Better Object Oriented Paradigm Inheritance and Interface through Cohesion Metrics

Varsha Mishra
M.Tech Scholar
MIT Ujjain, India

Shweta Yadav
Assistant Professor
MIT, Ujjain, India

ABSTRACT
Measurement is fundamental to any engineering discipline. Cohesion metrics play an important role in empirical software engineering research as well as in industrial measurement programs. The Cohesion metrics presented in this paper measure the difference between class inheritance and interface programming.. This paper presents a measurement to measure cohesion by Lack of Cohesion in Methods (LCOM1), LCOM2 in object oriented programming. A measurement is done for C# inheritance and interface programs. The metric values of class inheritance and interface prove which program is good to use and beneficial for C# developers.

KEYWORDS
Inheritance, Interface, Locm1, Lcom2, Cohesion metrics.

1. INTRODUCTION
In the object oriented paradigm, cohesion of a class re-fers to the degree to which members of the class are in-terrelated. Chidamber and Kemerer defined the first met-ric to measure cohesiveness of a class [1].Cohesion, originating in structured design [23], refers to the relatedness of the elements in a module. A highly cohesive module is one whose elements have tight relationships among themselves while providing a single functionality of the module. In object-oriented paradigm, the class cohesion can be thought as the measurement of relatedness among the members of class [4]. Relatedness for a class means similarity in the methods exposed by a class. A highly cohesive module is one whose elements have a close relationship among them in order to provide the sole functionality of the module. All the advantages provided by the object-oriented paradigm are mainly based on the notion of class. Thus, all object-oriented design methods emphasize the importance of the correct identification of classes from the application domain, and developers spend significant time and effort to identify the essential classes relevant to the system. Although each object-oriented design method provides various guidelines to identify a set of classes from the application domain, there is a general agreement that a class should be created to abstract the state and the behavior of the similar objects by its members (i.e. instance variables and methods). However, poorly designed classes may be produced from the inappropriate use of object-oriented concepts during analysis and design phases or uncontrolled change during maintenance activities. A class may be poorly designed if the class fails to represent the features of the corresponding objects by its members; that is, a class has disparate and non-related members, or two or more different kinds of objects are captured by one class. The principle of high cohesion has been migrated to object-oriented design by Coad and Yourdon [5, 6] and recent research has again lead to a large number of new cohesion measures for object-oriented systems being defined. However, because cohesion is a complex software attribute in object-oriented systems (e.g., there are several different mechanisms which are considered to contribute to cohesion of a class), and there has been no attempt to provide a structured synthesis, our understanding of the state-of-the-art is poor. 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. 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 [7, 8, 9]. The main objective of this paper is to measure, analyze and propagate the difference between using object oriented class inheritance and interfaces in C# source code using cohesion measures by metrics.

2. COHESION METRICS
High cohesion has traditionally been a desirable property of classes in object-oriented software systems. Several metrics have been proposed to assess the degree of cohesion. Most of them are derived from Chidamber and Kemerer's LCOM (Lack of Cohesion in Methods) metric, which, in its original form, counts the number of pairs of methods in a class using no attribute in common [10].

A. Lack of Cohesion in Methods (LCOM)
LCOM is a count of the number of method pairs whose similarity is zero, minus the count of method pairs whose similarity is not zero. Raymond [11] discussed for example, a class C with 3 methods M1, M2, and M3. Let I1= {a, b, c, d, e}, I2= {a, b, e}, and I3= {x, y, z}, where I1 is the set of instance variables used by method M1. So two disjoint set can be found: I1 Ç I2 (= {a, b, e}) and I3. Here, one pair of methods who share at least one instance variable (I1 and I2). So LCOM = 2-1 =1. [10] States ―Most of the methods defined on a class should be using most of the data members most of the time. If LCOM is high, methods may be coupled to one another via attributes and then class design will be complex. So, designers should keep cohesion high, that is, keep LCOM low.

B. Lack of Cohesion in Methods (LCOM 1)

$LCOM1(c) = |\{\{m_1, m_2\} | m_1, m_2 \in M_1(c) \wedge m_1 \neq m_2 \wedge AR(m_1) \cap AR(m_2) \cap A_1(c) = \varnothing\}|$

Note that this definition only considers methods implemented in class c, and that only references to attributes implemented in class c are counted. This is an additional interpretation of our own; the influence of inheritance on the cohesion of a

class has neither been addressed by Henderson-Sellers nor by Chidamber and Kemerer [12].
In [10], Chidamber and Kemerer give a new definition of LCOM:
Consider a Class C_1 with methods $M_1, M_2 \ldots M_n$. Let $\{Ii\}$ = set of instance variables used by method Mi.
There are n such sets $\{I_1\} \ldots \{I_n\}$. Let P= $\{(I_i, I_j) \mid I_i \cap I_j = Æ\}$ and Q= $\{(I_i, I_j) \mid I_i \cap I_j \neq Æ\}$. If all n sets $\{I_1\} \ldots \{I_n\}$ are Æ then let P = ∅.

LCOM= |P| - |Q|, if |P| > |Q|

LCOM = 0, otherwise

LCOM is the number of pairs of methods in a class having no common attribute references,, minus the number of pairs of similar methods, . However, if, LCOM is set to zero. The definition of this version of LCOM is almost operational.

C. Lack of Cohesion in Method s (LCOM 2)

Let P = {∅, if AR (m) = ∅ ∀ m € M_1 (c)

P= {{m_1, m_2} | m_1, m_2 € M_1 (c) ^ $m_1 \neq m_2$ ^ AR (m_1) ∩ AR (m_2) ∩ A_1 (c) = ∅}, else

Let Q= {{m_1, m_2}| m_1, m_2 € M_1 (c) ^ $m_1 \neq m_2$ ^ AR (m_1) ∩ AR (m_2) ∩ A_1(c) ≠ ∅}.

Then LCOM2(c) = {|P| - |Q|, if |P| > |Q|

LCOM2(c) = 0, otherwise

LCOM1: Lack of cohesion in methods – The number of pairs of methods in the class using no attribute in common. [13]

LCOM2: LCOM1 minus number of pairs of methods that use common attribute. When difference is negative it is set to zero. [13]

LCOM3: Make an undirected graph with methods as vertices and edges between them if there is a common attribute used. LCOM3 is the number of connected components in graph. [13]

LCOM4: Same as LCOM3 but edge is also considered if method invokes another method. [13]

TCC: Tight Class Cohesion – Percentage pairs of methods which directly or indirectly use and attribute. [13]

LCC: Loose Class Cohesion: Same as TCC but methods indirectly connected are also considered. [13]

3. CLASS INHERITANCE AND INTERFACE

The engineering process and products or any software can be quantitatively measured and assessed for the merits and benefits of their improvements [14]. The limits of object oriented programming are characterized by the use of inheritance between classes. Inheritance is one of the initial features of object oriented programming and is often prototyped as an ingredient for object oriented programming. [15][16]. Using inheritance, a derived class receives the methods and attributes of the base class and this relationship is referred as "is-a" or "isa-kind-of" relationship. Inheritance will create class hierarchy. Today interfaces are heavily used in all disciplines. Interfaces are advisable to be used in large type of applications because the interface makes the application easier to extend, modify and integrate new features. Inheritance also can be applied to interfaces. Interfaces can help to prevent the misuse of inheritance. Interfaces can be used like classes in declarations and signatures. By using the interface concept java helps to produce reusable code. Interfaces are prototype for a class and also inheritance hierarchy of interfaces is independent of class inheritance tree [17] [18] [19] [20].

4. BACKGROUND

The concept of interfaces has been measured in java programming by Fried Stiemann and Co [21] who represented the usage of interfaces compared to classes as 4:1. Ken Pugh [17] said that finding commonality among classes makes more it effective for object oriented programming. He explored the commonality in using inheritance and interfaces in object oriented programming. In this paper, the usage of interfaces is increased and the benefits of using interfaces are shown by cohesion measures. V. Krishnapriya and Dr. K. Ramar [2], through surveys and experiments, identify complexity between inheritance and interface programming using Coupling Matrices. It measure coupling between object (CBO), number of associations between classes (NASSocC), number of dependencies in metric (NDepIN) and number of dependencies out metric (NDepOut) in object oriented programming. The metric values of class and inheritance diagrams have been compared to prove which concept is good to use and beneficial for developers.
In 2011 Narendra Pal Singh Rathore, Ravindra Gupta [22] presented a novel approach to Measure difference between Inheritance and Interface to find better OOP Paradigm using C# through coupling metrics.

5. PROPOSED APPROACH

Goal: Comparing the inheritance and interface concepts in object oriented programming through cohesion- metrics.
Hypothesis: Two object oriented metrics are used for cohesion measures in C# inheritance source code and interface source code.
1. One object oriented C# program is used with inheritance concept in this paper.
2. This program is introduced with maximum possible interfaces.
3. Both Cohesion metrics are applied to both inheritance and interface C# program.
4. The results are compared between inheritance and interface cohesion measures.

6. RESULT

Now we have applied cohesion metrics on an example and calculated LCOM1 & LCOM2. All the approach is described below for particular example itself.

Example I: Implementation Using Inheritance:

```
using System;
class Shape
{
   public void Draw();
   public void Element();

}
```

```
class RegularPolygon:Shape
{
   public int Linesegment;
   public void Perimeter();

}
class Ellipse: Shape
{
  public  int  curved;
  public int  Surface;

}
class Triangle : RegularPolygon
{
      public int sumofangles = 180;
      public void setsides();
      public void Area();
}
class Rectangle : RegularPolygon
{
     public  int sumofangles = 360;
     public void setsides();
     public void Area();

}
class Circle : Ellipse
{
   public int symmetrical;
   public void Circumference();

}
class Salene : Triangle
{
   public int Nosidesequal;

}
class Isosceles : Triangle
{
   public int sideequal2;
   public int Anglesequal2;

}
class Equilateral : Triangle
{
   public int sidesequal3;
   public int Anglesequal3;

}
class Square : Rectangle
{
   public int oppositesidequal;
   public int angles;
}
```

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

1.	Shape	S
2.	Regular Polygon	Rp
3.	Ellipse	E
4.	Triangle	T
5.	Rectangle	R
6.	Circle	C
7.	Selene	Se
8.	Isosceles	Is
9.	Equilateral	Eq
10.	Square	Sq

LCOM1
Formula LCOM1 =|P|-|Q|

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

P = {<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>, <R, Eq> →4

<C, Se>, <C, Is>, <C, Eq>, <C, Sq> →4

<Se, Is>, <Se, Eq>, <Se, Sq> →3

<Is, Eq>, <Is, Sq>, <Eq, Sq> } →2+1

Total no of pairs in P is (7+6+6+3+4+4+3+2+1) = 36

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

Q = 9

LCOM1=|36| - |9| = 27

LCOM2:
Formula of LCOM2 =| [4|p| - n (n-1)] / 2 |
Where the 'P': No of Pairs in Disjoint set
No of classes are: n=10
Then
LCOM2 is =| [4|36|-10*9]/2 | = |54/2|= 27

LCOM2=27

Example I: Implementation Using Interface:

```
using System;
interface Shape
{
   public void Draw_Element();
}
interface RegularPolygon
{
   public void Linesegment();
   public void Perimeter();
}
interface Ellipse
{
   public void Circumference();
}
class Triangle : Shape
{
   int Sumofangles = 180;
   public void Draw_Element();
   public void setsides();
   public void Area();
```

```
}
class Rectangle : RegularPolygon
{
   int sumofangles = 360;
   public void Perimeter();
   public void Linessegment();
   public void setsides();
   public void Area();
}
class Circle : Ellipse
{
   int symmetricalpictur;
   public void Circumference();
}
class Scalene : Triangle
{
   int notequalsides;
}
class Isosceles : Triangle
{
   int sidesequal;
   int angleequal;
}
class Equilateral : Triangle
{
   int sidesequal;
   int angleequal;
}
class square : Rectangle
{
   int opposite;
   int sidesequal;
   int anglesequal;
}
```

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

1. Triangle T
2. Rectangle R
3. Circle C
4. Scalene Sc
5. Isosceles Is
6. Equilateral Eq
7. Square Sq

LCOM1:

Formula LCOM1 =|P|-|Q|

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

P = {<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 (3+4+4+3+2+1) = 17

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

LCOM1=|17|-|4|= 13

LCOM2: Formula of LCOM2 =| [4|p| - n (n-1)] /2 |[+]
Where the 'P': No of Pairs in Disjoint set
No of classes are: n=7
Then LCOM2 is =| [4|17|-7*6]/2 | = |26/2|= 13

LCOM2=13
The calculated values are of LCOM1 & LCOM2 for the same example:

Metrics/ Approach	Inheritance	Interface
LCOM1	27	13
LCOM2	27	13

Table 1: Cohesion Measure for Example I

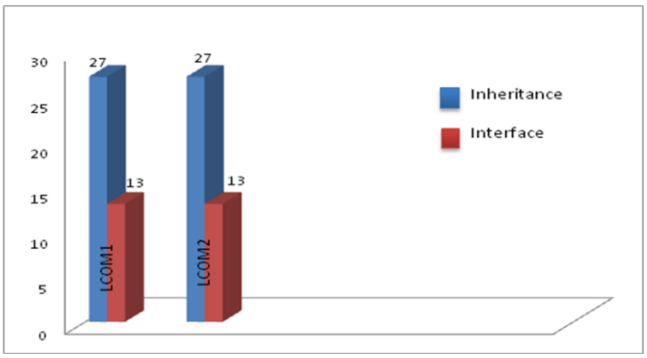

Fig. 1 : Cohesion Measure for Example I

In the table and graph above we see that the values tend to be generally lower for interface program as compared to inheritance program, which mean that cohesion is actually higher. High cohesion is associated with several desirable traits of software engineering like robustness, reliability, reusability and understand-ability.

7. CONCLUSION

To improve modularity and encapsulation the inter class cohesion measures should be larger. In Comparison of cohesion in between inheritance and interface for the modules, functions, attributes, classes in oops through cohesion metrics is done, and interface is calculated as more reusable code then inheritance. The more independent a class it is easier to be reused by another application. The further developed metrics are given that also can be implement in realistic behaviour so that an efficient way can be identified to optimise our approach for development of IT products. Due to the reduction in values of cohesion metrics the stability of the structure will be good. By using more interfaces compared to inheritance the cohesion measures are increased.

8. REFERENCES

[1] P. Chidamber and C. Kemerer, "Towards a Metrics Suite for Object Oriented Design," *Proceedings of 6th ACM Conference on Object Oriented Programming, Systems, Languages and Applications*, Phoenix, Arizona, 1991, pp. 197-211.

[2] V. Krishnapriya and Dr. K. Ramar, Exploring the Difference between Object Oriented Class Inheritance and Interfaces Using Coupling Measures. IEEE 2010.

[3] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*

[4] J. M. Bieman and B.K. Kang, "Cohesion And Reuse In An Object-Oriented System", Proceedings of The Symposium on Software Reusability (SSR'95), Seattle, 1995, pp. 259-262.

[5] P. Coad, E. Yourdon, "Object-Oriented Analysis", *Prentice Hall*, second edition, 1991.

[6] P. Coad, E. Yourdon, "Object-Oriented Design", *Prentice Hall*, first edition, 1991.

[7] Christopher L. Brooks, Chrislopher G. Buell, "A Tool for Automatically Gathering Object-Oriented Metrics", IEEE, 1994.

[8] Rajib Mall, "Fundamentals of Software Engineering", Chapter 1, Pg.No:1-18,2nd Edition, April 2004.

[9] Rudiger Lincke, Jonas Lundberg and Welf Lowe,"Comparing Software Metrics Tools", ISSTA'08, July 20-24, 2008, ACM 978-1- 59593-904-3/07.

[10] Chidamber, S. R. and Kemerer, C. F., "A Metric Suite for Object-Oriented Design," IEEE Transactions on Software Engineering, 20(6):476-493, 1994.

[11] Alexander et al 2003,‖Mathematical Assessment of Object-Oriented Design Quality‖, IEEE Transactions on Software Engineering, VOL. 29, NO. 11, November 2003.

[12] Lionel C. Briand, John W. Daly, and Jürgen Wüst, A Unified Framework for Cohesion Measurement in Object-Oriented System. ISERN-97-05.

[13] N. Kayarvizhy, S. Kanmani, Analysis of Quality of Object Oriented Systems using Object Oriented Metrics, 978-1-4244-8679-3/11/$26.00 ©2011 IEEE.

[14] Christopher L. Brooks, Chrislopher G. Buell, "A Tool for Automatically Gathering Object-Oriented Metrics", IEEE, 1994.

[15] Fisher K. and Reppy J, "The Design of Class Mechanism for Mobby", In Proceedings of the Conference on Programming Language Design and Implementation (PLDI), P.No:37-49, May 1999.

[16] Wegner .P. "Dimension of Object-Based Language Design ", In Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications, OOPSLA, Oct 1987.

[17] Ken Pugh," Interface Oriented Design", Chapter 5, 2005.

[18] ISRD GROUP," Introduction to object oriented programming through java", TATA Mc Graw HILL, Pg.No:109.

[19] Friedrich Stiemann, Philip Mayer and Andreas Meibner, "Decoupling Classes with Inferred Interfaces", Proceedings of the 2006 ACM Symposium on Applied Computing, P.No:1404 – 1408.

[20] Markus Mohnen, "Interfaces with Default Implementations in Java", Technical Report.

[21] Fried Stiemann, Wolf Siberski and Thomas Kuhne, " Towards the Systematic Use of Interfaces in Java Programming", 2nd Int. Conf. on the Principles and practice of Programming in Java PPJ 2003, P.No:13-17.

[22] Narendra Pal Singh Rathore, Ravindra Gupta, A Novel Coupling Metrics Measure difference between Inheritance and Interface to find better OOP Paradigm using C#. IEEE 2011 978-1-4673-0125-1_c 2011 IEEE

W. Stevens, G. Myers, and L. Constantine, "Structured Design," BM Systems J., vol. 12, no. 2, 1974