Measurement of Design Complexity of Different types of Inheritance using Cohesion Metrics

Ankita Mann
Mtech Scholar,
DCSA, M.D. University
Rohtak, Haryana, India

Sandeep Dalal
Assistant Professor, DCSA,
M.D. University Rohtak,
Haryana, India

Neetu Dabas, Ph.D
Maharshi Dayanand University
Rohtak, Haryana, India

ABSTRACT
Main aim of Software Engineering is to increase quality and maintain Software Product. Inheritance reflects the degree of reusability of existing classes and reuse increases productivity. Most Cohesion Metric tool do not consider inherited elements while measuring cohesion but we can measure design quality by including the concept of inheritance in Cohesion metrics. In this paper values of all cohesion metrics (LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, CO, TCC and LCC) is calculated including the concept of inheritance for Single, Multiple, Multilevel and Hierarchical Inheritance and compare results to determine design complexity of various types of Inheritances.

Keywords
Single inheritance, multiple inheritance, multilevel inheritance and Hierarchical Inheritance.

1. INTRODUCTION
Most important goal of Software Engineering is to develop a good quality Software that is stable and maintainable.

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

Inheritance is an necessary concept in which a class acquires access of all attributes and methods of class it inherits from and can revolutionize. Effective reuse increase productivity and measurements are essential for this. All existing work evaluates Inheritance in terms of Depth but there are more aspects that are not covered by existing Inheritance Metrics. Classes may depend on each other in various ways and some dependencies do not violate quality. Highly Cohesive module reflects Good Quality. In this paper Cohesion in all types of Inheritances is evaluated to measure design complexity of program that implement Inheritance.

2. INHERITANCE
In inheritance, the derived class inherits public and protected members of the base class and all new members. In class hierarchies derived class has a "kind of" relationship with the base class.

"*Inheritance* is a mechanism of reusing and extending existing classes without modifying them, thus producing hierarchical relationships between them."- IBM

With the help of Inheritance a Class can be defined in terms of another class without modifying existing class. Use of inheritance avoids redefining the inherited information from the base class in our derived classes.

2.1 Types of Inheritance
- **Single Inheritance** is a common form of inheritance in which classes have only one base class. Single inheritance is safe to use as compared to other types of Inheritances.
- **Multilevel Inheritance** is the hierarchy in which subclass acts as a base class for other classes means a class be can derived from a derived class.
- **Hierarchical Inheritance** is the hierarchy in which multiple subclasses inherit from one base class. It is a method of in which one or more derived classes are derived from common base class.
- **Multiple Inheritance** is the hierarchy in which one derived class inherits from multiple base class/es. With the help of multiple inheritance program can be structured as a set of inheritance lattices instead of a set of inheritance trees.

2.2 Visibility Mode and Inheritance
Visibility mode controls the scope of inherited base class members in the derived class. It can be either private or protected or public. Private is used as access modifier if we want confidential data.

- **Private Inheritance:** In private inheritance protected and public members of base class become private members of the derived class.
- **Public Inheritance:** In public inheritance, the protected members of base class become protected members and public members of the base class become public members of derived class.
- **Protected Inheritance:** In protected inheritance, the protected and public members of base class become protected members of the derived class.

Table 1: Cohesion metrics used in this research

Metrics	Description
LCOM1	Total numbers of pairs of methods that share common attribute.
LCOM2	|P|-|Q|, P is pair of method without common attribute and Q is pair of method with common attribute.
LCOM3	|Connected component of Graph G|
LCOM4	|Connected component of Graph G|+ Additional edge for method-method invocation
LCOM5	(a – kℓ) / (ℓ – kℓ), ℓ- number of attributes, *k*- number of methods, *a*- Total number of distinct attributes accessed by each method in a class.
Co	a / kℓ or 1 – (1 – 1/k)LCOM5
TCC	NDC/NP, NDC- Directly connected public methods, NP- Number of possible connections
LCC	NIC/ NP, NIC- Indirectly connected public methods

3. DESCRIPTION OF CASES DISCUSSED

In this research Design Complexity for four different types of Inheritances: Single, Multilevel, Hierarchical and Interfaces is discussed and measured with the help of eight different Cohesion Metrics (LCOM1, LCOM2, LCOM3, LCOM4, LCOM5, Co, TCC and LCC) and Compare results. Cases Discussed are as follows:

Case 1: Single Inheritance:

In this case Base class contain attribute “X”, method “Basemethod” and derive class contain method “Derivemethod”. Attribute “X” ia accessed by method of base class as well as for derived class.

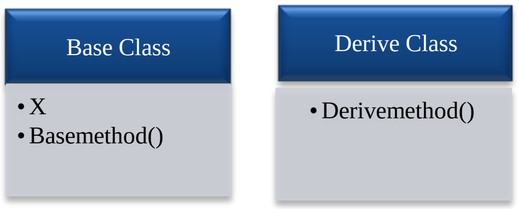

Figure1: Single Inheritance

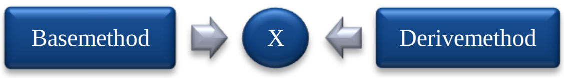

Figure2: Relation between methods and attribute of case1

Code:

```
Using System;

Class Base

{

Public int X;

          Public void basemethod (int Z)

          {

                    Z=X;

                    Console.WriteLine(“X=”,X);

          }

}

    Class Derive:Base

    {

          Public void Derivemethod (int Z)

          {

                    Z=X;

                    Console.WriteLine(“X=”,X);

          }

    }

    Class Simple Inheritance

    {

          Public void static Main()

          {

                    Base base= new Base();

                    Derive derive= new derive();

                    base.BaseMethod(10);

                    derive.DeriveMethod(20);

                    derive.BaseMethod(30);

          }

}
```

Case2: Multilevel Inheritance

In this case three Clsses are there A, B and C. C inherit all public and protected members of B which inherit A. A implement attribute “a” and “Amethod()”, B implement “b” and “Bmethod()” and C implement “Cmethod()”. C inherit all public and protected elements of both A and B. in this example “a” is accessed by “Amethod()” and “Bmethod()” and “Cmethod”.

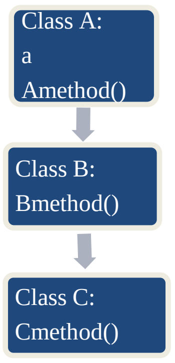

Figure3: Multilevel Inheritance

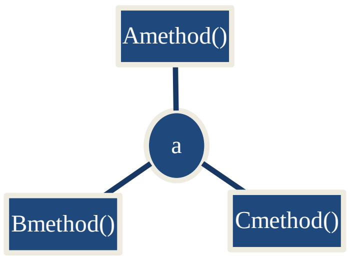

Figure4: Relation between methods and attribute of case2

Code:

```
Using System.Text;

Namespace Multilevel_Inheritance

{

Class A

{

        Protected int a;

        Public Amethod(int x)

        {

                a=x;

                Console.WriteLine("a=",+a);

        }

}

Class B: A

{

        Public Bmethod(int y)

        {

                a=y;

                Console.WriteLine("a=",+a);

        }

}

Class C: A

{

        Public Cmethod(int z)

        {

                a=z;

                Console.WriteLine("a=",+a);

        }

}

Class MultiLevelInheritance

{

        C c= new C;

        c.Amethod(10);

        c.Bmethod(20);

        c.Cmethod(30);

}

}
```

Case 3: Hierarchical Inheritance

In this case two classes C and D inherit A and D and E inherit C. Class A contain "A" and "Amethod()", Class B contain "Bmethod()", C contain "c","Cmethod()", D contain "Dmethod()" and E contain "Emethod()". "a" is accessed by Amethod(), Bmethod(), Cmethod(), Dmethod() and Emethod(). "C" is accessed by Cmethod(), Dmethod() and "Emethod()". "Cmethod()", "Dmethod()" and "Emethod()" access both variables "a" and "c".

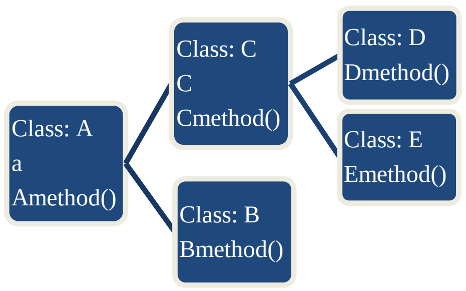

Figure5: Hierarchical Inheritance

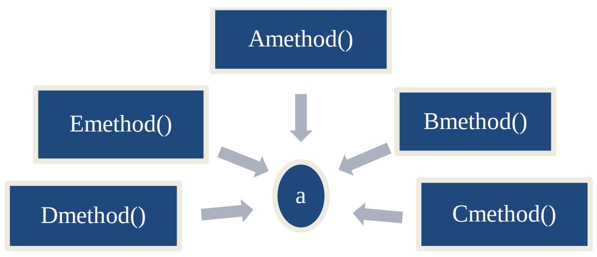

Figure6: Relation between methods and attribute “a”

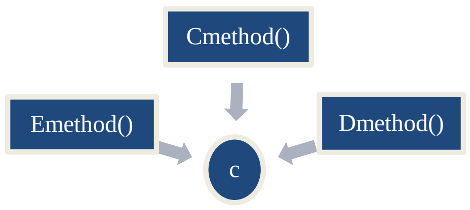

Figure7: Relation between methods and attribute “c”

Code:

```
Using System.Text;
Namespace Multilevel_Inheritance
{
Class A
{
        Protected int a;
        Public Amethod(int x)
        {
                a=x;
                Console.WriteLine(“a=”,+a);
        }
}
Class B: A
{
        Public Bmethod(int y)
        {
                a=y;
                Console.WriteLine(“a=”,+a);
        }
}
Class C: A
{
        Protected int c;
        Public Cmethod(int i, int j)
        {
                a=i;
                c=j;
                Console.WriteLine(“a=”,+a);
                Console.WriteLine(“c=”,+c);
        }
}
Class D: C
{
        Public Dmethod(int i, int j)
        {
                a=i;
                c=j;
                Console.WriteLine(“a=”,+a);
                Console.WriteLine(“c=”,+c);
        }
}
Class E: C
{
        Public Emethod(int i, int j)
        {
                a=i;
                c=j;
                Console.WriteLine(“a=”,+a);
                Console.WriteLine(“c=”,+c);
        }
}
Class MultiLevelInheritance
{
        E e= new E;
        e.Amethod(10);
        e.Bmethod(20);
        e.Cmethod(30,40);

        e.Dmethod(50,60);

        e.Emethod(70,80);

}

}
```

Case 4: Interface

In this case base class implement Interface, here interface "Display" contain method "Print()", Class "A" and Class "B" override method "Print()". Class "A" inherits Interface Display and Class "B" inherits Class "A". In this type of situation object of sub class converted to interface type.

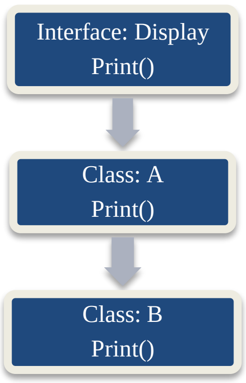

Figure8: Interface

Code:

```
Using System;

Interface Display

{

Void Print();

}

Class B: Display

{

        Public void print()

        {

                Console.WriteLine("Base Class
                Method");

        }

}

Class D: B

{

        Public void print()

        {

                Console.WriteLine("Base Class
                Method");

        }

}

Class Interacetest

{

        Public static void main()

        {

                B b= new b();

                b.print();

                Display d= (Display) d;

                d.print();

        }

}
```

4. RESULTS

Table 2 represents calculated values for all Cohesion metrics in four type of inheritance. Here we compare values of different Cohesion metrics min all Cases.

LCOM1: Value of LCOM1 is zero in all cases except Hierarchical Inheritance means performance for LCOM1 is good in Single, Multiple and Multilevel Inheritance because LCOM1 should be low for highly cohesive module.

LCOM2: LCOM2 is zero for all cases, means performance of all types of Inheritance is same for LCOM2.

LCOM3: LCOM3 give number of independent components in program which should be one. Value of LCOM3 is one for all cases except multiple inheritances.

LCOM4: LCOM4 is one for all so performance is similar in all cases. And one indicates that code is highly cohesive.

LCOM5: LCOM5 is zero for Single, multiple and multilevel cohesion which is a good value. Increasing LCOM5 above 0 towards 1 means worsening cohesion. So we can say that performance of hierarchical inheritance is poor for LCOM5.

Co: Co is little variation of LCOM5**.**

TCC and LCC: For TCC and LCC one is considered as highly cohesive module and if value is less than half module is considered as non cohesive. Here TCC and LCC is one for all cases except hierarchical inheritance.

Table2: Calculated values of Cohesion metrics for Single, Multiple, Hierarchical and Interface

Metrics	LCOM1	LCOM2	LCOM3	LCOM4	LCOM5	Co	TCC	LCC
Single Inheritance	0	0	1	1	0	1/4	1	1
Multilevel Inheritance	0	0	1	1	0	1	1	1
Hierarchical Inheritance	5	0	1	1	3/5	3/5	½	2/5
Multiple Inheritance	0	0	2	1	0	0	1	1

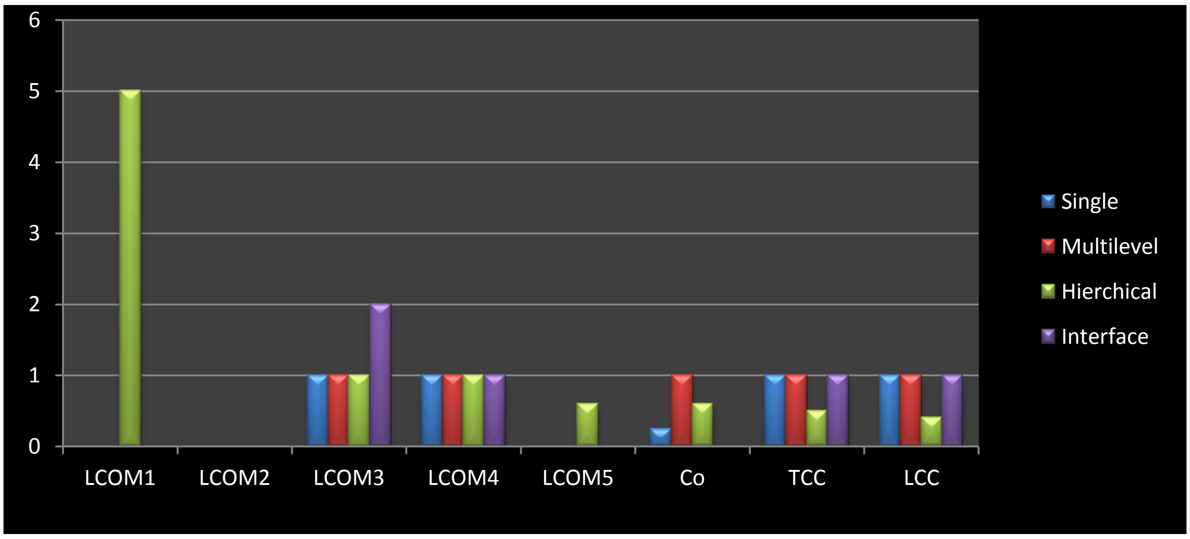

Figure 9: Graphical representation of all Cohesion metric values

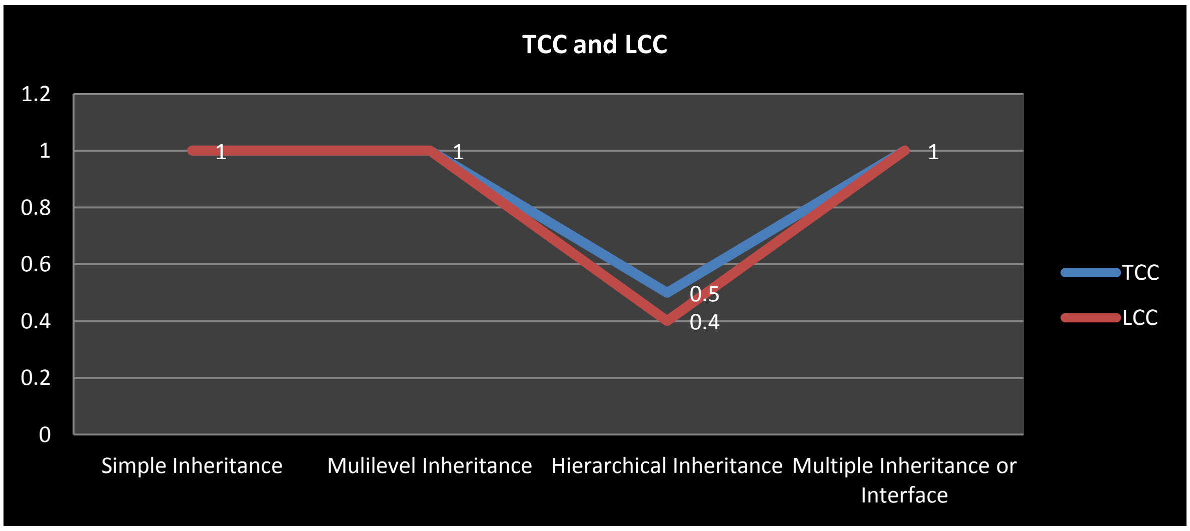

Figure 10: Graphical representation for of TCC and LCC in all Cases

5. CONCLUSION

Figure 9 represent difference between Cohesion Metric values for different types Inheritances. Conclusion generated from these values is that Hierarchical Inheritance is worst inheritance and interface is best. Values for all other cases are almost similar except hierarchical Inheritance. LCOM1 is zero for all types of inheritances except hierarchical inheritance and LCOM2 is zero for all cases. LCOM3 is one for all cases except interface. LCOM4 is one in all cases and LCOM5 show variation only for hierarchical inheritance. Co is different in all cases and similarly TCC and LCC are also different for hierarchical Inheritance. TCC and LCC should be one for highly cohesive and effectively designed code. According to these results this is we want to show that there is also discrepancy in types of Inheritance and quality of design vary with type of Inheritance. According to this research use of hierarchical inheritance should be avoided and using Interfaces for better design and due to its good results with TCC, LCC LCOM1 an LCOM4. In future this topic can be extended by working on the concept of overloading and overriding of methods and use of public, private, protected and internal members of class.

6. REFERENCES

[1] Aine Mitchell's " An Empirical Study Of Run-Time Coupling And Cohesion Software Metrics" 2005

[2] Danial C. Halbert And Patrick D. O'brian's "Using Type And Inheritance In Object Oriented Metricss"

[3] Safwat M. Ibrahim, Sameh A. Salem, Manal A. Ismail, And Mohamed Eladawy's "Identification Of Nominated Classes For Software Refactoring Using Object-Oriented Cohesion Metrics", Ijcsi International Journal Of Computer Science Issues, Vol. 9, Issue 2, No 2, March 2012

[4] M.V.Vijaya Saradhi 1 B.R.Sastry "A Quality Indicator For Software Interoperability", International Journal Of Engineering Science And Technology Vol. 2(7), 2010, 2587-2594

[5] G. Sri Krishna And Rushikesh K. Joshi's "Inheritance Metrics: What Do They Measure?"

[6] Serge Demeyer And St´Ephane Ducasse's "Metrics, Do They Really Help?"

[7] Seyyed Mohsen Jamali's "Object Oriented Metrics"

[8] Ferderick T. Sheldon, Kshamta Jerath And Hong Chung "Metric For Maintainability Of Class Inheritance Hierarchy"

[9] Alan Snyder's "Encapsulation And Inheritance In Object-Oriented Programming Languages"

[10] Ferid Cafer'S "Estimating Complexity Of A Software Code"

[11] Randy Charles Morin's "Oop Concepts By Example"

[12] Kenneth Baclawski And Bipin Indurkhya's "The Notion Of Inheritance In Object-Oriented Programming"

[13] Al Lake's "Use Of Factor Analysis To Develop Oop Software Complexity Metrics"

[14] Letha Etzkorn, Carl Davis, And Wei Li "A Statistical Comparison Of Various Definitions Of The Lcom Metric"

[15] Luca Cardelli's "A Semantics Of Multiple Inheritance", Information And Computation 76, 138-164,1988

[16] Dirk Beyer, Claus Lewerentz And Frank Simon's "Impact Of Inheritance On Metrics For Size, Coupling, And Cohesion In Object Oriented Systems"

[17] E Da-Wei's "The Software Complexity Model And Metrics For Object-Orient