Class Inheritance Metrics-An Analytical and Empirical Approach

KUMAR RAJNISH [1], VANDANA BHATTACHERJEE[2]
[1]Department of Computer Science & Engineering, Birla Institute of Technology, Ranchi-01, India
kumar_rajnish_in@yahoo.com
[2]Department of Computer Science & Engineering, Birla Institute of Technology, Ranchi-01, India
vbhattacharya@bitmesra.ac.in

Abstract-Inheritance is a powerful mechanism in Object-Oriented (OO) programming. This mechanism supports the class hierarchy design and captures the IS-A relationship between a super class and its subclass. Several OO metrics have been proposed and their reviews are available in the literature. Among the various measurements of OO characteristics, this paper focuses on the metrics of class inheritance hierarchies. In this paper first a class inheritance metric DITC (Depth of Inheritance Tree of a Class) metric based on finding the depth of inheritance tree of a class (DITC) metric for class inheritance hierarchy in terms of sum of the attributes (private, protected, public and inherited) and methods (private, protected, public and inherited) at each level is proposed, then an analytical evaluation of DITC metric against Weyuker's axioms [18] is given in discussion part and then attempt has been made to define an empirical relation between development time with respect to its dependence on classes in class inheritance hierarchy at each level. Attempt has also been made to analyze the various dependencies of development time of class in class inheritance hierarchy at each level upon its different class inheritance metric values. Data for several class inheritance hierarchies has been collected from various resources [23].

Keywords- Object-Oriented Design, Classes, Class Inheritance Hierarchy, Cohesion, Object-Oriented Metrics, Class Inheritance Metrics.

(Received June 08, 2007 / Accepted September 13, 2007)

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 characterize software engineering products, processes and people. If used properly they can allow us to identify and quantify improvement and make meaningful estimates.

The recent drive towards Object-Oriented (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 C&K (Chidamber & 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 lease 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 C&K metrics using a metric evaluation framework proposed by Kitchenham et al [25] and discovered some of the deficiencies of C&K 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 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. 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). Inheritance is a key feature of the OO paradigm. 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 in [30] [31] [36] [38]. 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 propose a new metric for the class inheritance hierarchy.

In this paper first overview of Chidamber and Kemerer metrics [13] for class inheritance hierarchy is discussed, and then a proposal for a new class inheritance metric is made. The paper is organized as follows–Section 2 lists out Weyuker's nine properties and why analytical evaluation required. Section 3 provides an overview of C&K inheritance metrics. Section 4 presents a proposed metric on inheritance. Section 5 presents a statistical analysis that how closely the DITC metric of a class in class inheritance hierarchy at each level were correlated to the development time of various C++ classes in the class inheritance hierarchy. Section 6 presents the discussion and section 7 presents the conclusion and future scope.

2. Weyuker's Properties

The basic nine properties proposed by Weyuker's [18] are listed below. The notations used are as follows: P, Q, and R denote classes, P+Q denotes 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 the same as suggested by [1], i.e., the combination of two classes results in another class whose properties (methods and instance variables) are the 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 is a finite number of cases having the same metric value. This property will be met by any metric measured at the 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 the same.

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

Property 6. Non-equivalence of interaction:

$\exists$ P, $\exists$ Q, $\exists$ R such that $\mu(P) = \mu(Q)$ does not imply that $\mu(P+R) = \mu(Q+R)$.

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:

$\exists$ P and $\exists$ Q such that:

$\mu(P) + \mu(Q) < \mu(P + Q)$

Weyuker's list of properties has been criticized by some researchers; however, it is a widely known formal approach and serves as an important measure to evaluate metrics. In the above list however, properties 2 and 8 will be 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 Object-Oriented Design (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 6.1 under section 6 have been taken from Chidamber and Kemerer [13] regarding the distribution of methods and instance variables in the discussions for each of 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_is are independent and identically distributed. The same is true for all the Y_is, and Z_is. 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 combines 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 stand alone classes; there is some use of sub classing.

3. Chidamber and Kemerer Inheritance Metrics

3.1 DIT Metric

Chidamber and Kemerer proposed the Depth Of Inheritance of a class is the DIT metric for the class [13]. In cases involving multiple inheritance, the DIT will be the maximum length from the node to the root of the tree. The DIT metric is a measure of how many ancestor classes can potentially affect this class. The deeper a class is in the hierarchy, the higher the degree of methods inheritance, making it more complex to predict its behavior. Deeper trees constitute greater design complexity, since more methods and classes are involved. The deeper a particular class is in the hierarchy, the greater the potential reuse of inherited methods.

3.2 NOC Metric

Chidamber and Kemerer proposed the Number Of Children of a class as the NOC metric for the class, which is the number of immediate subclasses subordinate to a class in the class hierarchy [13]. NOC is a measure of how many subclasses are going to inherit the methods of the parent class. The greater the number of children, the greater the potential for reuse, since inheritance is a form of reuse. The greater the number of children, the greater the likelihood of improper abstraction of the parent class. The number of children gives an idea of the potential influence a class has on the over all design.

4. Proposed Inheritance Metric

A class is composed of attributes and methods. In this proposal the Depth of Inheritance Tree of a Class (DITC) metric for class inheritance hierarchy is measured in terms of sum of the attributes (Private, Protected, public and inherited) and Methods (Private, Protected, public and inherited) at each level. The DITC metric of a class is calculated as:

$$DITC(C_i) = \sum_{i=1}^{L} LEV_i * i$$

Where,

LEV_i = Attribute (C_i) + Method (C_i)

C_i = A class in the i^{th} level of class inheritance hierarchy.

Attribute (C_i) = Count the total number of protected, private, public and inherited attributes within a class in the class inheritance hierarchy at each level.

Method (C_i) = Count the total number of protected, private, public and inherited methods within a class in the class inheritance hierarchy at each level.

L = Total height in the class inheritance hierarchy i.e. the maximum distance from the last node (last level in the class inheritance hierarchy) to the root node (first level in the class inheritance hierarchy), ignoring any shorter paths in case of multiple inheritance is used.

Viewpoints:

DITC Metric is based on the following assumptions:

- Deeper a particular class is in the class inheritance hierarchy at any level, greater the possibility of reusing inherited methods or attributes or both. This implies greater DITC and difficulty to maintain that class in class inheritance hierarchy. More development time will be required to analyze the class at this level in terms of design and coding.
- For classes at any level in class inheritance hierarchy absence of attributes (or inherited attributes) and methods (or inherited methods) will imply that DITC=0. Software developer requires some amount of time to analyze even for more classes and development time.
- High DITC indicates that more methods and attributes may be inherited at this level, thus making it more complex to predict the behavior of the class.
- A Deeper inheritance tree implies, greater DITC in the design. Development time increases in terms of design and coding as the level of class inheritance hierarchy increases, since more methods, attributes and classes are involved.

Consider the class inheritance tree in Figure 1 where, *Rounded Rectangle* represents *class wise information* i.e. first part contains *class name,* second part contains *attributes (or instance variables of a class),* and third part contains *methods.* From Figure 1, p_t represents protected, p_r represents private, p_u represents public and I_r represents inherited methods (or attributes) in a class. Development Time for Class A will be 3 minutes at level 1, Development Time for Classes B and C will be 4 minutes each at level 2 and Development Time for class D at level 3 will be 12 minutes. DITC Metric is calculated at each level are as follows:

At Level 1, DITC (A) = 1

At Level 2, DITC (B) = 2 DITC(C) =2

At Level 3, DITC (D) = 18

All the above values represent the DITC Metric of a class for the class inheritance hierarchy of Figure 1.For the above values, high values of the DITC (D) implies that more methods and attributes may be inherited making it more complex to predict the behavior of the class D and more development time will be required in terms of design and coding at Level 3.

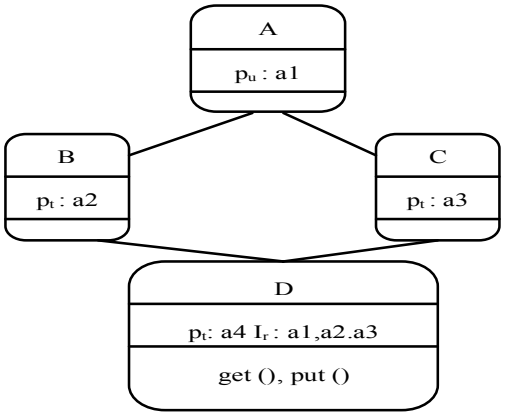

Figure.1 Class Inheritance Tree

5. Results

Statistical analysis on a small set of data of ten (10) class inheritance hierarchies from various sources [23]. Correlation coefficients for different class inheritance metric were calculated for a class inheritance hierarchy at each level with respect to the Development time (DEV) in minutes. The statistical analysis of the data in the tables has been generated with the aid of MATLAB [29], shown in Appendix (Table I and Table II). The statistical distribution of Table I data set is given in Appendix (shown in Figure 11 and Figure 12).

6. Discussion

6.1 Analytical Evaluation of DITC metric against Weyuker's properties

Let X_P =DITC for class P and X_Q =DITC for class Q. X_P and X_Q are the functions of the number of methods (public, private, protected, inherited) and number of instance variables (public, private, protected, inherited) at any level in the class inheritance hierarchy of class P and class Q.

It follows from assumption 1 [as shown in section 2] (since functions of independent and identically distributed, instance variables are also independent and identically distributed) that X_P and X_Q are independent and identically distributed. Therefore, property 1 (Non-Coarseness) and property 3(Non-Uniqueness) is satisfied because a statistical distribution of methods (public, private, protected, inherited) and number of instance variables (public, private, protected, inherited) among classes at any level in the class inheritance hierarchy is assumed. So at any level in class inheritance hierarchy

DITC (P) =DITC (Q) and DITC (P) ≠ DITC (Q).

Property 4 (Design details are important) is satisfied because design of class at any level in class hierarchy involves choosing what properties the class must inherit in order to perform its functions. Its means that the classes at the same level may have the same functionality but it does not guarantee that they have the same DITC metric value. In others words DITC metric is design implementation dependent.

When any two classes P and Q are combined there are three possible cases:

Case 1: class P and class Q are siblings

Case 2: class P and class Q are neither children nor siblings of each other.

Case 3: One is the child of other.

Case 1: class P and class Q are siblings. See Figure 2 and Figure 3.

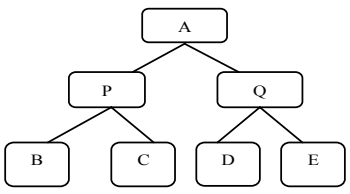

Figure 2 Class Inheritance tree when class P and class Q are at the same level

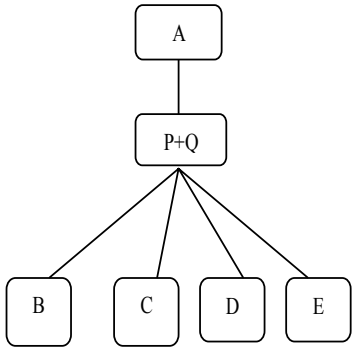

Figure.3 Class Inheritance tree when class P+Q is combined

From Figure 2, suppose Class A has two attributes (*say a, b*), one method (*say f ()*). Class P has one attribute (*say c*), one method (*say f1 ()*) and two inherited attributes from class A, so DITC (P) = 8 at level 2, and class Q has one attribute (*say d*), one method (*say f2 ()*) and two inherited attributes from class A, then DITC (Q) =8 at level 2. From Figure 3, when P and Q are combined the P+Q will contain two attributes (*say c, d*), two methods (*f1 (), f2 ()*) and two inherited attributes (*say a, b*) which is common both in P and Q, then DITC (P+Q) at level 2 is 12. So DITC (P) ≤ DITC (P+Q) and DITC (Q) ≤ DITC (P+Q). Property 5 is satisfied, which is also satisfied in DIT and NOC metric of Chidamber and Kemerer [13].

Case 2: when class P and class Q are neither children nor siblings of each other. See Figure 4 and figure 5.

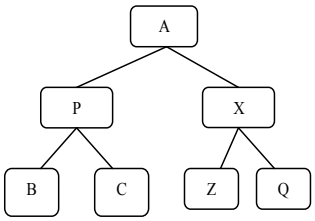

Figure 4 Class Inheritance tree when class P and class Q are at different level

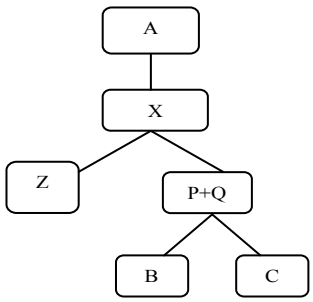

Figure.5 Class Inheritance tree when P+Q is combined

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

From Figure 4, suppose class A has two attributes (*say a, b*), class P has one attribute (*say c*) and two inherited attributes from class A. class X has one attribute (*say d*) and two inherited attributes from class A, class Q has two attributes (*say e,f*) and one inherited attributes from class X and two inherited attributes from class A. So DITC (P) =6 at level 2 and DITC (Q) =15 at level 3. From Figure 5, when class P and class Q are combined, then DITC (P+Q) =18 at level 2. So, DITC (P) ≤ DITC (P+Q) and DITC (Q) ≤ DITC (P+Q). Property 5 (Monotonicity) is satisfied, which is also satisfied in DIT and NOC metric of Chidamber and Kemerer [13].

Case 3: one is the child of other. See Figure 6 and Figure 7.

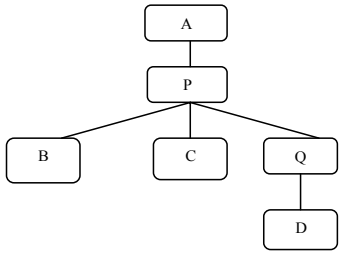

Figure.6 Class Inheritance tree when class P and class Q are at different level and one is a child of another

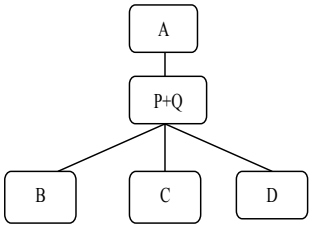

Figure.7 Class Inheritance tree when class P+Q is combined

From Figure 6, suppose class A has one attribute (*say a*), class P has one attribute (*say b*) and one inherited attribute from class A then, DITC (P) = 4 at level 2, class Q has one attribute (*say c*) and two inherited attributes one from class P and one from class A then, DITC (Q) =9 at level 3. From Figure 7, When class P and class Q are combined, DITC (P+Q) = 6 at level 2. So, DITC (P+Q) ≤ DITC (Q) that violates the property 5. Hence, Property 5 (Monotonicity) is not satisfied, which is not satisfied in DIT metric but satisfied in NOC metric of Chidamber and Kemerer [13].

See Figure 8, Figure 9 and Figure 10.

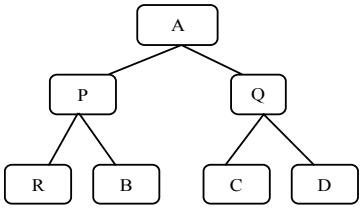

Figure.8 Class Inheritance tree when class P and class Q are at same level and class R is a child of class P

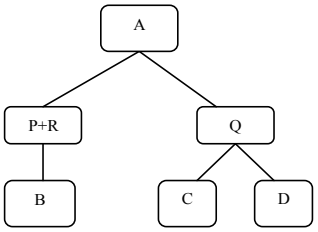

Figure.9 Class Inheritance tree when class P +R is combined

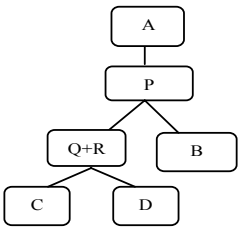

Figure.10 Class Inheritance tree when class Q +R is combined

From Figure 8, let class P and class Q be the siblings of class A and class R be child of class P. suppose class A has one attribute (*say a*), class P has one attribute (*say b*) and inherited attribute from class A, class Q has one attribute (*say c*) and inherited attribute from class A. So, DITC (P) = 4 and DITC (Q) = 4 at level 2. Suppose, class R has one attribute (*say d*) and two inherited attributes one from class P and one from class A. From Figure 9, when class P and class R is combined then, DITC (P+R) = 6 at level 2. When class Q and class R is combined then, from Figure10, DITC (Q+R) = 12. Therefore, DITC (P) = DITC (Q) from Figure 8 at level 2, it does not imply that, DITC (P+R) = DITC (Q+R). Hence, Property 6 (non-equivalence of interaction) is satisfied. Property 7 requires that permutation of program statements can change the metric value. This

metric is meaningful in traditional program design. In object-oriented design the ordering of statements within the class in class inheritance hierarchy at any chosen level will have no effect in eventual execution. Hence, property 7 is satisfied. But, it has been suggested that property 7 is not appropriate for OOD (Object-Oriented Design) metrics. Property 8 will be satisfied by all metrics measured at the class level, since they will not be affected by the names of classes in the class inheritance hierarchy or the methods or the instance variables. For any two classes P and Q, such that from Figure 6 and Figure 7 in case 3 as stated in property 5, DITC (P) = 4 at level 2 and DITC (Q) = 9 at level 3 (from Figure 6). DITC (P+Q) =6 at level 2 (from Figure 7). Therefore, DITC (P) + DITC (Q) = 13. So, DITC (P) + DITC (Q) is not less than DITC (P+Q), it violates the property 9 (Interaction increases complexity) condition (DITC (P) + DITC (Q) < DITC (P+Q)). Hence, property 9 is not satisfied, which is also not satisfied in DIT and NOC metric of Chidamber and Kemerer [13].

6.2 Differences among the inheritance metrics studied

DIT metric is the maximum distance from the node to the root of the tree (ignoring any shorter paths in case of multiple inheritance is used), whereas NOC metric count the number of immediate classes directly subordinate to a class in the class hierarchy. NOC does not count the number of non-immediate subclasses in the class hierarchy since class has influence over all of its subclasses. Both DIT and NOC are not focused on the properties of the classes, because deeper a particular class in the class hierarchy, greater the complexity of class hierarchy, since more methods and attributes are involved. None of the above metrics considers the internal characteristics (variables or methods) of a class. DITC metric measure the Depth of inheritance tree of a class of class inheritance hierarchies in terms of sum of the attributes (private, public, protected & inherited) and methods (private, public, protected and inherited) at each level. DITC metric focuses on the properties (method, attribute) of a class, so it can easily view what data members and functions can be inherited by the class and which super class/ super interfaces bring these members.

See in Table 3 (shown in Appendix), Analytical Evaluation results of inheritance metrics against Weyuker's axioms [18].

6.3 Observations

Certain interesting observation from Table 1 (shown in Appendix) can be made. According to the mean statistics classes in hierarchy H6 at any chosen level are most likely to inherit more member functions (or methods) and attributes, thus making it more complex to predict the behavior of classes in hierarchy H6 and more development time will be required to analyze the class at this level in terms of design and coding, than in H7, H1, H8, H4, H5, H10, H9, H2, H3. High mean value of H6 indicates, greater possibilities of reusing inherited methods or attributes or both at lowest level in H6 implying greater DITC of particular class in H6 and harder to maintain that class in hierarchy, since more methods, attributes and classes are involved. The same is true for Median (first H6 then H5 and H4, H9 and H1, H10 and H2, H7, H8, H3) and Standard Deviation (S.D.) statistics (first H8 then H7, H3, H5, H4, H2, H1, H10, H9, H6). According to both Mean and Median statistics Hierarchy H6 is most likely to inherit more member functions (or methods) and attributes.

Now consider the entries in Table 2 (shown in Appendix). The proposed metric DITC correlates very well with development time (DEV) (first row of Table 2) and can be used as a good predictor for it. In all the columns, the entries corresponding to DITC metric is the highest except in one case for hierarchy H8 where DIT performs slightly better than DITC.

7. Conclusion and Future scope

An attempt has been made to define a class inheritance metric based upon the sum of the number of attributes (private, public, protected & inherited) and methods (public, protected, private & inherited) to measure the Depth of Inheritance Tree of a Class for class inheritance hierarchy at each level. In the work presented here, the goal was to find the effect of the different class inheritance metrics values at each level upon the development time (DEV). The approach taken was analytical and empirical. The DIT metric of Chidamber and Kemerer [13] was used to derive from DITC Metric (DIT=L) as stated in Section 4. The object-oriented language used in the data set was C++. As seen from Table 2, the DITC metric is a good predictor of development time since the correlation is very high in all the cases.

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. Some more program parameter has to be incorporated to DITC metric for satisfying the Weyuker's property 5 and property 9. Also further characteristics of classes need to be studied to establish an empirical relationship between the different class inheritance metric and proposed one w.r.t. to development time and behavior of the classes.

The future work will be towards further validation with an extended set of classes and further evaluation of proposed DITC metric will in turn improvement of the quality of classes.

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., 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, Melo. WL, "A validation of object-oriented design metrics as quality indicators", Technical report, University of Maryland, Department of Computer Science, 1995; 1-24.

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

[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, 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, 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, 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, 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, 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, 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, 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, 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, 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

Appendix

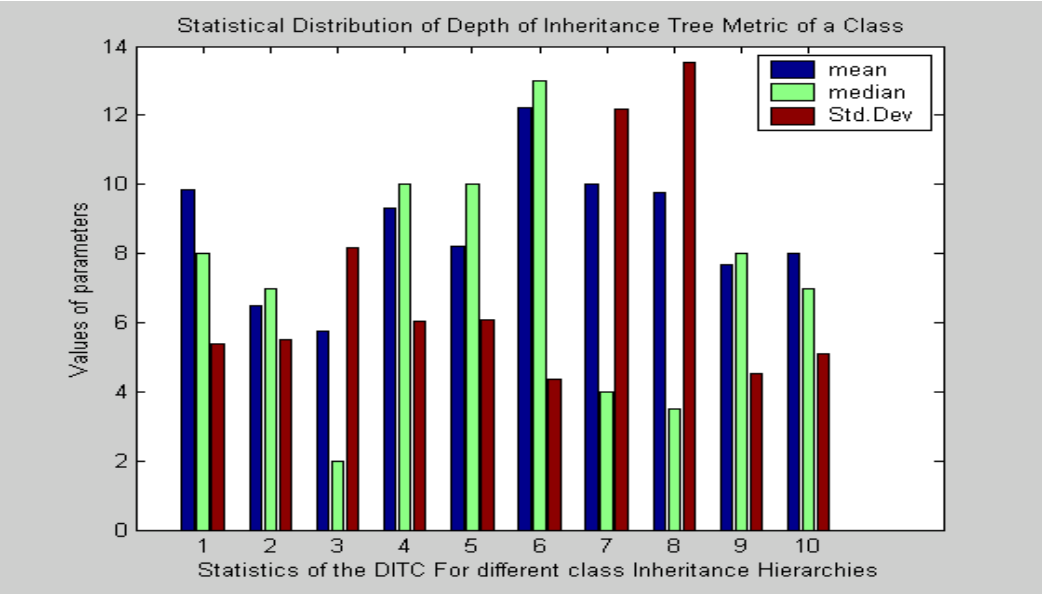

Figure 11. Statistics of the DITC metric for different class Inheritance Hierarchies.

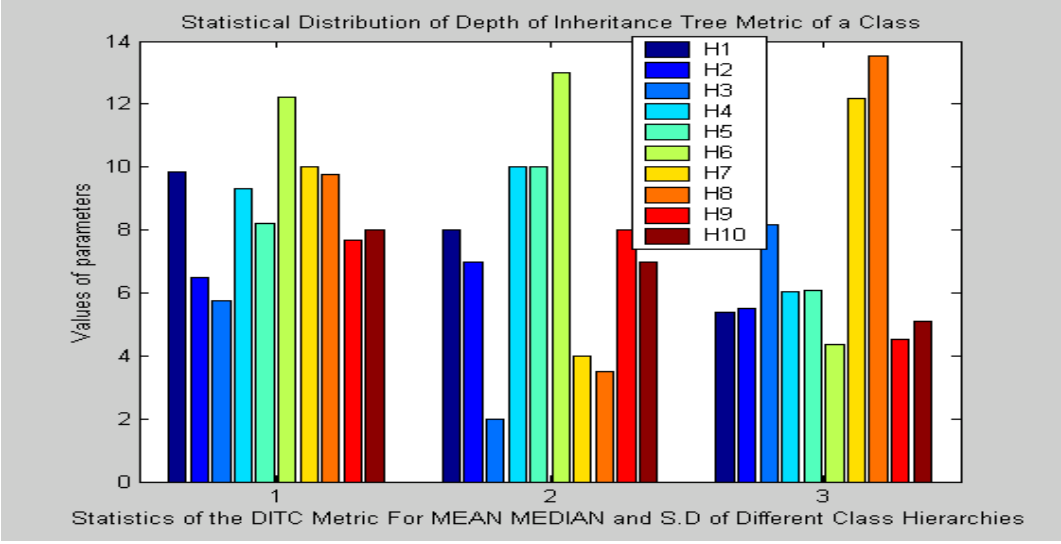

Figure 12. Statistics of the DITC Metric for the Mean ,Median and Standard Deviation of different C++ Class Inheritance Hierarchies.

	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10
Mean	9.8571	6.5	5.75	9.3333	8.2	12.2	10	9.75	7.667	8
Median	8	7	2	10	10	13	4	3.5	8	7
Std.Dev	5.3675	5.5076	8.1803	6.0277	6.0992	4.3665	12.166	13.53	4.5092	5.099

Table 1: Descriptive Statistics for the DITC Metric Analysis of Different Class Hierarchies

	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10
DITC	0.8850	0.9540	0.9878	0.9099	0.9845	0.8296	0.9987	0.9679	0.9921	0.9309
DIT	0.1357	0.2208	0.9206	0.8660	0.5514	0.6660	0.9245	0.9780	0.9820	0.9535
NOC	0.2195	-0.2208	-0.9206	-0.5	-0.5183	-0.7276	-0.9912	-0.9540	-0.7559	-0.7303

Table 2: Correlation Coefficient of Class Inheritance Metrics at each level in the Class Inheritance Hierarchies w.r.t. Development Time (DEV)

Weyuker's Properties No.	DITC	DIT	NOC
1	√	√	√
2	√	√	√
3	√	√	√
4	√	√	√
5	X	X	√
6	√	√	√
7	√	√	√
8	√	√	√
9	X	X	X
√ indicates that the metric satisfies the corresponding property. X indicates that the metric does not satisfy the corresponding property.			

Table 3: Analytical Evaluation Results for DITC, DIT and NOC Metrics Against Weyuker's Properties