the productivity, and reducing the cost of developing products, and better quality of software providing. To improve software products and process, Measurements are essential. Software measurement plays an important role in finding the software quality, performance, maintenance and reliability of software products [9][10]. The concept of measurement requires appropriate measurement tools to measure, to collect, to verify and validate relevant metric data. Nowadays, many metric tools are available for software measurement [11][12].The main objective of this Paper work is to measure, analyze and propagate the difference between using object oriented class inheritance and interfaces in C# source code using Cohesion and coupling measures by metrics.

2. Related Work

Inheritance and interface are important concept in object oriented programming. Software engineering has been using interfaces for more than 25 years. Various metrics are existing to measure class, method, inheritance, polymorphism and system level [10]. There is no significant effort 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 as well as lowcoupling and make the code more reusable. Inheritance and interface concept measured 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][2]. In this paper, measurement of inheritance and interface is calculated using coupling metrics as well as Cohesion metrics using an example and prove the usage of interface increased the reusability

3. CLASS INERITANCE AND INTERFACE

Inheritance is one of the essential 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/modify 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 and behaviors called objects that can be based on previously created objects. In classical inheritance where objects are defined by classes, classes can inherit other classes. The new classes, known as subclasses (or derived classes), inherit attributes and behavior (i.e. previously coded algorithms) of the pre-existing classes, which are referred to as super classes, ancestor classes or base classes. The inheritance, relationships of classes gives rise to a hierarchy. In prototype-based programming, objects can be defined directly from other objects without the need to define any classes, in which case this feature is called differential inheritance. The inheritance concept was invented in 1967 for Simula [3].

Interfaces allow only method definitions and constant attributes. Methods defined in the interfaces cannot have implementations in the interface. Classes can .implement. The interface by providing bodies for the methods defined in the interface. An interface is a contract between a client class and a server class [3]. It helps to decouple the client from the server. Any intended change on the methods defined in the interface will impact both the client and server classes. Possible changes are as follows: 1) changing the name of a method, 2) changing the signature of a method, and 3) changing the return type of a method. There are two other possible changes that worth noting. If a new method is added to an interface, this will also impact the server and client classes that currently use or implement the interface. On the other hand if the implementation detail of a method inside a server class is changed, this change only affects the client class and not the interface. This specific case is more a code issue than a design issue and therefore it is not a concern in this evaluation. Interfaces have another very important role in the C# programming language. Interfaces are not part of the class hierarchy, although they work in combination with classes.The C# programming language does not permit

multiple inheritance (inheritance is discussed later in this lesson), but interfaces provide an alternative. In C#, a class can inherit from only one class but it can implement more than one interface.Therefore, objects can have multiple types: the type of their own class and the types of all the interfaces that they implement.

4. OBJECT ORIENDED METRICS SET

Several authors have introduced different approaches and proposed measures to coupling in object-oriented systems we are using following metrics in our research work.

Sr. No	Metrics	Object oriented Attribute	Description	Source
1	Coupling between object (CBO)	Coupling	CBO for a class is count of the number of other classes to which it is coupled. Two classes are coupled when methods declared in one class use methods or instance variables defined by the other class.	[Chidamber 94]
2	No of Association (NoACC)	Coupling	The number of Association per class metric is defined as the total number of associations a class has with other classes or with itself. This metric is used to measure complexity and coupling [4][5]. When the number of associations are less the coupling between objects arereduced. This metric was introduced by Brian.	[Brian 96]
3	No of Dependencies In(NDepIN)	Coupling	The Number of Dependencies IN metric is defined as the number of classes that depend on a given class. When the dependencies are reduced the class can function more independently. This metrics was introduced by Brian.	[Brian 96]
4	No of Dependencies out(NDepIN)	Coupling	The Number of Dependencies Out metric is defined asthe number of classes on which a given class depends.When the metric value is less the class can functionindependently. This metric was introduced by Brian.	[Brian96]
5	No Of Children (NOC)	Coupling	Number of children metric was introduced by CK [4]. NOC defines number of immediate sub-classes subordinated to a class in the class hierarchy. This metric measures how many sub-classes are going to inherit the methods of the parent class. NOC relates to the notion of scope of properties. If NOC grows it means reuse increases. On the other hand, as NOC increases, the amount of testing will also increase because more children in a class indicate more responsibility. So, NOC represents the effort required to test the class and reuse.[5]	[Chidamber 94]
6	Depth of Inheritance (DIT)	Coupling	Depth of a class with in the inheritance hierarchy is the maximum number of steps from the class node to the root of the tree or the length of the longest path from the class to the root of the hierarchy. This is measured by the number of ancestor classes[4]	[Chidamber 94]
7	Tightly class cohesion (TCC)	Cohesion	The Tight Class Cohesion metric measures the cohesion between the public methods of a class. That is the relative number of directly connected public methods in the class. Classes having a low cohesion indicate errors in the design [4]. The Tight Class Cohesion (TCC) measures the ratio between the actual number of visible directly connected methods in a class NDC(C)[4][6][7] divided by the number of maximal possible number of connections between the visible methods of a class NP(C).We define two measures of class cohesion based on a direct and indirect connection of a method pair. Let NP(C) be the total no. of pairs of abstracted methods in AC(C). NP is the maximum possible no. of direct or indirect connections in a class. If there are N methods in a class c, NP(C) is N*(N-1)/2. Let NDC(C) be the no. of direct connection and NIC(C) be the no. of indirect connection in AC(C).Tight class cohesion is a related no. of directly connected methods Following formula is used for calculating TCC value : TCC(C) = NDC(C)/NP(C)	[Briand 99]
8	Loosely class Cohesion (LCC)	Cohesion	Loose Class Cohesion is the relative no. of directly or indirectly connected method.[4][7] Following formula is used for calculating LCC value : LCC (C) = (NDC(C) + NIC(C)) / NP(C)	[Briand 99]

5. PROPOSED APPROACH

Goal: Exploring the difference between class inheritance and interface in C# programming through coupling and cohesion metrics.

Hypothesis: Six object oriented Coupling metrics and Two object oriented cohesion metrics are used for coupling as well as cohesion measures in object oriented class inheritance and interface programs respectively.

1. Two C# program is used with inheritance concept in this paper.
2. These programs are introduced with maximum possible interface.
3. All metrics are applied to both inheritance and interface programs.
4. The results are compared between inheritance and interface coupling and cohesion measures.

To validate the above metrics two object oriented inheritance programs have been taken and possible interfaces have been introduced in class inheritance programs. The first study considered is a vehicle classification class inheritance program which is represented in Figure 1

```
using system;
classvehicle
{
publicint wheelcount,scooty,cab;
publicvoid setdata()
 {
publicint motor;
 }
publicvoid getdata()
 {
publicint car,scooty;
 }
}
class lightmotor : vehicle
{
 pubic int speedlimit,capacity,fresh;
publicvoid setdata()
 {
publicint speedlimit,fan;
 }
publicvoid displaydata()
 {
  vehicle.scooty;
 }
}
class heavymotor : vehicle
{
publicint speedlimit,capacity,permitno;
publicvoid setdata()
 {
publicint engine,speedlimit;
  vehicle.cab;
 }
publicvoid displaydata()
 {
  vehicle.scooty;
 }
}
class gearmotor : lightmotor
{
publicint gearcount;
publicvoid setdata()
 {
publicint gearcount,cash;
 }
publicvoid displaydata()
 {
  lightmotor.fresh;
 }
}
class nongearmotor : lightmotor
{
publicvoid setydata()
 {
publicint engine;
  lightmotor.gearcount;
 }
publicvoid displaydata()
 {
  lightmotor.fresh;
 }
}
class passenger : heavymotor
{
publicint sitting;
publicvoid displaydata()
 {
publicint sitting;
  heavymotor.capacity;
 }
publicvoid displaydata()
 {
  heavymotor.permitno;
 }
}
class goods : heavymotor
{
publicvoid setdata()
 {
publicint delivery;
 heavymotor.speedlimit;
 }
publicvoid displaydata()
 {
  heavymotor.permitno;
 }
 }
```

Figure 1: Vehicle classification using Class Inheritance – adopted from [8]

The above said class inheritance Figure1 is introduced with possible number of interface and represented in Figure2.

```
using System;
Interface vehicle
{
publicvoid setdata();
publicvoid getdata();
}
Interface lightmotor
{
publicvoid displaydata1();
}
Interface heavymotor
{
publicvoid displaydata2();
}
class gearmotor : vehicle,lightmotor
{
publicint gearcount,scooty,fresh;
publicvoid setdata()
 {
```

```
publicint scooty,cash;
 }
publicvoid getdata()
 {
publicint fresh;
 }
publicvoid displaydata1()
 {
publicint gearcount,fresh;
 }
}
class nongearmotor : vehicle,lightmotor
{
publicint
engine,wheelcount,gearcount,cab,fresh;
publicvoid setdata()
 {
publicint wheelcount;
 }
publicvoid getdata()
 {
publicint cab;
 }
publicvoid displaydata1()
 {
publicint fresh,gearcount,engine;
 }
}
class passenger : vehicle,heavymotor
{
publicint
sitting,capacity,wheelcount,scooty,permit
no;
publicvoid setdata()
 {
publicint wheelcount,scooty;

 }
publicvoid getdata()
 {
publicint scooty,permitno;
 }
publicvoid displaydata2()
 {
publicint capacity,sitting,permitno;
 }
}
class goods : vehicle,heavymotor
{
publicint
speedlimit,wheelcount,delivery,permit
no;
publicvoid setdata()
 {
publicint delivery;
 }
publicvoid getdata()
 {
publicint wheelcount,permitno;
 }
publicvoid displaydata2()
 {
publicint speedlimit,permitno;
 }
}
```

Figure 2: Vehicle classification using Interface adopted from [8]

6. Result of Coupling Measurement

Six Coupling metrics discussed above are applied for above inheritance and interface programs. The results are show in Figure 3.

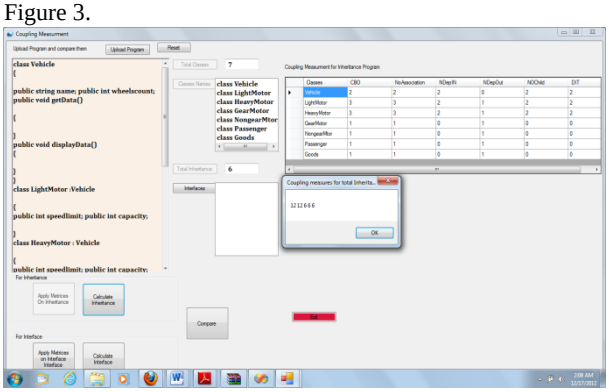


Figure 3 snapshot of coupling measurement for inheritance source code

Figure -3 Shows the upload C# Program of inheritance and Calculate the total classes, Name of Classes ,total inheritance in the program then apply the coupling metrics on the uploaded program and evaluate the data and calculate the coupling measurement.

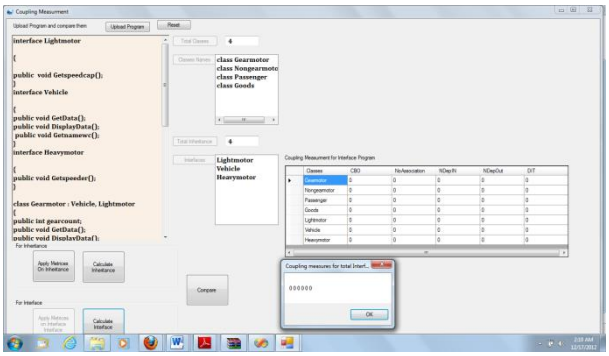


Figure 4 snapshot of coupling measurement for interface source code

Figure 4 Shows the upload C# Program of interface and Calculate the total classes, Name of Classes ,total inheritance and interface with name in the program then apply the coupling metrics on the uploaded program and evaluate the data and calculate the coupling measurement.

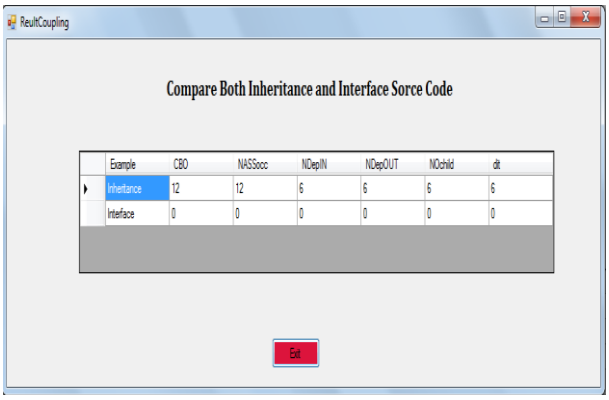


Figure 5 snapshot of comparision both inheritance and interface source code through the coupling metrics

For the above said two programs the coupling metrics are measured and compared in Figure 5. By comparing thefigure5 values for both the programs the interface values are reduced for almost all metrics.

7. Result of Cohesion Measurement

Similar process will repeat for cohesion measures, Two Cohesion metrics discussed above are applied for above both inheritance and interface programs. The results are show in Figure 6 and 7.

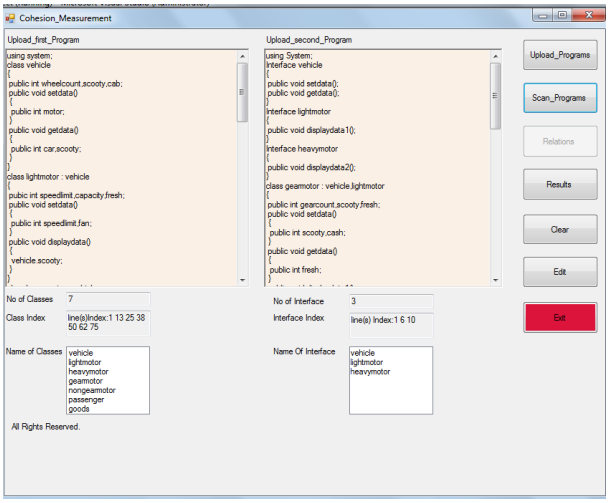


Figure 6 snapshot of calcuation of both programs

Figure – 6 Shows the upload C# Program of inheritance and interface simultaneously and Calculate the total classes, index of class Name of Classes ,total inheritance total interface in the program .

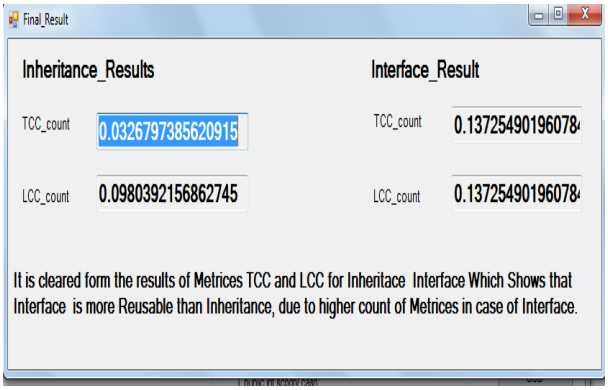


Figure 7 snapshot of comparision both inheritance and interface source code through the cohesion metrcis

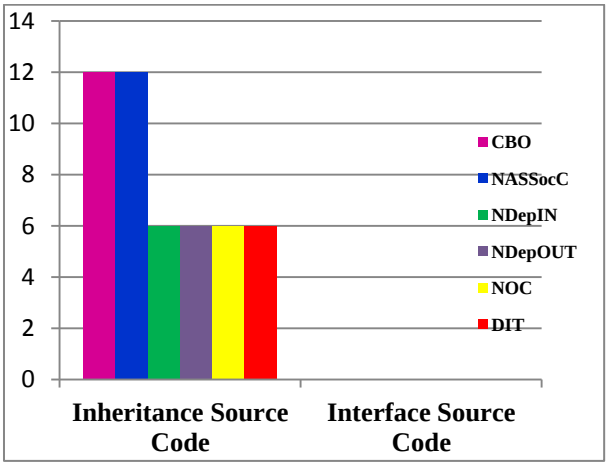
Figure 7shows Both TCC and LCC cohesion metrics apply the on the inheritance and interface programs and Calculate the total TCC count and LCC count for Inheritance and Interface. By comparing the values for both the programs the interface values are reduced for almost all metrics.

RESULT ANALYSIS FOR COHESIONMEASURMENT

We applied six metrics on source codes of inheritance and interface and calculated the data values of CBO, NASSocC, NDepIN, NDepOUT, NOC, and DIT using coupling metrics show above in figure – 5 Calculated data shown in Table -1

TABLE -1 Total metrics weighted values on both inputted code

Metric's/ source code	Calculated values for Inheritance source code	Calculated values for Interface source code
CBO	12	0
NASSocC	12	0
NDepIN	6	0
NDePOUT	6	0
NOC	6	0
DIT	6	0



Graph-1The comparison between inheritance and interface based on calculated data of coupling metrics

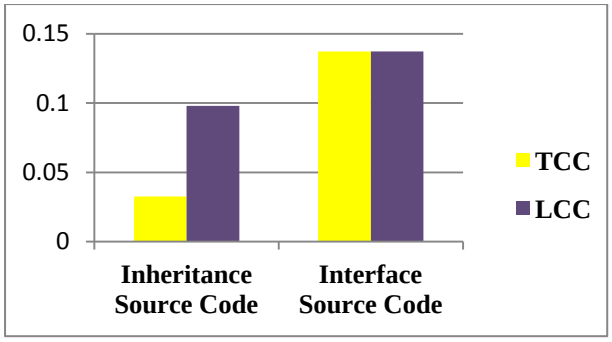
Here from the result of table-1 and graph-1, Value of All six metrics of interface is less than inheritance so we conclude that coupling will reduce in interface so interface is more reusable as compare than inheritance.

RESULTANALYSIS FOR COHESION MEASURMENT

We applied source codes of inheritance and interface and calculated the data values of TCC and LCC using cohesion metrics show above in figure 7Calculated data shown in Table -2

TABLE 2 Calculation of TCC and LCC value for both the inputted source code for cohesion measurement

Metric's/ source code	Calculated values for Inheritance source code	Calculated values for Interface source code
TCC	0.0326	0.1327
LCC	0.0980	0.1327



Graph-1 the comparison between inheritance and interface based on calculated data of TCC and LCC

Here from the result of table-2 and graph-1, we can see the percentage of cohesion is more in interface than in inheritance. The value of LCC& TCC does not vary in interface we conclude that cohesion will increase in interface so interface is more reusable as compare to inheritance.

8. CONCLUSION

The purpose of this paper is how to reduce coupling and increase cohesion in object oriented programming. It issupportive for the developers to check which concept is best between inheritance and interface. When CBO is reduced then complexity also reduced and reusability will be increased and TCC and LCC is increase then reusability will increased. We have proposed an approach to measure the reusability of object oriented program based upon Cohesion and Coupling metrics. Since reusability is a characteristic of software quality, we can calculate software quality by measuring software reusability. Hence, this approach is important to measure difference between class inheritance and interface.

9. REFERENCES

[1] Bowman, M., Debray, S. K., and Peterson, L. L. 1993. Reasoning about naming systems. .

[2] Ding, W. and Marchionini, G. 1997 A Study on Video Browsing Strategies. Technical Report. University of Maryland at College Park.

[3] Fröhlich, B. and Plate, J. 2000. The cubic mouse: a new device for three-dimensional input. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems

[4] Tavel, P. 2007 Modeling and Simulation Design. AK Peters Ltd.

[5] Sannella, M. J. 1994 Constraint Satisfaction and Debugging for Interactive User Interfaces. Doctoral Thesis. UMI Order Number: UMI Order No. GAX95-09398., University of Washington.

[6] Forman, G. 2003. An extensive empirical study of feature selection metrics for text classification. J. Mach. Learn. Res. 3 (Mar. 2003), 1289-1305.

[7] Brown, L. D., Hua, H., and Gao, C. 2003. A widget framework for augmented interaction in SCAPE.

[8] Y.T. Yu, M.F. Lau, "A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions", Journal of Systems and Software, 2005, in press.

[9] Spector, A. Z. 1989. Achieving application requirements. In Distributed Systems, S. Mullender.