An Effort to Improve Cohesion Metrics Using Inheritance

Ankita Mann[1], Sandeep Dalal[2],Dhreej Chhillar[3]

[1,2,3,]Department of Computer Science and Applications M.D. University Rohtak, Haryana, India

ABSTRACT

Software metrics are used to assess and improve the quality of Software Product. Class Cohesion is degree of relatedness of class members and important measurement attribute of quality of a software product. But most of the existing cohesion tools and approaches do not consider inherited elements. But to measure overall design of product it is necessary to include all inherited elements. In this paper we calculate all existing metrics on implemented elements as well as on inherited elements to measure quality of design.

KEYWORDS: *Software metrics, Cohesion.*

I. INTRODUCTION

Concept of Software cohesion was started in 1970's by Stevens et al. [7] who wants inter module metrics for procedural Software's. Software cohesion is a measure of degree to which elements of a module belong together and measured at class level and it is most important Object Oriented Software attribute.

1.1. Types of cohesions:

Method Cohesion: Method is a module or bracketed piece of code that implement some functionality. Method cohesion focuses on the methods you write inside the class. Classical cohesion adapts seven degree of cohesion summarized from worst to best:

Coincidental Cohesion	Worst Cohesion
Logical Cohesion	
Temporal Cohesion	↓
Procedural Cohesion	
Communicational Cohesion	
Sequential Cohesion	
Functional Cohesion	Best Cohesion

Table 2: Classification Classical Cohesion

1.2.Class Cohesion: Classes are the fundamental units of object-oriented programs, and provided as the units of encapsulation promoting their modifiability and reusability. Class cohesion measure quality of classes by measuring binding of the elements defined with in the class. Inherited instance variables and inherited methods are not considered. Member methods are elements of a class and perform a common task and allow you to systematize class reasonably. Cohesiveness of a class depends upon its representation of its object as single, meaningful data abstraction. Cohesion of a class is rated as separable if its objects signify various unconnected data abstractions combined in one ne object. In this case instance variables and methods of class are partitioned into sets such that no method of one set uses instance variable or invoke method of a different set. For example if you have ten methods that act as utilities for network operations, you can very well split the class into two classes: one serving the utilities and the other, network operations. After split each separated class represent single data abstraction.

II. MAJOR EXISTING COHESION METRICS

Classes are basic units of Object oriented Software and should be designed to have good quality. Improper modeling produce improper responsibilities and assignment and result will be low cohesion. In order to assess class cohesion in object oriented systems we have several metrics. Cohesion Metrics measure togetherness of method of a class. A highly cohesive class performs one function. Most of the Cohesion Metrics are based on LCOM (lack of cohesion in methods) metrics defined by Chidamber and Kemerer[Chidamber 91] and it is also refined by many others. The Chidamber & Kemerer metrics suite originally consists of 6 metrics for each class: WMC, DIT, NOC, CBO, RFC and LCOM1. Suite has later been amended by RFC´, LCOM2, LCOM3 and LCOM4 by other authors.

- **LCOM1:** LCOM1 was introduced in Chidamber & Kemerer metrics suite. This metric calculate the number of pairs of methods in class using no instance variable in common.

LCOM1 = P – Q, if value of result is positive. LCOM1 = 0 if value of subtraction is negative.
0 indicates that class is cohesive and LCOM1 > 0, indicates that the class needs or can be split into two or more classes.

- **LCOM2**:P is Number of pairs of methods that do not share attributes and Q = Number of pairs of methods that share attributes.

LCOM2=P-Q if P-Q>0 and otherwise LCOM2=0

- **LCOM3:** =LCOM3 is proposed by Li & Henry, 1993 and calculated by number of disjoint components in the graph that represents each method as a node and the sharing of at least one attribute as an edge.

- **LCOM4:** LCOM4 measures the number of "connected components" in a class. A connected component is a set of related methods (and class-level variables).

LCOM4=1 indicates a cohesive class, which is the "good" class.
LCOM4>=2 indicates a problem. The class should be split into so many smaller classes.
LCOM4=0 happens when there are no methods in a class. This is also a "bad" class.

- **LCOM5:** LCOM5 is purposed by Henderson-Sellers.

LCOM5 = (a – kℓ) / (ℓ – kℓ),
ℓ => number of attributes,
k=> number of methods,
a => summation of the number of distinct attributes accessed by each method in a class.

- **Coh:** Coh is variation in LCOM5.

Coh = a / kℓ or Coh = 1 – (1 – 1/k)LCOM5

- **TCC (Tight Class Cohesion):** TCC is defined as the percentage of method pairs that are directly related.

Consider a class with n public methods. Let NP be the maximum number of public method pairs:
NP = maximum number of possible connections= N * (N-1) / 2 where N is the number of methods
NDC = number of direct connections (number of edges in the connection graph)
Tight class cohesion TCC = NDC/NP

- **LCC (Loose Class Cohesion):** LCC is defined as the relative number of indirectly connected public methods. NIC is number of direct or indirect connections between public methods.

NID = number of indirect connections.
LCC= NIC/ NP

III. INHERITANCE AND COHESION

Inheritance is a key concept in object-oriented programming and implementing the inheritance relations in a software system properly is expected to improve the quality of the system. [12] But most of the existing metrics tools do not consider inherited methods. But design of full project depends upon both inherited as well as implemented methods. To measure overall design complexity both study of both elements is required. Therefore there is need to examine cohesion with or without inherited elements.

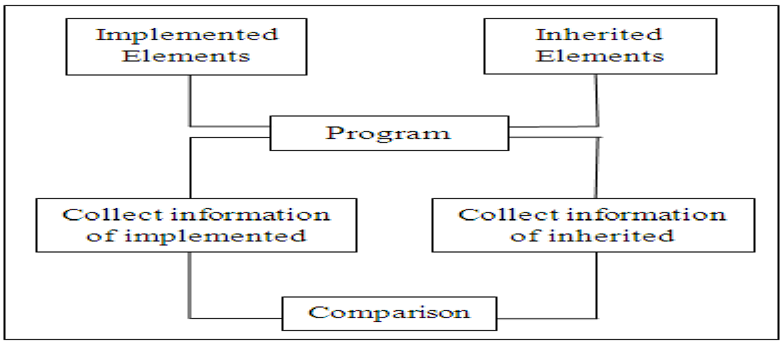
Implemented Elements
Inherited Elements
Program
Collect information of implemented
Collect information of inherited
Comparison

Fig. 1: Procedure Followed in this paper

Three Cases discussed in this paper are:
Case1:

```
Using System;
Class Base
{
Protected int Volume;
Public out()
{
Console.writeline("this is base class");
}
}
Class Sub: Base
{
Public Conevol()
{
Float Radius, Height;
Console.writeLine("enter radius of cube");
Radius= Console.readLine;
Console.writeLine("enter height of cube");
Height= Console.readLine;
Volume=1/3*22/7*Radius*Radius*Height;
}
Public Cylindervol()
{
Float Radius, Height;
Console.writeLine("enter radius of cylinder");
Radius= Console.readLine;
Console.writeLine("enter height of cylinder");
Height= Console.readLine;
Volume=22/7*Radius*Height;
}
Public Prismvol()
{
Float Width, Height, Lenght;
Console.writeLine("enter width of prism");
Width= Console.readLine;
Console.writeLine("enter height of prism");
Height= Console.readLine;
Console.writeLine("enter length of prism");
Length= Console.readLine;
Volume= Length*Width*Height;
}
}
Class mainclass
{
Public static void Main()
{
Sub sub=new Sub();
sub. out();
sub. Conevol();
sub. Prismvol();
}
}
```

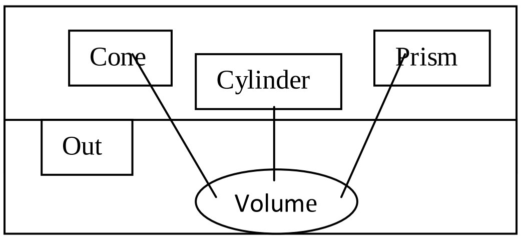

Fig. 2: Diagrammatic representation for Case1

In this example Cone, Prism and Cylinder are implemented elements of sub class that share a common attribute Volume that is element of super class here cohesion is low if only implemented elements are considered because there is no connection between implanted methods if we exclude inherited elements. Out is method of super class and there is no link of out with any other implemented or inherited element. Values of TCC and LCC are zero if we exclude inherited elements.

Implemented Elements	Inherited Elements	Metrics
3	3	LCOM1
3	0	LCOM2
3	2	LCOM3
3	2	LCOM4
0	1/3	LCOM5
1	7/9	Coh
0	½	TCC
0	2/6	LCC

Table 4: Result for Cohesion metrics including and excluding inherited elements

Case2:

```
Using System;
Class Base
{
int Area;
protected square()
{
int side;
Console.writeLine("enter side of square");
side= Console.readLine;
Area= side*side;
Console.writeLine("Area of square is="Area);
}
protected rectangle()
{
int length, breath;
Console.writeLine("enter sides of rectangle");
lenght= Console.readLine;
breath = Console.readLine;
Area= lenght*breath;
Console.writeLine("Area of rectangle is="Area);
}
}
Class Sub: base
{
Public method1()
{
Console.writeLine("subclass method1");
}
Public method2()
{
Console.writeLine("subclass method2");
}
}
Class test
{
Public static void main()
{
Sub sub= new sub();
sub.square();
sub.method1();
sub.method2();
}}
```

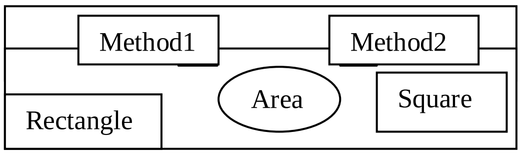

Fig. 3: Diagrammatic Representation for Case2

In this program Method1 and Method2 are implemented method in sub class and rectangle and square are method of inherited class that shares a common super class variable Area. In this example cohesion of sub class elements is almost zero So if we include inherited elements in calculating cohesion of this code results are better.

Implemented Elements	Inherited Elements	Metrics
1	5	LCOM1
1	4	LCOM2
2	3	LCOM3
2	3	LCOM4
0	2/3	LCOM5
1	½	Coh
0	1/6	TCC
0	1/6	LCC

Table 5: Result for Cohesion metrics including and excluding inherited elements

Case3:

```
using System;
namespace InheritanceApplication
{
   class Shape
   {
      protected int width;
      public void setWidth()
      {
         width = w;
      }
Class Rectangle: Shape
{
protected int height;
 public void setHeight(int h)
   {
     Height= h;
    }
Public area()
{
Return (width * height);
}
Public volume()
{
Return (1/2(width + height));
}
}
class RectangleTester
   {
      Public static void Main()
      {
         Rectangle Rect = new Rectangle();
         Rect.setWidth(5);
         Rect.setHeight(7);
         Console.WriteLine("Total area: {0}",  Rect.Area());
         Console.WriteLine("Total area: {0}",  Rect.volume());
         }
```

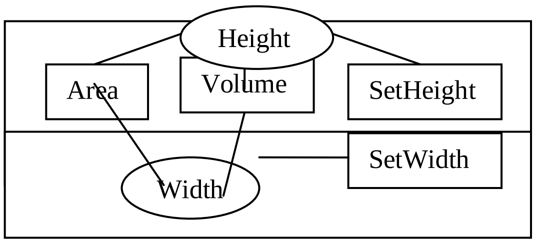

Fig. 4 Diagrammatic Representation for Case3

In this case Area, Volume, Setheight and height are implemented elements. Width and Set width are elements defined in base class. Design of this program is better than other cases. Cohesion of this program is good in both cases but value of LCC indicates design quality of this program.

Implemented Elements	Inherited Elements	Metrics
0	1	LCOM1
0	0	LCOM2
1	1	LCOM3
1	1	LCOM4
0	0.333	LCOM5
1	0.75	Coh
1	0.8333	TCC
0.666	1	LCC

Table 5: Result for Cohesion metrics including and excluding inherited elements

CASE1:

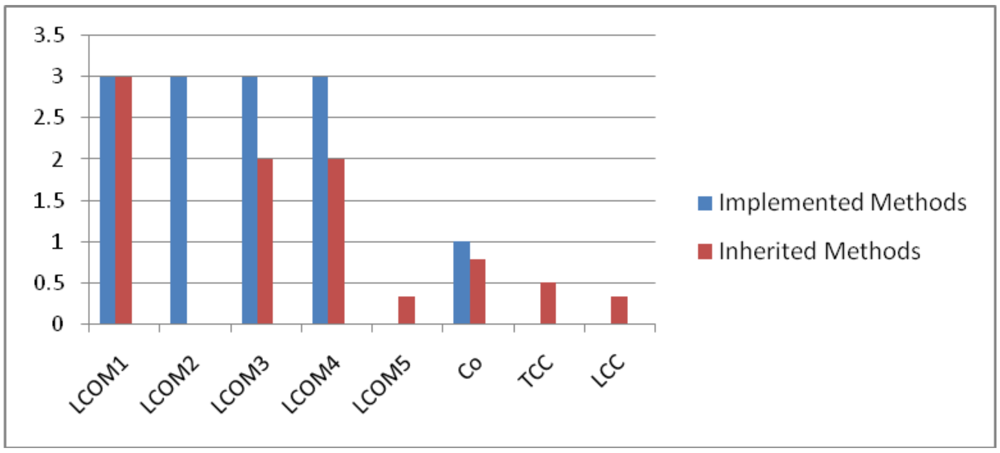

Fig.5: Graphical representation of results of Case1

CASE2:

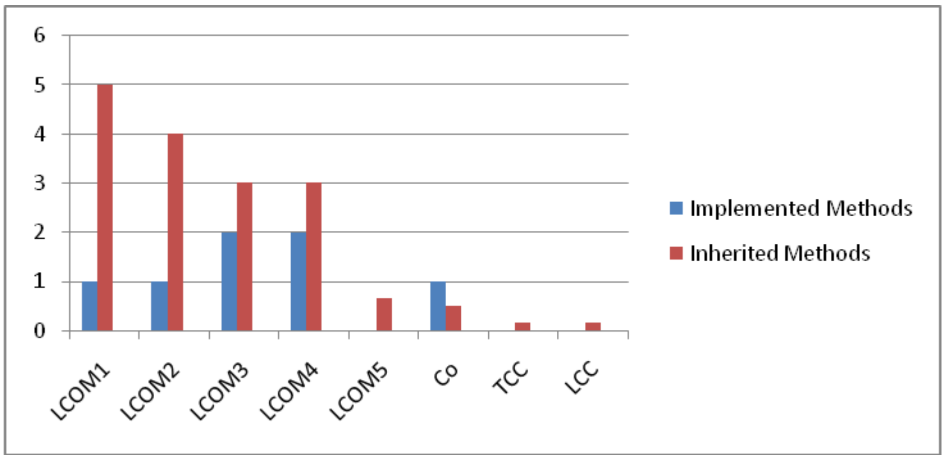

Fig.6: Graphical representation of results of Case2

CASE3:

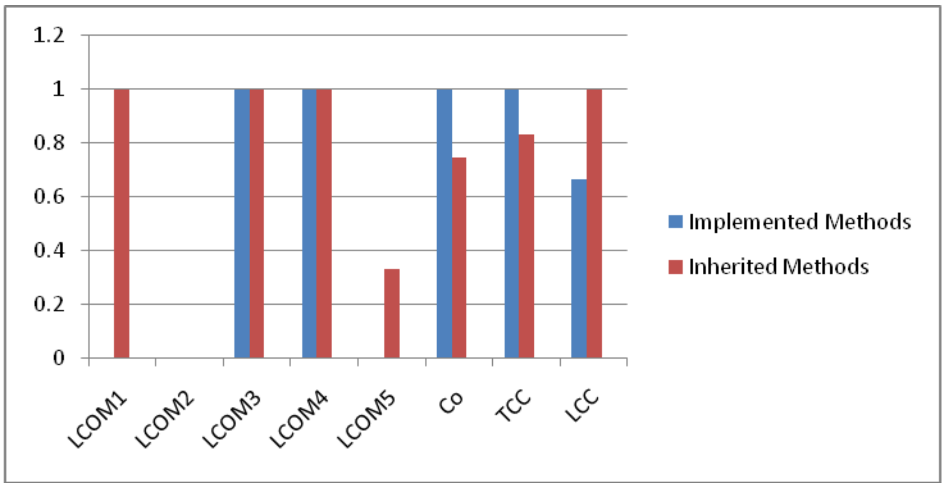

Fig. 7: Graphical representation of results of Case3

IV. CONCLUSION AND FUTURE WORK

By including inherited elements cohesion can be increased or decreased depending upon design structure of super class and sub class. Additional efforts are requires to measure design complexity by including inherited elements. TCC and LCC are used to measure design complexity of software.

This study will help to improve the applicability of existing cohesion metrics to measure the requirement of refactoring the classes. LCC helps to assess reusability of code. There are some aspects related to inheritance that are not discussed in this paper for example: Concept of public, private, protected and internal elements require investigation. Concept of direct or indirect call by instance variable should be considered.

REFERENCES

[1] **Improving Applicability of Cohesion Metrics Including Inheritance**", Jaspreet Kaur, Rupinder Kaur, International Journal of Application or Innovation in Engineering & Management (IJAIEM)

[2] **Empirical Study of Object-Oriented Metrics**", K.K.Aggarwal, Yogesh Singh, Arvinder Kaur, Ruchika Malhotra, JOURNAL OF OBJECT TECHNOLOGY

[3] **Observations on Lack of Cohesion Metrics**", Sami Mäkelä and Ville Leppänen, International Conference on Computer Systems and Technologies - CompSysTech'06

[4] **Comparison of Various Lacks of Cohesion Metrics**", Taranjeet Kaur, Rupinder Kaur, International Journal of Engineering and Advanced Technology (IJEAT)

[5] **A Proposal for Normalized Lack of Cohesion in Method (LCOM) Metric Using Field Experiment**", Ezekiel Okike, IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 4, No 5, July 2010

[6] **An Object-Oriented High-Level Design-Based Class Cohesion Metric**", Jehad Al Dallal,

[7] **Using Object-Oriented Metrics for Automatic Design Flaws Detection in Large Scale Systems**", Dipl.-Ing. Radu Marinescu

[8] **Mathematical Validation of Object-Oriented Class Cohesion Metrics**", Jehad Al Dallal, INTERNATIONAL JOURNAL OF COMPUTERS Issue 2, Volume 4, 2010

[9] **Modeling Class Cohesion as Mixtures of Latent Topics**"

[10] **Design ACM TOSEM, April 2006 Steve Counsell, Stephen Swift, and Jason**"

[11] http://www.aivosto.com/project/help/pm-oo-cohesion.html

[12] **The Impact of Inheritance on the Internal Quality Attributes of Java Classes**", Jehad Al Dallal Department of Information Science- Kuwait University

[13] **Identification of nominated classes for software refactoring using Object –Oriented Cohesion metrics**", Safwat M. Ibrahim, Sameh A. Aalem, Manal A. Ismail and Mohamed Eladawy

[14] **Complexity Identification of Inheritance and Interface based on Cohesion and Coupling Metrics to Increase Reusability**", Maya Yadav, Jasvinder Pal Singh, Pradeep Baniya, International Journal of Computer Applications (0975 – 8887)