An Empirical and Analytical View of New Inheritance Metric for Object-Oriented Design

Kumar Rajnish
Department of IT
Birla Institute of Technology Ranchi-835215, India

Yashbir Singh
Department of CSE
Birla Institute of Technology Ranchi-835215, India

ABSTRACT
Object-Oriented (OO) design is becoming more popular in software development environment and OO design metrics are essential parts of software environment. Inheritance is one of the main features of OO programming paradigm. The inheritance metrics gives information about the inheritance tree of the system. This mechanism supports the class hierarchy design and captures the IS-A relationship between a super class and its subclass. This paper presents a new approach for inheritance metrics CIT (**C**lass **I**nheritance **T**ree) for measuring the inheritance tree. The proposed metric is evaluated against Weyuker's properties (established criteria for validity) and present empirical data from academic projects (developed by experienced PG students) to illustrates the usefulness of new metric. In this paper we also consider the Chidamber and Kemerer (CK) and Li's inheritance metrics for study and presents a comparative study between existing and propose metrics and the focus is on, how propose metric is correlated with the existing ones.

General Terms
Design, Metrics, Measurement.

Keywords
Object-Oriented, Metrics, Inheritance Tree, Classes.

1. INTRODUCTION
It is clear that measurement of any process or product is necessary for its success. Software engineering metrics are units of measurement, which are used to characterized software engineering products, processes and people. If used properly they can allow us to identify and quantify improvement and make meaning estimates.

The recent drive towards OO technology forces the growth of OO metrics [1]. Several such metrics have been proposed and their reviews are available [2-5]. The metrics suite proposed by Chidamber and Kemerer (CK) is one of the best OO metric [6-7]. Alshayeb and Li predict that OO metrics are effective (at least in some cases) in predicting design efforts [8]. Rajnish and Bhattacherjee have also studied on the class inheritance tree which is based on finding the depth of inheritance tree of a class inheritance tree which is based on finding the depth of inheritance tree of a class metric for class inheritance tree in terms of sum of attributes (private, protected and inherited) at each level on various C++ class hierarchies [10-11] [14-16] [20-21].

Among the various measurements, metrics for class inheritance tree is chosen for study. The inheritance metrics gives 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 [7] and the reuse of software through inheritance is claimed to produce more maintainable, understandable and reliable software [22]. 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 [23], which makes validation a daunting and difficult task. For example, the experimental research of Harrison et al. [24] indicates that a system not using inheritance is better for understandability or maintainability than a system with inheritance is easier to modify than system with no inheritance.

However, it is agreed that the deeper the inheritance tree, 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 [7]. So it is necessary to measure the complexity of inheritance hierarchy to resolve differences between the depth and shallowness of it.

This paper presents a new approach of inheritance metrics for measuring the inheritance tree. The proposed metric is evaluated against Weyuker properties (established criteria for validity) [12] and present empirical data from academic projects (developed by experienced PG students) [27] to illustrates the usefulness of new metric. This paper considered Chidamber and Kemerer (CK) and Li's inheritance metrics for study and present a comparative study between existing and propose metrics and the focus is on, how propose metric is correlated with the existing one. The rest of the paper is organized as follows: Section 2 presents the Weyuker's properties. Section 3 presents the brief description of existing OO metrics. Section 4 presents Results (which contains description of proposed metric, Analytical evaluation, empirical validation and their interpretation). Section 5 presents conclusion and future scope respectively.

2. WEYUKER'S PROPERTY
All The basic nine properties proposed by Weyuker [24] are listed below. The notations used are as follows: P, Q, and R denote combination of classes P and Q, μ denotes the chosen metrics, μ (P) denotes the value of the metric for class P, and P≡Q (P is equivalent to Q) means that two class designs, P and Q, provide the same functionality. The definition of combination of two classes is taken here to be same as suggested by [1], i.e., the combination of two classes results in

another class whose properties are union of the properties of the component classes. Also, combination stands for Weyuker's notion of "concatenation".

Property 1. Non-coarseness: Given a class P and a metric μ, another class Q can always be found such that, $\mu(P) \neq \mu(Q)$.

Property 2. Granularity: There are a finite number of cases having same metric value. This property will be met by any metric measured at class level.

Property 3. Non-uniqueness (notion of equivalence): There can exist distinct classes P and Q such that, $\mu(P) = \mu(Q)$.

Property 4. Design details are important: For two class designs, P and Q, which provide the same functionality it does not imply that the metric values for P and Q will be same.

Property 5. Monotonicity: For classes P and Q the following must hold: $\mu(P) \leq \mu(P+Q)$ and $\mu(Q) \leq (P+Q)$ where P+Q imply combination of P and Q.

Property 6. Non-equivalence of interaction: ∃P, ∃Q, ∃R such that, $\mu(P) = \mu(Q)$ does not imply $\mu(P+R) = \mu(Q+R)$.

Property 7. Permutation of elements within the item being measured can change the metric value.

Property 8. When the name of the measured entity changes, the metric should remain unchanged.

Property 9. Interaction increases complexity: ∃P and ∃Q such that: $\mu(P) + \mu(Q) < \mu(P+Q)$.

Weyuker's list the properties has been criticized by some researchers; however, it is widely known formal approach and serves as an important measure to evaluate metrics. In the above list however, property 2 and 8 will trivially satisfied by any metric that is defined for a class. Weyuker's second property "granularity" only requires that there be a finite number of cases having the same metric value. This metric will be met by any metric measured at the class level. Property 8 will also be satisfied by all metrics measured at the class level since they will not be affected by the names of class or the methods and instance variables. Property 7 requires that permutation of program statements can change the metric value. This metric is meaningful in traditional program design where the ordering of if-then-else blocks could alter the program logic and hence the metric. In OOD (Object-Oriented Design) a class is an abstraction of a real world problem and the ordering of the statements within the class will have no effect in eventual execution. Hence, it has been suggested that property 7 is not appropriate for OOD metrics.

Analytical evaluation is required so as to mathematically validate the correctness of a measure as an acceptable metric. For example, Properties 1, 2, and 3 namely Non-Coarseness, Granularity, and Non-Uniqueness are general properties to be satisfied by any metric. By evaluating the metric against any property one can analyze the nature of the metric. For example, property 9 of Weyuker will not normally be satisfied by any metric for which high values are an indicator of bad design measured at the class level. In case it does, this would imply that it is a case of bad composition, and the classes, if combined, need to be restructured. Having analytically evaluated a metric, one can proceed to validate it against data.

Assumptions. Some basic assumptions used in section 4.2 under section 4 have been taken from Chidamber and Kemerer [26] regarding the distribution of methods and instance variables in the discussions for the metric properties.

Assumption 1:

Let X_i= the number of methods in a given class i

Y_i= the number of methods called from a given method i

Z_i= the number of instance variables used by a method i

X_i, Y_i, Z_i are discrete random variables each characterized by some general distribution functions. Further, all the X_i's are independent and identically distributed. The same is true for all Y_i's and Z_i's. This suggests that the number of methods and variables follow a statistical distribution that is not apparent to an observer of the system. Further, that observer cannot predict the variables and methods of one class based on the knowledge of the variables and methods of another class in the system.

Assumption 2: In general, two classes can have a finite number of *"identical"* methods in the sense that a combination of the two classes into one class would result in one class's version of the identical methods becoming redundant. For example, a class *"foo_one"* has a method "draw" that is responsible for drawing an icon on a screen; another class *"foo_two"* also has a "draw" method. Now a designer decides to have a single class *"foo"* and combine the two classes. Instead of having two different "draw" methods the designer can decide to just have one "*draw"* method.

Assumption 3: the inheritance tree is *"full"*, i.e. there is a root, intermediate nodes and leaves. This assumption merely states that an application does not consist only of standalone classes; there is some use of sub classing.

3. EXISTING OBJECT-ORIENTED METRIC FOR STUDY

The brief description of existing OO metrics are presented in Table 1.

Table 1. Existing OO Metrics

OO Metrics	Description
Depth of Inheritance Tree (DIT)	The depth of inheritance tree will be the maximum length from the node to the root of the tree[7]
Number of Children (NOC)	Number of immediate subclasses subordinated to a class in the class inheritance tree is the NOC for that class [7].
Number of Ancestor class (NAC)	The NAC measures the total number of ancestor classes from which a class inherits in the class inheritance tree [9].
Number of Descendent class (NDC)	The NDC measures the total number of descendent classes of a class [9].

4. RESULT

4.1 Class Inheritance Tree (CIT) Metric

CIT is used to measure the class inheritance tree. The primary purpose of this metric is to measure how class is inherited by multiple classes and how class inherits multiple classes at any level in the inheritance tree. CIT is defined as follows:

$$CIT(Ci) = \sum_{i=1}^{L} CIN(Ci) + COUT(Ci)$$

Where Ci is the classes at the i^{th} level in the inheritance tree.

CIN (C_i) = 1 if C_i is inherits multiple classes in the inheritance tree.

0, otherwise.

COUT(C_i) = 1 if C_i is inherited by multiple classes in the inheritance tree.

0, otherwise.

Intuitive Ideas for CIT

- Deeper the class in the class inheritance tree, more possibilities of classes inherits multiple classes and classes inherited by multiple classes, making it difficult to predict the behavior of classes in inheritance tree.
- Deeper inheritance tree constitutes more depth, greater design complexity, since more classes involved.
- Since a class inherits multiple classes, so deeper a particular class in the inheritance tree, greater the possibilities of reuse (since inheritance is a form of reuse).

4.2 Analytical evaluation of CIT against Weyuker's Properties

Since the inheritance tree has a root and leaves. There may be a situation where classes at the leaf node inherit multiple classes and the root node is inherited by multiple classes. Also, since every tree has at least some nodes with siblings; there will always exist at least two classes with same CIT. Suppose class P and Class R be leaves and class Q is the root, therefore there may be a situation where CIT (P) =CIT(R) and CIT (Q) ≠CIT(R) or CIT (P). So, *property 1 (Non-coarseness) and property 3(Non-uniqueness) is satisfied.*

Property 2 (Granularity) is satisfied because there is finite number of cases where CIT of classes may have the same values at any level in the inheritance tree.

Generally design of classes involves choosing what properties the class must inherit in order to perform its operation and also it involves decision on the scope of the methods declared within the class i.e. the sub classing for the class. The number of subclasses is therefore dependent upon the design implementation of the class. In other words CIT is design implementation dependent. Hence, property 4 (Design details are important) is satisfied.

When any two classes (say classes P and Q) are combined in the inheritance tree then there are three possible cases:

Case 1: When class P and class Q are at same level in the inheritance tree.

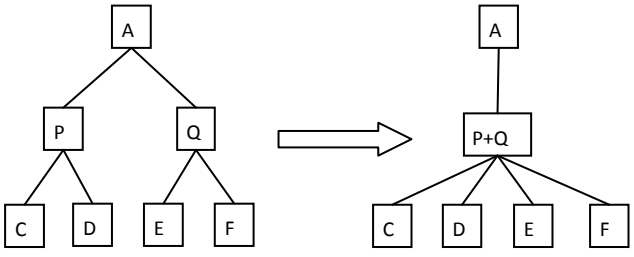

Fig. 1 Before and After combination of Class P and Q

From Fig. 1, CIT (P) =n, CIT (Q) =n, and CIT (P+Q) =n

So, CIT (P) ≤CIT (P+Q) and CIT (Q) ≤CIT (P+Q). *Hence, property 5 is satisfied.*

Case 2: When class Q is not a child of class P and is at the different level in the inheritance tree (see Fig 2).

If class P+Q is located as the immediate ancestor to B and C (P's location in the inheritance tree, the combine class cannot inherit method from X, however if P+Q is located as an immediate child of X (Q's location) the combined class can still inherits method from all the ancestors of P and Q, therefore, P+Q will be located Q's location.

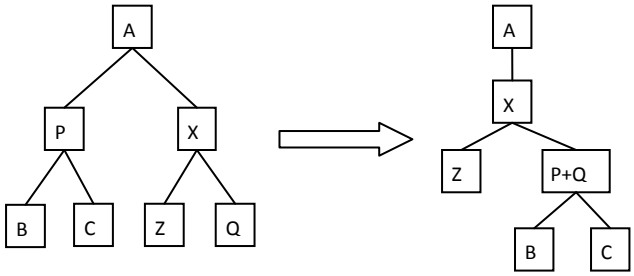

Fig. 2 Before and After combination of Class P and Q

From Fig. 2, CIT (P) =x, CIT (Q) =y and y>x.

CIT (P+Q) =y

CIT (P) ≤CIT (P+Q)

CIT (Q) ≤CIT (P+Q)

Hence, property 5 is satisfied.

Case 3: When one is child of another

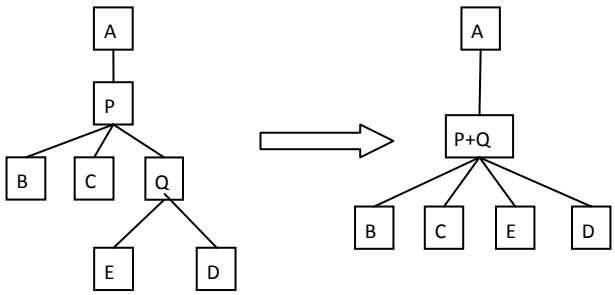

Fig. 3 Before and After the combination of Class P and Q.

From Fig. 3, CIT (P) =n, CIT (Q) =n+1

CIT (P+Q) =n

So, CIT (Q) is not ≤ CIT (P+Q). Hence, *property 5 is not satisfied.*

Suppose there exist three classes P, Q', R where P, Q' are at same level (as siblings) and R is the child of P.

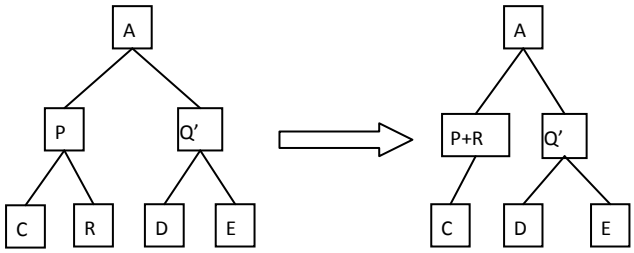

Fig. 4 Before and After combination of class P and R.

From Fig. 4, CIT (P) =n, CIT (P+R) =n, CIT (Q') =n.

So, CIT (P) =CIT (Q') and CIT (P+R) =n.

After combining Q'+R

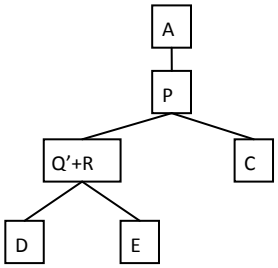

Fig. 5 After combination of class Q' and R.

From Fig 5, CIT (P) =n, CIT (Q'+R) =n+1.

CIT (P) =CIT (Q) and CIT (P+R) ≠CIT (Q'+R)

Hence property 6 is satisfied.

For any two classes P and Q

CIT (P+Q) =CIT (P) or CIT (Q)

Or given any two classes P and Q with X_P and X_Q children respectively, the following relationship hold

CIT (P) =X_P and CIT (Q) =X_Q

CIT (P+Q) =X_P+X_Q-δ

Where δ is the number of common children.

CIT (P+Q) ≤CIT (P) + CIT (Q)

Hence, *property 9 is not satisfied.*

4.3 Discussion

Data Collection. Data has been collected from undergoing academic projects Library System and Hostel Management [27]. These projects are developed by experienced PG students. These are small projects and two versions are developed. Version 1.0 has 20 inheritance classes and in Version 1.1 some of these 20 classes are updated and 10 more classes are added.

Empirical Data. The Bar Diagrams, summary statistics and Correlation coefficients for different existing and propose inheritance metrics for both version (version1.0 and version 1.1) are shown in Fig. 6, Fig. 7, Fig. 8, Fig. 9, Fig. 10, Fig. 11, Fig. 12, Fig. 13, Fig. 14, Fig. 15 and Table 2, Table 3, Table 4, and Table 5.

Table 2. Summary Statistics for data set version 1.0

OO Metric	Min	Max	Mean
DIT	0	3	1.2105
NOC	0	5	0.8596
NAC	0	4	1.2544
NDC	0	13	1.2895
CIT	0	1	0.3333

Table 3. Summary Statistics for data set version 1.1

OO Metric	Min	Max	Mean
DIT	0	8	2.4401
NOC	0	10	0.9499
NAC	0	8	2.5209
NDC	0	47	2.4847
CIT	0	2	0.2758

Table 4. Correlation Coefficient for the data set version1.0

Correlation coefficient	DIT	NOC	NAC	NDC	CIT
DIT	1.0000	-0.685	**0.959**	-0.648	-0.489
NOC	-0.685	1.0000	-0.670	**0.819**	**0.750**
NAC	**0.959**	-0.670	1.000	-0.632	-0.406
NDC	-0.648	**0.819**	-0.632	1.000	**0.644**
CIT	-0.489	**0.750**	-0.406	**0.644**	1.000

Table 5. Correlation Coefficient for the data set version1.1

Correlation coefficient	DIT	NOC	NAC	NDC	CIT
DIT	1.0000	-0.378	**0.966**	-0.466	-0.390
NOC	-0.378	1.0000	-0.382	**0.685**	**0.700**
NAC	**0.966**	-0.382	1.000	-0.438	-0.366
NDC	-0.446	**0.685**	-0.438	1.000	**0.452**
CIT	-0.390	**0.700**	-0.366	**0.452**	1.000

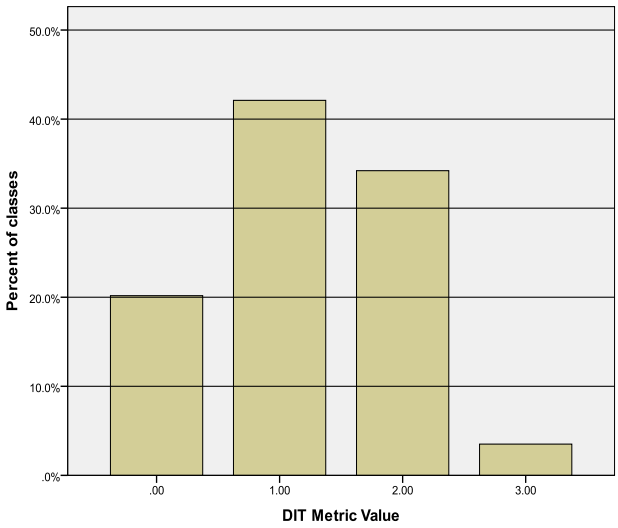

Fig. 6 Bar chart for Version 1.0 for DIT Metric

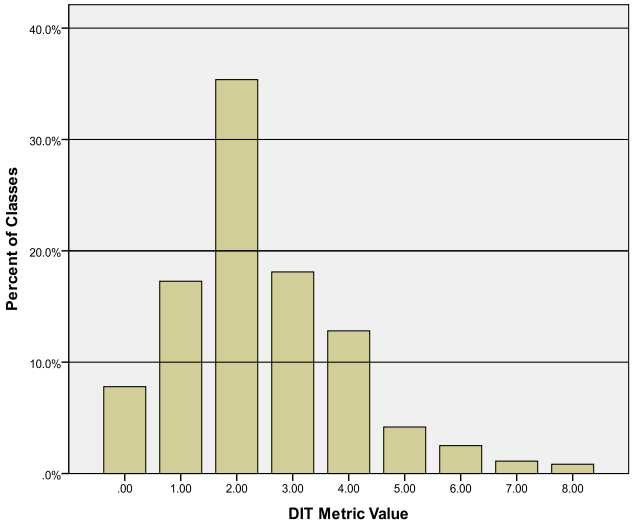

Fig. 7 Bar chart for Version 1.1 for DIT Metric

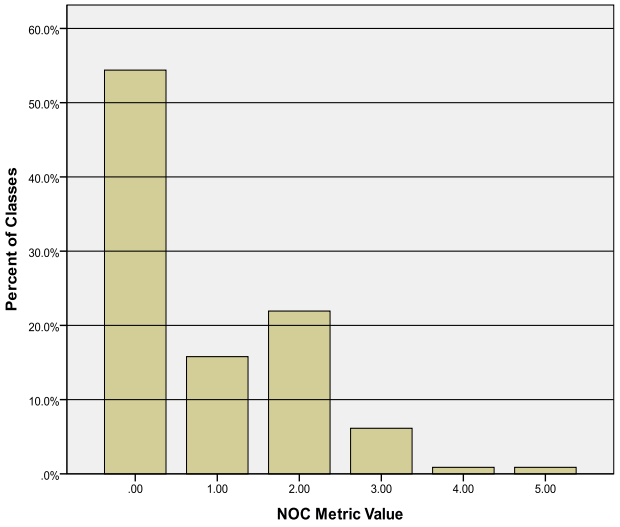

Fig. 8 Bar chart for Version 1.0 for NOC Metric

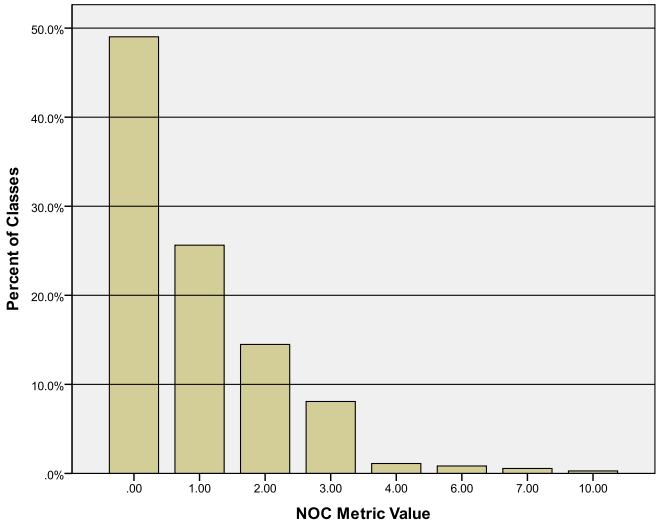

Fig. 9 Bar chart for Version 1.1 for NOC Metric

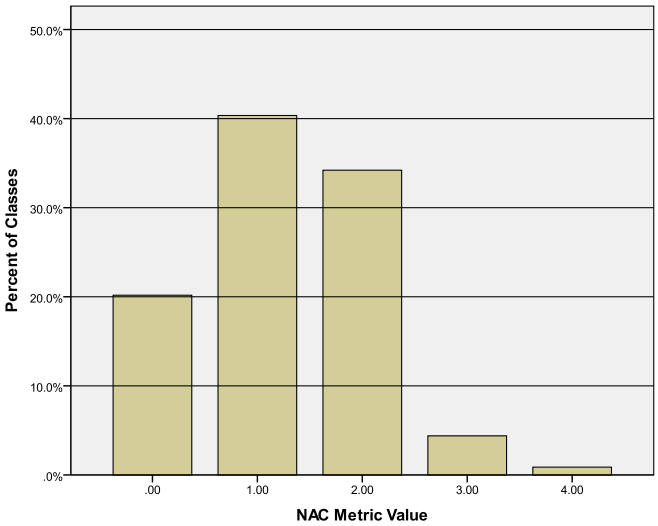

Fig. 10 Bar chart for Version 1.0 for NAC Metric

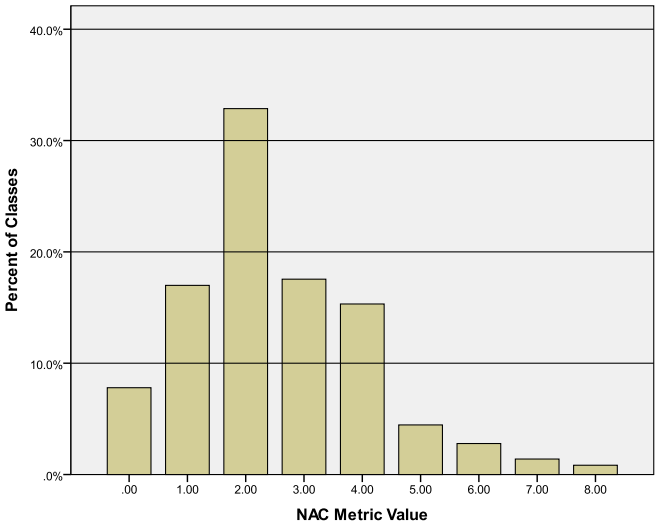

Fig. 11 Bar chart for Version 1.1 for NAC Metric

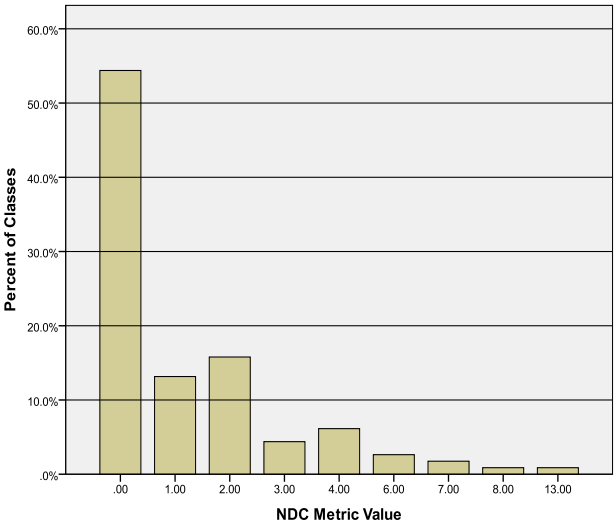

Fig. 12 Bar chart for Version 1.0 for NDC Metric

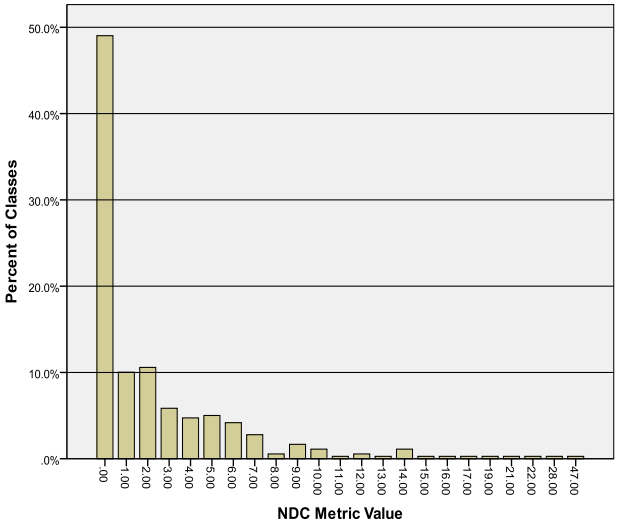

Fig.13 Bar chart for Version 1.1 for NDC Metric

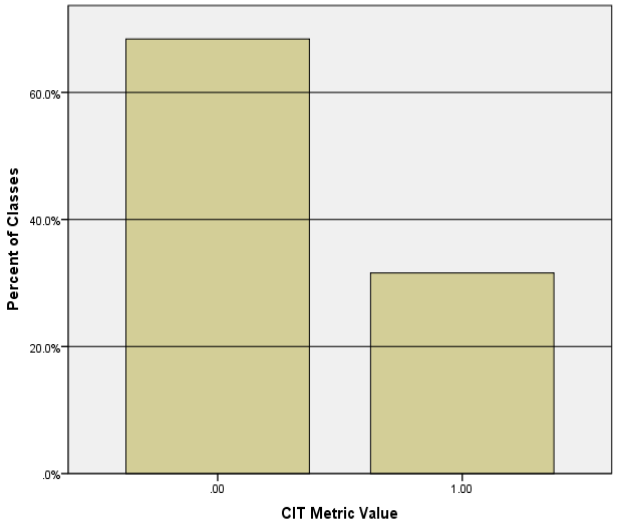

Fig. 14 Bar chart for Version 1.0 for CIT Metric

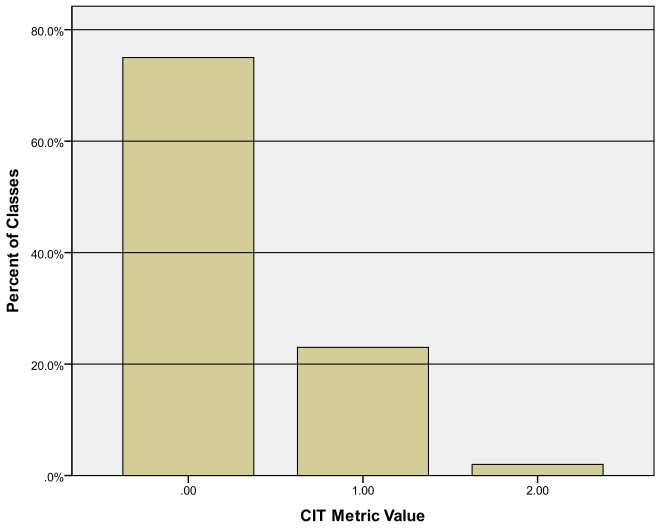

Fig. 15 Bar chart for Version 1.1 for CIT Metric

4.4 Interpretation of Result

From Table 2 and Table 3, it is observed that in both version (version 1.0 and version1.1) have low value for DIT, NOC, NAC, NDC, and CIT metric. This suggests that most of the classes in an application tend to close the root in the inheritance tree. By observing the DIT, NOC, and NAC metric value in both versions, a designer can determine whether the design is too many classes near the root or many classes near the bottom of the inheritance tree. By observing CIT metric value whose maximum value is 2 in version 1.1 and also has less mean value *0.2758* as compare to version 1.0 whose maximum value is 1 with mean value *0.3333*. This suggests that, designer is not taking the advantage of inheritance classes in the inheritance tree.

By observing NDC metric value whose maximum value is 13 in version 1.0 and 47 in version 1.1. This suggests that designer tends to keep the number of levels of abstraction to a manageable number in order to facilitate comprehensibility in the overall architecture of the system. Designers may be for shaking reusability through inheritance for simplicity of understanding.

From Table 4 and Table 5, it is observed that CIT is correlated very well with NOC and NDC in both versions (version 1.0 and version 1.1) especially CIT has a highest correlation with NOC in version 1.1 **(0.700)**. The negative correlation of CIT with DIT and NAC in both versions because of may be less reuse and most of the classes are top heavy (too many classes are near the root of the tree).

From Fig 14 and Fig 15, a Bar Chart for CIT. It is observed that 65% of classes whose CIT is 0 and 35% of classes whose CIT is 1 in version 1.0 (this indicates classes are less inherited by subclasses and classes inherits less classes in the inheritance tree) and in version 1.1, 75% of classes whose CIT is 0, 23% of classes whose CIT is 1 and 2% with CIT 2. This suggests that designer is not taken the advantage of inherited classes in the inheritance tree with CIT 0. It is observed that both versions are top heavy (many classes are used near the root).

5. CONCLUSION AND FUTURE WORK

In this paper an attempt has been made to define a new metric for inheritance tree Class Inherited Tree (CIT) for measuring the inheritance tree of the system which is based on how the class is inherited by multiple classes and how class inherits multiple classes in the inheritance tree.

As seen from Table 4 and Table 5 it is observed that CIT is correlated very well with NOC and NDC metric in both versions (version 1.0 and version 1.1) especially with NOC, CIT has highest correlation. Through CIT one can easily find the inherited classes (both super and subclasses) in the inheritance tree.

As seen from Fig. 14 and Fig. 15 it is observed that in both versions CIT has very high percentages of values 0 and 1, this indicates a less reuse of inheritance from super class to subclasses and vice versa. Since CIT may be considered as a good measure for the inheritance tree because in some situation if designer will get most percentages of classes with CIT is 2, that indicates more classes are inherited from super class to subclasses and vice versa hence, reusability is high.

The future scope includes the following fundamental issues:

(1) The validation of proposed metric is done on small data sets, so, in the next stage we perform validation of CIT with all other existing metrics on large systems which in turn to

improve the quality of classes.

(2) Try to find the impact of CIT on inheritance tree on different versions of data sets.

ACKNOWLEDGEMENT

Our sincere thanks to Birla Institute of Technology, Mesra, Ranchi, India for encouraging us to do this work.

6. REFERENCES

[1] G.Booch, "object-oriented Design and Application", Benjamin/Cummings, Mento Park, CA, 1991.

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

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

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

[5] M.Lorenz and J. Kidd, "*Object-Oriented Software Metrics*": A Practical Guide, 1994.

[6] S. R. Chidamber and C. F. Kemerer, "Towards a Metric Suite for Object-Oriented Design", in *Proc. Sixth OOPSLA Conf.,* (1991), pp.197-211.

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

[8] M. Alshayeb and W. Li, "An Empirical Validation of Object-Oriented Metrics in Two Different Iterative Software Processes", *IEEE Trans. on Software Engineering*, 29, 11 (2003), pp.1043-1049.

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

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

[11] K. Rajnish and V. Bhattacherjee," 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.

[12] E.J.Weyuker. "Evaluating Software Complexity Measures*", IEEE Trans. on Software Engineering*, 14, 1998, 1357-1365.

[13] P. K. Mahanti, K. Rajnish and V. Bhattacherjee, "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.

[14] K. Rajnish and V. Bhattacherjee, "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.

[15] K. Rajnish and V. Bhattacherjee, "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

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

[17] G. Roy, "On the Applicability of Weyuker Property Nine to Object-Oriented Structural Inheritance Complexity Metrics, M. Tech. Minor Project Report, *Faculty of Eng,. Dayalbagh Educational Inst.,* Agra.

[18] Gurusaran and G.roy, "On the applicability of Weyuker Property Nine to Object- Oriented Structural Inheritance Complexity Metrics, *IEEE Transaction on Software Engineering*, Vol.27, no.4, 2001, 361-364.

[19] L. Zhang and D. Xie, "Comments on „On the applicability of Weyuker Property Nine to Object-Oriented Structural Inheritance Complexity Metrics, *IEEE Transaction on Software Engineering*, Vol.28, no.5, 526-527.

[20] K. Rajnish, V. Bhattacherjee and S. K. Singh, "An Empirical Approach to Inheritance Tree Metric", *proceedings of National Level Technical Conf. (Techno Vision-2007)*, Sri Shankaracharya college of Engineering and Technology, Department of MCA, Bhillai, 2007, pp. 145-150.

[21] K. Rajnish, A. K. Choudhary, A. M. Agrawal, "Inheritance Metrics for Object-Oriented Design", *IJCSIT*, Vol. 2 No.6, December 2010, pp.13-26.

[22] Basili. VR, Briand. L. C and Melo. WL, "A validation of object-oriented design metrics as quality indicators", Technical report, University of Maryland, Department of Computer Science, 1995; 1-24.

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

[24] 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.

[25] 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.

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

[27] Internal reports, Department of Computer Science & Engineering, Birla Institute of Technology, Ranchi.