INHERITANCE METRICS FOR OBJECT-ORIENTED DESIGN

Kumar Rajnish[1], Arbind Kumar Choudhary[2] and Anand Mohan Agrawal[3]

[1]Department of CS & E, BITIC, Muscat, Sultanate of Oman.
kumarrajnish089@gmail.com
[2] Department of CS & E, BITIC, Muscat, Sultanate of Oman.
arbind@waljat.net
[3]Professor of Management, Muscat, Sultanate of Oman.
amagrawal2000@gmail.com

ABSTRACT

The inheritance metrics give us information about the inheritance tree of the system. Inheritance is a key feature of the Object-Oriented (OO) paradigm. This mechanism supports the class hierarchy design and captures the IS-A relationship between a super class and its subclass. Several OO inheritance metrics have been proposed and their reviews are available in the literature. Among the various measurements we focus on the metrics of class inheritance hierarchies. In this paper we consider the inheritance metrics of F.T. Sheldon et al (2002) and Henderson Seller's (1996) for comparison with proposed inheritance metric suites. In doing so, an attempt has been made to define empirical relationship between the proposed inheritance metric suites with considered existing inheritance metrics and the focus was on which how the inheritance metric suites were correlated with the existing ones. Data for several C++ classes has been collected from various sources.

KEYWORDS

Object-Oriented, Inheritance Tree, Inheritance Hierarchy, Complexity, Classes, Metrics

1. INTRODUCTION

Software metrics are essential to software engineering for measuring software complexity and quality, estimating cost and project effort to simply name a few. The traditional metrics like function point, software science and cyclomatic complexity have been well used in the procedural paradigm. However, they do not readily apply to aspects of the OO paradigm: class, inheritance, polymorphism, virtual function, etc.

The OO technology forces the growth of OO software metrics [6]. Several such metrics have been proposed and their reviews are available [5] [7] [9-10] [14] [21] [22] [27]. The metrics suite proposed by Chidamber and Kemerer is one of the best-known OO metrics [12-13]. Various researchers have conducted empirical studies to validate the OO metrics for their effects upon program attributes and quality factors such as development or maintenance effort [8] [24]. Alshayeb and Li predict that OO metrics are effective (at least in some cases) in predicting design efforts [1]. Chae, Kwon and Bae investigated the effects of dependence variables on cohesion metrics for OO programs [11]. Several other researchers have validated OO metrics for effects of class size and with the change proneness of classes [2] [16-17]. Li [26] theoretically validated Chidamber and Kemerer metrics using a metric evaluation framework proposed by Kitchenham et al [25] and discovered some of the deficiencies of Chidamber and Kemerer metrics in the evaluation process and proposed a new suite of OO metrics that overcome these deficiencies. Rajnish and Bhattacherjee have studied the effect of class complexity (measured in terms of lines of codes, distinct variables names and function) on

DOI : 10.5121/ijcsit.2010.2602

development time of various C++ classes [4] [32] [37]. Rajnish and Bhattacherjee have also studied on cohesion metrics for OO programs on various C++ and Java classes by accessing a common variable by a pair of methods in a class as in [33] [28] [34] [35].

Among the various measurements, we focus on the metrics of class inheritance hierarchies. The inheritance metrics give us information about the inheritance tree of the system. Inheritance is a key feature of the OO paradigm. This mechanism supports the class hierarchy design and captures the IS-A relationship between a super class and its subclass. Class design is central to the development of OO systems. Because class design deals with functional requirements of the system, it is the highest priority in OOD (Object-Oriented Design). The use of inheritance is claimed to reduce the amount of software maintenance necessary and ease the burden of testing [13] and the reuse of software through inheritance is claimed to produce more maintainable, understandable and reliable software [3]. However, industrial adoption of academic metrics research has been slow due to, for example, a lack of perceived need. The results of such research are not typically applied to industrial software [19], which makes validation a daunting and difficult task. For example, the experimental research of Harrison et al. [20] indicates that a system not using inheritance is better for understandability or maintainability than a system with inheritance. However, Daly's experiment [15] indicates that a system with three levels of inheritance is easier to modify than a system with no inheritance. Research has also been conducted regarding class inheritance metrics by Rajnish and Bhattacherjee as in [30] [31] [36] [38] [39] [43].

However, it is agreed that the deeper the inheritance hierarchy, the better the reusability of classes, making it harder to maintain the system. The designers may tend to keep the inheritance hierarchies shallow, discarding reusability through inheritance for simplicity of understanding [13]. So it is necessary to measure the complexity of the inheritance hierarchy to resolve differences between the depth and shallowness of it.

In this paper we consider the inheritance metrics of F.T. Sheldon et al [40] and Henderson Seller's [41] for comparison with proposed inheritance metric suites. In doing so, an attempt has been made to define empirical relationship between the proposed inheritance metric suites with the existing considered inheritance metrics and the focus was on which how the inheritance metric suites were correlated with the existing one. The rest of the paper is organized as follows. Section 2 presents the brief overview of the inheritance metrics of F.T. Sheldon et al [40] and Henderson Seller's [41]. Section 3 presents the proposed inheritance metric suites and examples for illustration. Section 4 presents the results based on collected data [23] and Tools used [18] [42]. Section 5 presents the discussion and Section 6 presents the conclusion and future scope respectively.

2. INHERITANCE METRICS OF F. T. SHELDON AND HENDERSON SELLER

2.1. Average Degree of Understandability (AU) Metric of F. T. Sheldon et al

As F.T. Sheldon et al defines *understandability*, the ease of understanding a program structure or a class inheritance structure [40].

To calculate AU, first defining the degree of understandability (denoted by U) of a class as follows:

U of class C_i = PRED (C_i) +1

Where C_i is i th class.

PRED (C_i): the total number of predecessors of class *i*

Next, the total degree of understandability (TU) of a class inheritance tree is defined as follows:

TU of a class inheritance = $\sum_{i=1}^{t}(PRED(C_i)+1)$

Where *t* is the total number of classes in the class inheritance tree.

Finally, the average degree of understandability (AU) of a class inheritance tree is as follows:

AU of a class inheritance = $\sum_{i=1}^{t}(PRED(C_i)+1)$ / t

Consider the class inheritance tree given in Figure 1, AU is calculated as follows:

U (A) = PRED (A) + 1= 0+1 =1; similarly U (B) = 2; U(C) =2; U (D) =3; U (E) =5;

TU=1+2+2+3+5=13; AU= 13/5=2.6;

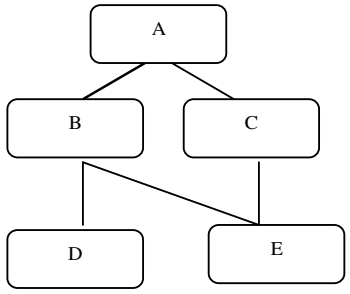

Figure 1. A Class inheritance tree

This value of AU represents the degree of understandability for the class inheritance tree as mentioned in Figure 1. An important point to emphasize here is that a lower value of AU highlights better understandability.

2.2. Average Degree of Modifiability (AM) Metric of F. T. Sheldon et al

As F.T. Sheldon et al defines *modifiability*, the ease with which a change or changes can be made to a program structure or a class inheritance structure [40].

To calculate AM, first defining the degree of modifiability (denoted by M) of a class as follows:

M of a class C_i = U (C_i) + SUCC (C_i) /2

Where C_i is *i* th class.

SUCC (C_i): the total number of successors of class *i*

Next, the total degree of modifiability (TM) of a class inheritance tree is defined as follows:

TM of class inheritance= TU + $\sum_{i=1}^{t}(SUCC(C_i)/2)$

Where *t* is the total number of classes in the class inheritance tree.

Finally, the average degree of modifiability (AM) of a class inheritance tree is as follows:

AM of the class inheritance tree is

AU + $(\sum_{i=1}^{t}(SUCC(C_i)/2))/t$

For the example in Figure 1, AM is calculated as follows:

M (A) = U (A) + SUCC (A)/2= 1+4/2=1+2=3

M (B) =U (B) + SUCC (B)/2=2+2/2=2+1=3

M (C) = U (C) + SUCC (C)/2= 2 + 1/2 = 2+0.5=2.5

M (D) = U (D) + SUCC (D)/2= 3+ 0 = 3

M (E) = U (E) + SUCC (E)/2= 5+0=5

TM = 3 + 3 + 2.5 + 3 + 5 =16.5

AM = 16.5/5 = 3.3

This value of AM represents the degree of modifiability of the class inheritance tree in Figure 1. A lower value of AM represents a better index for modifiability.

2.3. Henderson-Seller's Average Inheritance Depth (AID)

The AID of a class is calculated by [41] as:

AID = ($\sum$ depth of each class) / number of classes

For the example in Figure 1, AID is calculated as follows:

depth of class A=1; depth of class B=2; depth of class C=2; depth of class D=3; depth of class E=3;

AID = 11/5 = 2.5.

3. PROPOSED INHERITANCE METRICS

In this section inheritance metrics for OO design has been proposed which are as follows:

3.1. Derive Base Ratio Metric (DBRM)

DBRM is the ratio of the total derived classes to the total base classes in the class inheritance tree. DBRM is calculated as follows:

$$\text{DBRM} = \left(\sum_{i=0}^{N} TD(C_i)\right) / \left(\sum_{i=0}^{N} TB(C_i)\right)$$

Where $\sum_{i=0}^{N} TD(C_i)$: total number of derived classes in the class inheritance tree.

$\sum_{i=0}^{N} TB(C_i)$: total number of base classes in the class inheritance tree.

N: total number of classes in the class inheritance tree.

Assumptions.

• DBRM measures how many base classes are presents in an inheritance tree which directly affect the immediate or non-immediate subclasses.

• DBRM measures how many derived classes are presents in an inheritance tree which directly or indirectly affect the ancestor classes. More derived classes more reuse, since inheritance is a form of reuse and more mental exercise is required to design and code a class in an inheritance tree.

• In class inheritance tree where DBRM=1.00 will imply a simple case of single inheritance (one base class and one derive class) and multilevel inheritance.

3.2. Average Number of Direct Child (ANDC) Metric

ANDC metric is the ratio of the total number of immediate child to the total number of classes in the inheritance tree. ANDC metric is calculated as follows:

$$\text{ANDC} = \sum_{i=0}^{N} (TDC(C_i)) / N$$

Where, $\sum_{i=0}^{N} TDC(C_i)$: total number of immediate child in the class inheritance tree.

N: total number of classes in the class inheritance tree.

Assumptions.

• Since ANDC metric is concerned with the total numbers of direct child. This gives an indication of the subordinate classes in the class inheritance tree.

• ANDC metric measures how many immediate subclasses are going to inherit the properties (methods and attributes) of classes.

• ANDC metric gives an idea of the potential influence a class has on the overall design.

3.3. Average Number of Indirect Child (ANIC) Metric

ANIC metric is the ratio of the total number of indirect child to the total number of classes in the inheritance tree. ANIC metric is calculated as follows:

$$\text{ANIC}=\sum_{i=0}^{N}(TIC(C_i))/N$$

Where $\sum_{i=0}^{N}TIC(C_i)$: total number of indirect child in the class inheritance tree.

N: total number of classes in the class inheritance tree.

Assumptions

• Since ANIC metric is concerned with the total numbers of non-immediate classes in the class inheritance tree. This metric gives an indication of how many ancestors' classes potentially affect the subclasses in the class inheritance tree.

• Deeper trees constitute greater design complexity, since more properties of classes are involved.

• The deeper a class is in the tree, the higher the degree of methods inheritance, making it more complex to predict its behavior.

3.4. Examples for Illustration

Consider the example shown above in Figure 1 and another example shown below in Figure 2 to illustrates the existing inheritance metrics and proposed inheritance metric suites.

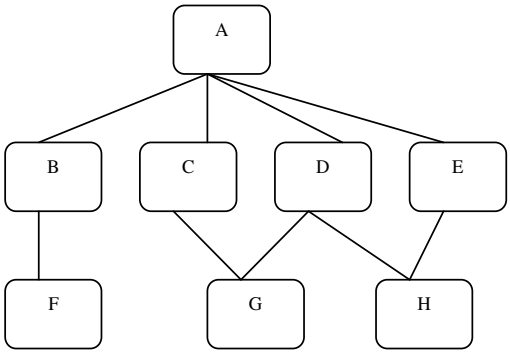

Figure 2. A Class inheritance tree

From *Figure 1*,

AU = 2.6; *AM* = 3.3; *AID* = 1.2;

Calculation of DBRM:

Total base classes = 3 [class A for classes B, C and Classes B, C for classes D and E]

Total derived classes = 4 [Classes B, C, D, and E]

DBRM = 4/3 = 1.334;

Calculation of ANDC:

TDC (A) = 2; TDC (B) = 2; TDC (C) = 1; TDC (D) =0;

TDC (E) = 0;

ANDC = 5/5 =1;

Calculation of ANIC:

TIC (A) = 2; TIC (B) = TIC (C) = TIC (D) = TIC (E) =0;

ANIC = 2/5 = 0.4;

Similarly the calculation of the above mentioned metrics for the example of *figure 2*, are given below:

- *Calculation of AU*

U (A) =1; U (B) = U (C) = U (D) = U (E) = 2; U (F) = 3; U (G) = U (H) = 5;

TU = 1 + 8 + 3 + 10 = 22;

AU = TU / 8 = 22 / 8 = 2.75.

- *Calculation of AM*

M (A) = U (A) + (SUCC (A) / 2) = 1 + 7/2= 4.5;

M (B) = U (B) + (SUCC (B) / 2) = 2 + 1/2= 2.5;

M (C) = U (C) + (SUCC (C) / 2) = 2 + 1/2= 2.5;

M (D) = U (D) + (SUCC (D) / 2) = 2 + 2/2= 3;

M (E) = U (E) + (SUCC (E) / 2) = 2 + 1/2= 2.5;

M (F) = U (F) + (SUCC (F) / 2) = 3 + 0= 3;

M (G) = U (G) + (SUCC (G) / 2) = 5 + 0= 5;

M (H) = U (H) + (SUCC (H) / 2) = 5 + 0= 5;

TM= 4.5+2.5+2.5+3+2.5+3+5+5= 28;

AM = TM / 8 = 28 / 8 = 3.5

- *Calculation of AID*

Depth of Class A=1; Depth of Classes B=C=D=E=2

Depth of Class F = G = H = 3

AID= (1 + 8 + 9) / 8 = 18/8 = 2. 25

- *Calculation of DBRM*

Total derived classes in the class inheritance tree= 7

Total base classes in the class inheritance tree= 5

DBRM = 7/5 = 1. 40

- *Calculation of ANDC*

TDC (A) = 4; TDC (B) = 1; TDC (C) = 1; TDC (D) = 2; TDC (E) = 1; TDC (F) = TDC (G) = TDC (H) = 0;

ANDC = 9 / 8 = 1.125

- *Calculation of ANIC*

TIC (A) = 3; TIC (B) = TIC (C) = TIC (D) = TIC (E) = TIC (F) = TIC (G) = TIC (H) = 0;

ANIC = 3 / 8 = 0.375

The entire results of existing and proposed inheritance metrics values for Figure 1 and Figure 2 are shown in Table 1.

Table 1. Inheritance Metrics values for Figure 1 and Figure 2

Metric Values	**AU**	**AM**	**AID**	**DBRM**	**ANDC**	**ANIC**
Figure 1	2.6	3.3	1.2	1.334	1	0.4
Figure 2	2.75	3.5	2.25	1.4	1.125	0.375

From Table 1, following observations have been made which are as follows:

• Since lower value of AU and AM highlights better understandability and modifiability. Through AU and AM metrics it can be predict that Figure 2 highlights better for understandability and better index for modifiability, still Figure 2 is much more complex than Figure 1.

• High value of AID involves greater design complexity due to high depth (sometimes may be same) and more methods, attributes and classes are involved in the class inheritance tree. From Figure 2, it involves more classes than Figure 1 due to high AID.

• DBRM measures the ratio of the total derived classes to the total base classes in the class inheritance tree. High DBRM means more derived classes and base classes are involved in class inheritance tree. From Figure 1 and Figure2 it can be easily analyzed that Figure 2 involves more derived and bases classes than Figure 1 in the inheritance tree.

• Since ANDC is concerned with direct child (immediate) in the inheritance tree. Through this metric it can be easily analyzed how many classes reused the properties of base classes in an inheritance tree at the design stage. From Figure 1 and Figure 2 it can be analyzed that Figure 2 reuse more properties of classes than Figure 1.

• Since ANIC is concerned with the total number of indirect child (non-immediate) in the inheritance tree. Through this metric it can be analyzed that how many subclasses can potentially affect the ancestor classes at the design stage. From Figure 1 and Figure 2 it can be analyzed that Figure 2 can affect more ancestor classes than Figure 1.

Main-body text is to written in fully (left and right) justified 11 pt. Times New Roman font with a 6pt. (paragraph) line spacing following the last line of each paragraph, but a 12pt. (paragraph) line spacing following the last paragraph. Do not indent paragraphs.

4. RESULTS

In this section statistical analysis has been performed based on collected data from [23]. Metric tools were used for generating inheritance metric values as in [18] [42]. Correlation coefficients for different inheritance metrics were calculated for a class inheritance trees and the focus was on which how different inheritance metrics were correlated to each other. The statistical analysis of the data in the tables has been generated with the aid of MATLAB [29].

Table 2. Descriptive Statistics for the various Inheritance Metric values

	AU	AM	AID	DBRM	ANDC	ANIC
Mean	2.0519	2.4724	1.1518	1.3580	0.7401	0.2008
Median	2	2.5	1	1	0.67	0.2
Std.Dev	0.4843	0.5612	0.6957	0.6058	0.2145	0.2236
Max	3	3.3750	2.836	3	1.25	0.714
Min	1.5	1.75	0.34	0.5	0.5	0

Table 3. Correlation Coefficient w. r. t. different inheritance metric values

	AU	AM	AID	DBRM	ANDC	ANIC
AU	1.000	0.8695	0.4109	0.2465	0.7800	0.6643
AM	0.8695	1.0000	0.4100	0.0484	0.8772	0.7104
AID	0.4109	0.4100	1.0000	0.3499	0.2644	0.7059
DBRM	0.2465	0.0484	0.3449	1.0000	0.1494	0.1658
ANDC	0.7800	0.8722	0.2644	0.1494	1.0000	0.3994
ANIC	0.6643	0.7104	0.7059	0.1658	0.3994	1.000

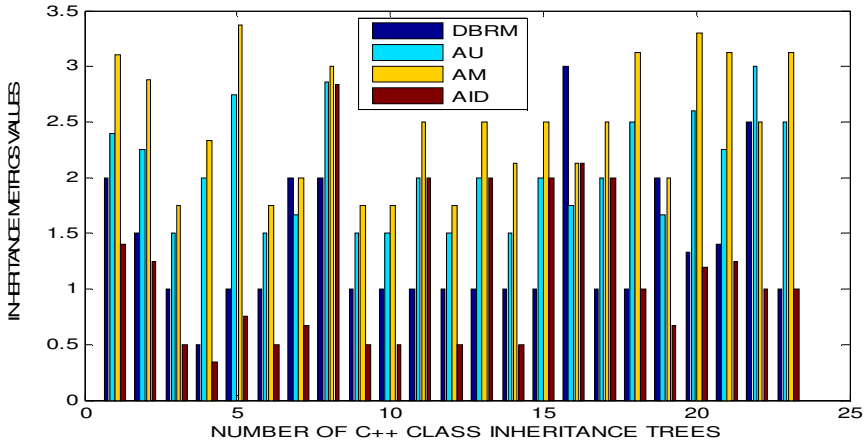

Figure 3. Comparisons of DBRM metric values with AU, AM and AID metrics from various C++ class inheritance trees.

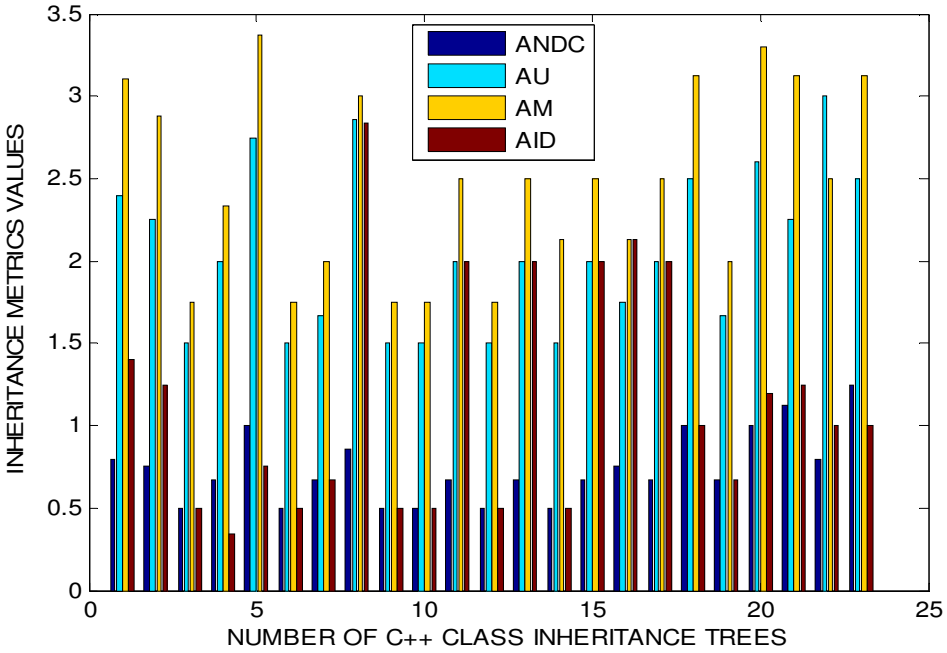

Figure 4. Comparisons of ANDC metric values with AU, AM and AID metrics from various C++ class inheritance trees

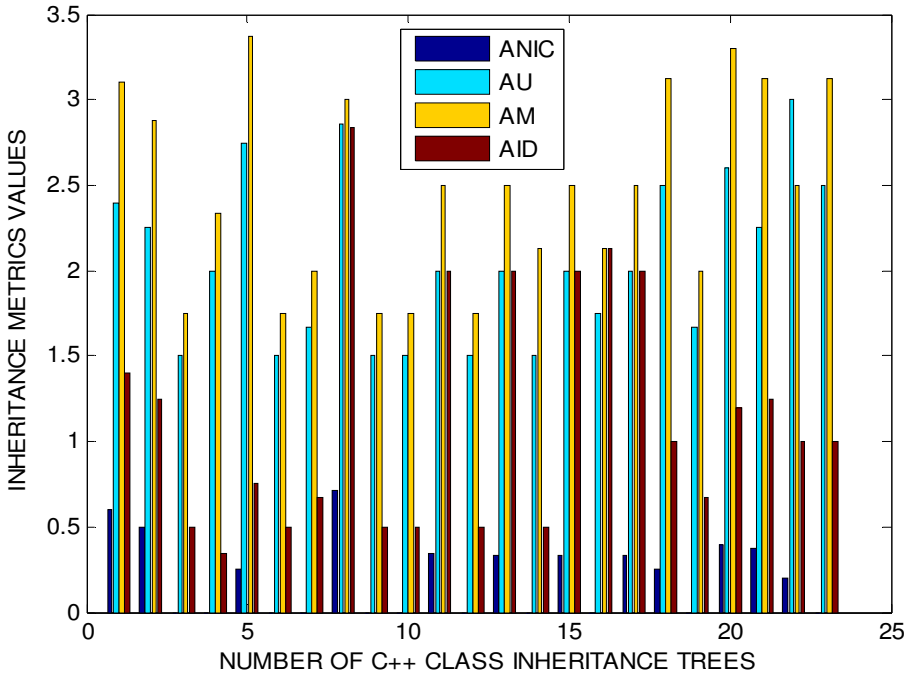

Figure 5. Comparisons of ANIC metric values with AU, AM and AID metrics from various C++ class inheritance trees.

5. DISCUSSION

Certain observations from above mentioned tables and figures done based on collected data from various sources [23].

From Table 2, following observations have been made which are as follows:

• Both AU and AM metric have high mean values as compared to the other metrics in the Table, this suggest more understanding of classes and more modification will be required at any stages of class inheritance trees.

• Mean value (1.1518) of AID represents higher depth. This suggests if classes have higher depth it is very difficult to maintain the sequence of classes (i.e. more classes involved) in the class inheritance tree.

• Mean value (1.3580) of DBRM metric suggest more involvements of base classes and derive classes in the class inheritance trees.

• ANDC metric have less mean value as compared to the other metrics in the Table except ANIC metric, this suggest less number of immediate child is attached to the root of the tree. If mean value of ANDC is high then more number of immediate child are presents in the class inheritance tree and more reuse since inheritance is a form of reuse.

• ANDC and ANIC have low median values this suggests that most classes in an application tend to be close to the root in the inheritance tree.

• By observing the DBRM, ANDC ANIC, AID, AU and AM metric for classes in an application one can determine whether the design is *top heavy* (too many classes near the root) or *bottom heavy* (many classes are near the bottom of the tree).

• By observing the above mentioned metrics values most of the classes appears to be top heavy, this suggest that one may not take the advantage of reuse of methods and attributes through inheritance.

• Another interesting aspect is that maximum value of AU, AM, AID, DBRM, ANDC and ANIC metric is rather small (3.3750 for AM and all are less than AU). One possible explanation is that one can tend to keep that number of levels of abstraction to a manageable number in order to facilitate comprehensibility of the overall architecture of the system. Another possible explanation is that reusability through inheritance for simplicity of understanding.

From Table 3 it is observed that ANDC and ANIC correlates very well with the AU, AM and AID metrics. In column 2 ANDC (correlation: 0.8722 highest in the column) correlates very well with the AU, AM and AID metrics. In column 3 ANIC (correlation: 0.7059 highest in the column) correlates very well with the AU, AM and AID metrics. DBRM metric does not correlate well with all metrics (existing as well as proposed one). In all the columns it has the lowest correlation than existing metrics (AU, AM, AID) and proposed metrics (ANIC and ANDC). It is because DBRM metric may require some extra or additional parameters for to be good correlation with the existing as well as proposed metrics.

6. CONCLUSION AND FUTURE SCOPE

In this paper an attempt has been made to present the different inheritance metrics for measuring the class inheritance trees. In the work presented here, the goal was to find the effect of different

class inheritance metrics values at each level at the design stage. The approach taken was empirical. The OO languages used in the data set was C++. As seen from Table 3 ANIC and ANDC has very well correlation with the existing metrics. This firmly believe us that both ANIC and ANDC metrics is a good measure for class inheritance trees at the design stage.

It must be mentioned that the programs used for the study were very small compared to large industry system. Therefore in terms of future scope, we plan to study some fundamental issues,

(1) some more program parameter has to be incorporated to our proposed DBRM (Derive Base Ratio Metric) because its correlation is not very well with the existing as well as our proposed ANIC and ANDC metrics [see in Table 3].

(2) Further characteristics of classes need to be studied to establish an empirical relationship between the different existing class inheritance metrics and proposed one.

(3) Towards further validation with an extended set of classes and further evaluation of proposed metrics will in turn improvement of the quality of classes.

ACKNOWLEDGEMENTS

This research work has been partially funded by UGC [F. No.: 33 – 61/ 2007 (SR)] under financial grants for Major Research Project.

REFERENCES

1. Alshayeb. M and Li. W, "An Empirical Validation of Object-Oriented Metrics in Two Different Iterative Software Processes", *IEEE Trans. on Software Engineering,* 29, 11 (2003) 1043-1049.
2. Arisholm. E, Briand. L. C and Foyen, "A Dynamic coupling measures for Object-Oriented Software", *IEEE Trans. on Software Engineering,* 30, 8 (2004) 491-506.
3. Basili. VR, Briand. L. C and Melo. WL, "A validation of object-oriented design metrics as qualityindicators", Technical report, University of Maryland, Department of Computer Science, 1995; 1-24.
4. Bhattacherjee. V and Rajnish. K, " Class Complexity-A Case Study", *Proceedings of First International Conference on Emerging Application of Information Technology(EAIT-2006),* Kolkata,India,2006, pp. 253-258.
5. Bieman. J. M and Kang. B.K, "Cohesion and Reuse in an Object-Oriented System", in *Proc. Symp. Software Reliability*, (1995) 259-26.
6. Booch.G, Object-*Oriented Design and Application*, Benjamin/Cummings, Mento Park, CA, 1991.
7. Briand. L. C, Daly. J. W and Wust. J. K, "*A Unified Framework for Cohesion Measurement in Object-Oriented Systems*", Empirical Software Eng., 1, 1 (1998), 65-117.
8. Briand. L. C and Wust. J. K,"Modeling Development Effort in Object-Oriented Systems Using Design Properties", *IEEE Trans. on Software Engineering.* , 27, 11(2001), 963-986.
9. Brotoeabreu. F, "The MOOD Metrics Set", in *Proc. ECOOP'95 Workshop Metrics*, 1995.
10. Chae.H.S, Kwon. Y. R and Bae.D. H, "*A Cohesion Measures for Object-Oriented Classes*", Software practice and Experiences, 30, 12 (2000), 1405-1431.
11. Chae. H. S, Kwon. Y. R and Bae. D. H, "Improving Cohesion Metrics for Classes by considering Dependent Instance Variables", *IEEE Trans. on Software Engineering*, 20, 6 (1994), 476-493.
12. Chidamber. S. R and Kemerer. C. F, "Towards a Metric Suite for Object-Oriented Design", in *Proc. Sixth OOPSLA Conf.*, (1991), 197-211.

13. Chidamber. S. R and Kemerer. C. F, "A Metric Suite for Object-Oriented Design*", IEEE Trans. on Software Engineering*, 20, 6(1994), 476-493.

14. Churcher. N. I and Sheppered. M. j, Comments on "A Metric Suite for Object-Oriented Design", *IEEE Trans. on Software Engineering.,*21 (1995), 263-265.

15. Daly. J, Brooks. A, Miller. J, Roper. M, Wood. M, "Evaluation inheritance depth on the maintainability of object-oriented software", *Empirical Software Engineering* 1996; 1(2): 109-132.

16. Emam. K. EL, Benlarbi. S, Goel. N and Rai. S. N, "The Confounding Effect of the Class Size on the Validity of Object-Oriented Metrics", *IEEE Trans. on Software Engineering*, 27, 7(2001), 630-650.

17. Evanco. W. M, Comments on ""The Confounding Effect of the Class Size on the Validity of Object-Oriented Metrics", *IEEE Trans. on Software Engineering*, 29, 7 (2003), 670-672.

18. Website: http://www.CCCC.sourceforge.net and http://www. sourceforge.net/project/CCCC

19. Fenton. NE, Neil. M, "Software metrics: Successes, failures and new directions", *The Journal of Systems and Software* 1999; 47(2-3):149-157.

20. Harrison. R, Counsell. SJ, Nithi. RV, "An evaluation of the MOOD set of object-oriented software metrics". *IEEE Trans. On Software Engineering* 1998; 24(6):491-496.

21. Henderson-Sellers. B and Edwards. J. M, "*Books Two of Object-Oriented Knowledge: The Working Object*", Prentice Hall, Sydney, 1994.

22. Hitz. M, and Montazeri. B, Correspondence, Chidamber and Kemerer's Metrics Suite: "A Measurement Theory Perspective", *IEEE Trans. on Software Engineering*, 22, 4(1996), 267-271.

23. Internal Reports, *Department of Computer Science & engg.* Birla Institute of Technology, Ranchi, India.

24. Kabaili. H, Keller. R. K and Lustman. F, "Cohesion as Changeability Indicator in Object-Oriented System", in *Proc. Fifth European Conf. Software Maintenance and Reengineering*, 2001.

25. Kitchenham. B, Pfleeger. SL and Fenton. NE, "Towards a framework for software measurement validation", *IEEE Trans. On Software Engineering* 1995; 21(12):929-944.

26. Li. W," Another metric suite for object-oriented programming", *The Journal of Systems and Software* 1998; 44(2): 155-162.

27. Lorenz. M, and Kidd. J, "Object-Oriented Software Metrics": A Practical Guide, 1994.

28. Mahanti,. P. K, Rajnish. K and Bhattacherjee. V, "Measuring Class Cohesion: An Empirical Approach", *Proceedings of ISCA 19th International Conference on Computer Applications in Industry and Engineering (CAINE-2006),* November 13-15, Las Vegas, Nevada, USA, pp. 193-198.

29. Pratap. R,"*Getting Started with MATLAB-VI"*, Oxford University Press, 1998.

30. Rajnish. K and Bhattacherjee. V, "Maintenance of Metrics through class Inheritance hierarchy",*proceedings of International conference on Challenges and Opportunities in IT Industry",* PCTE, Ludhiana, 2005, pp.83.

31. Rajnish. K and Bhattacherjee. V," A New Metric for Class Inheritance Hierarchy: An Illustration*", proceedings of National Conference on Emerging Principles and Practices of Computer Science & Information Technology"*, GNDEC, Ludhiana, 2006, pp 321-325.

32. Rajnish. K and Bhattacherjee. V,"Complexity of Class and Development Time: A Study", Journal of Theoretical and Applied Information Technology (JATIT-2K6), Asian Research Publication Network, Vol. 3, No.1, June-Dec-2006, pp. 63-70.

33. Rajnish. K and Bhattacherjee. V, "Cohesion Metric for Object-Oriented Design". *Proceedings of Second National on Innovation in Information and Communication Technology(NCIICT-2006)*, July 7-8, PSG College of Technology, Coimbatore, India, pp. 73-78.

34. Rajnish. K and Bhattacherjee. V, " Class Cohesion and development Time : A Study", *Proceedings of National Conference on Methods and Models in Computing(NCM2C-2006)*, December 18-19 2006, School of Computer and Systems Sciences, JNU, New Delhi, India, pp. 26-34.

35. Rajnish. K and Bhattacherjee. V, "Class Cohesion: An Empirical and Analytical Approach" International Journal of Science and Research (IJSR), Victoria, Australia, Vol.2, No.2, 2007, pp. 53-62.

36. Rajnish. K and Bhattacherjee. V, "Class Inheritance Metrics and development Time: A Study", International Journal Titled as "PCTE Journal of Computer Science", Vol.2, Issue 2, July-Dec-06, pp. 22-28.

37. Rajnish. K and Bhattacherjee. V, "Object-Oriented Class Complexity Metric-A Case Study", *Proceedings of 5th Annual International Conference on Information Science Technology and Management (CISTM)*, 2020 Pennsylvania Ave NW, Ste 904, Washington DC, *published by the Information Institute*, USA, July 16-18, Hyderabad, 2007, pp.36-45 http://www.cistm.org.

38. Rajnish. K and Bhattacherjee. V, "Applicability of Weyuker Property 9 to Object-Oriented Inheritance Tree Metric-A Discussion", *proceedings of IEEE 10th International Conference on Information Technology (ICIT-2007), published by IEEE Computer Society Press*, pp. 234-236, December-2007, http://ICIT2007.home.comcast.net/ , http://www.computer.org.

39. Rajnish. K and Bhattacherjee. V, "Class Inheritance Metrics-An Analytical and Empirical Approach", *INFOCOMP Journal of Computer Science*, Federal University of Lavras, Brazil, Vol. 7, No. 3, 2008, pp. 25-34.

40. Sheldon. T. F, Jerath. K and Chung. H," Metrics for maintainability of class inheritance hierarchies", *Journal of software maintenance and evolution: Research and practice*, Vol. 14, pp. 1-14, 2002.

41. Hender-Seller. B," *Object-Oriented Metrics: measures of Complexity*" Prentice Hall PTR Englewood Cliffs NJ, 1996.

42. Website: http:// www.msquaredTechnologies.com/ ftp / rsmdocs.zip.

43. Rajnish. K, Bhattacherjee. V and Singh. S. K, "An Empirical approach to Inheritance Tree Metrics", Department of MCA, Shri Shankaraacharya college of Information Science and Technology, Bhillai, 2008, pp. 145-150.

Authors

Dr. Kumar Rajnish is a Senior Lecturer in the Department of CS&E at Birla Institute of Technology, International Centre Muscat, Sultanate of Oman. He received his PhD in Engineering from BIT Mesra Ranchi, Jharkhand, India in the year of 2009. His Research area is Object-Oriented Metrics, Object-Oriented Software Engineering, Programming Languages, and Database System.

Mr. Arbind Kumar Choudhary is a Senior Lecturer in the Department of CS&E at Birla Institute of Technology, International Centre Muscat, Sultanate of Oman. His Research area is Object-Oriented Metrics, Software Engineering and Programming Languages.

Dr. Anand Mohan Agrawal is a Professor of Mangement at Birla Institute of Technology Mesra Ranchi, Jharkhand, India. Currently, He is working in Muscat, Oman. His Research Interest is Operation Management, E-Commerce and Information Technology.