International Journal of Engineering and Technology Volume 1 No. 1, October, 2011

An Analytical Evaluation on Li's Inheritance Metric Suites against Weyuker's Properties

Kumar Rajnish[1], **Vandana Bhattcaherjee**[2]

[1]Department of Computer Science & Engineering
Birla Institute of Technology
International Centre (Waljat Colleges Applied Sciences)
P.O. Box-197, Rusayl, Muscat, Postal Code-124, Sultanate of Oman
{krajnish@bitmesra.ac.in}

[2]Department of Computer Engineering and Engineering
Birla Institute of Technology
Mesra, Lalpur
ZIP-834 001, Ranchi, India
{vbhattacharya @bitmesra.ac.in}

ABSTRACT

Inheritance is a powerful mechanism in an Object-Oriented (OO) programming. This mechanism supports the class hierarchy design and captures the IS-A relationship between a super class and its subclass. Since the analytical evaluation of Li's inheritance metrics [19] is not described in the literature. In this paper an attempt has been made to present an analytical evaluation on Li's inheritance metric suite [19] against Weyuker's properties [24] and an attempt has also been made to present the results of analytical evaluation of Chidamber and Kemerer inheritance metric [11], Brito and Carapuca candidate structural inheritance complexity metric [33], and Rajnish and Bhattacherjee inheritance metric [34] against Weyuker's properties [24].

Key words: *Inheritance Hierarchy, Object-Oriented, Inheritance Tree, Metrics, Object-Oriented Design, and 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 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 technology forces the growth of object oriented software metrics [1]. Several such metrics have been proposed and their reviews are available [2-9]. The metrics suite proposed by Chidamber and Kemerer(C&K) is one of the best-known OO metrics [10-11]. 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 [12-13]. Alshayeb and Li predict that OO metrics are effective (at least in some cases) in predicting design efforts [14]. Chae, Kwon and Bae investigated the effects of dependence variables on cohesion metrics for OO programs [15]. Several other researchers have validated OO metrics for effects of class size and with the change proneness of classes [16-18]. Li [19] theoretically validated Chidamber and Kemerer metrics using a metric evaluation framework proposed by Kitchenham et al [20] 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. Brito and Carapuca have studied on candidate structural inheritance complexity metric [33]. 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 [23] [25]. Rajnish and Bhattacherjee have 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 [26-29] [35]. Rajnish and Bhattacherjee have also studied on the class inheritance hierarchies which is 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 on various C++ class hierarchies [21-22] [30] [32] [34] [39] [40] . Rajnish and Bhattacherjee have also studied to measure the complexity of class at the design stage as in [31].

The rest of the paper is organized as follows: Section 2 presents the Weyuker's properties; Section 3 presents some inheritance metrics; Section 4 presents the analytical evaluation results of some inheritance metrics; Section 5 presents the analytical evaluation on Li's inheritance metric suites against Weyuker's properties and finally Section 6 deals with conclusion.

2. WEYUKER'S PROPERTIES

The basic nine properties proposed by Weyuker's [24] are listed below. The notations used are as follows: *class P*, *class Q*, and *class R* denote classes, *class 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 $P \equiv Q$ (*class P* is equivalent to class Q) means that two classes designs, *class P* and *class Q*, provide the same functionality. The definition of combination of two classes is taken here to be the same as suggested by [24], 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, *class P* and *class Q*, which provide the same functionality, it does not imply that the metric values for *class P* and *class 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 *class P + Q* implies combination of *class P* and *class 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:

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

Some researchers have criticized Weyuker's list of properties; 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 OOD metrics.

Assumptions: Some basic assumptions used in Section 5.1 and Section 5.2 under Section 5 which has been taken from Chidamber and Kemerer [11] 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 identicals 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. INHERITANCE METRICS

In this section we present the brief description of the inheritance metrics proposed various researchers. The inheritance metrics proposed by Chidamber and Kemerer [11] are summarized in Table 1. Brito and Carapuca have proposed candidate inheritance metrics [33] and are summarized in Table 2. Li's [19] have proposed alternative representation of Depth of Inheritance Tree (DIT) and Number Of Children (NOC) metrics of Chidamber and Kemerer [11] and are summarized in Table 3. Rajnish and Bhattacherjee have proposed class

inheritance metric [22] [30] [34] and are summarized in Table 4.

Table 1: Chidamber and Kemere Inheritance Metrics

Metrics	Description
Depth of Inheritance Tree (DIT)	Depth of inheritance of the class is the DIT metric for the class. In cases involving multiple inheritance, the DIT will be the maximum length from the node to the root of the tree.
Number of Children (NOC)	Number of immediate subclasses subordinated to a class in the class inheritance hierarchy is the NOC for that class.

Table 2: Brito and Carapuca Inheritance Metrics

Metrics	Description
Total Progeny Count (TPC)	Number of classes that inherit directly or indirectly from a class is the Total Progeny Count (TPC) of that class.
Total Parent Count (TPAC)	The number of superclasses from which a class inherits directly is the Total Parent Count (TPAC) of that class.
Total Ascendancy Count (TAC)	The number of superclasses from which a class inherits directly or indirectly is the Total Ascendancy Count (TAC) of that class.

Table 3: Li's Inheritance Metrics

Metrics	Description
Number of Ancestor Class (NAC)	The definition of NAC measures the total number of ancestor classes from which a class inherits in the class inheritance hierarchy.
Number of Descendent Class (NDC)	The definition of NDC metric is the total number of descendent classes (subclasses) of a class.

4. Analytical Evaluation Results of Inheritance Metrics

Table 5 summarizes the evaluation results for inheritance metrics listed in Section 3 for Weyuker's properties. The evaluation of the inheritance metrics listed in Table 5 is described in and [11] [36] and [34]. In the next Section we presents the analytical evaluation of Li's NAC and NDC metrics against Weyuker's properties [24].

Table 4: Rajnish and Bhattacherjee Inheritance Metrics

Metric	Description
Depth of Inheritance Tree of a Class (DITC)	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.

Table 5: Analytical Evaluation results of some Inheritance metrics

Weyuker property No	DIT	NOC	TPC	TPAC	TAC	DITC
1	√	√	√	√	√	√
2	√	√	√	√	√	√
3	√	√	√	√	√	√
4	√	√	√	√	√	√
5	×	√	×	√	×	×
6	√	√	√	√	√	√
7	√	√	√	√	√	√
8	√	√	√	√	√	√
9	×	×	×	×	×	×

√ indicates that metric satisfies the corresponding property
X indicates that the metric does not satisfy the corresponding property

5. Analytical Evaluation on Li's NAC and NDC Metric against Weyuker's Properties

In this Section we present an analytical evaluation of Li's Number of Ancestor class (NAC) and Number of Descendent Class (NDC) metric against Weyuker's properties.

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* 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.

5.1 Analytical Evaluation Of Number Of Ancestor Class (NAC) Metric

Let X_P =*NAC* for *class P* and X_Q =*NAC* for *class Q*. X_P and X_Q are the functions of the number of methods in the class inheritance tree of *P* and *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. The Number of Ancestor Class (NAC) of a leaf is always greater than that of the root. Therefore, *property 1 (non-coarseness)* is satisfied. Also, since every tree has at least some nodes with siblings there will always exists at least two classes with the same Number of Ancestor Class i.e. *NAC (P) = NAC (Q).* Therefore, *property 3 (non-uniqueness)* is satisfied.
Property2 (Granularity) is trivially satisfied by any metric defined for a class. There is a finite number of cases where at any level in the class inheritance tree the *NAC* metric have the same values.

Design of class involves choosing what properties the class must inherit in order to perform its function. In other words, Number of Ancestor Classes is design implementation dependent. It means that two classes may have the same functionality but it does not guarantee that they have the same *NAC* values. Therefore, *property 4 (design details are important)* is satisfied.
When any two classes *P* and *Q* are combined there are three possible cases:

Case 1: *class P* and *class Q* are siblings (see Figure 1 and Figure 2)

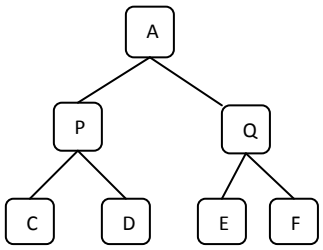

Figure 1

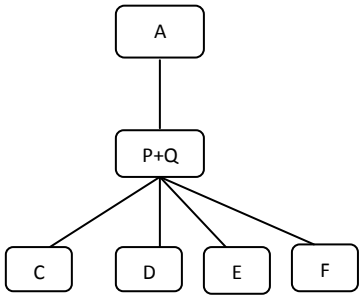

Figure 2

In this case, *NAC (P) = NAC (Q) = n* (from Figure 1), and *NAC (P+Q) = n* (from Figure 2). Therefore, *property 5 (monotonicity)* is satisfied.

Case 2: class P and *class Q* are neither children nor siblings of each other (see Figure 3 and Figure 4)

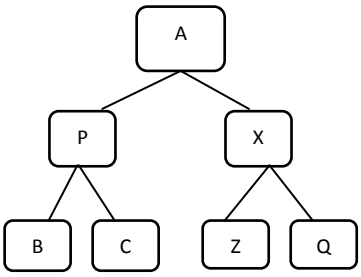

Figure 3

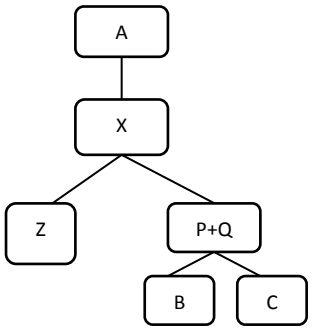

Figure 4

If *class P+Q* is located as the as the immediate ancestor to *class B* and *class C* (*class P's* location) in the tree, the combined class cannot inherits methods from *class X*, however if *class P+Q* is located as the immediate child of *class X (class Q's* location), the combined class can still inherits methods from all the ancestors of *class P* and *class Q*. Therefore, *class P+Q* will be in located *class Q's* location. In this case from Figure 3, *NAC (P) = x* and *NAC (Q) = y* and *y > x*. From Figure 4, *NAC (P+Q) = y* i.e. *NAC (P+Q) > NAC (P)* and *NAC(P+Q)=NAC(Q)*. Therefore, *property 5 (monotonicity)* is satisfied.

Case 3: When one is child of other (see Figure 5 and Figure 6)

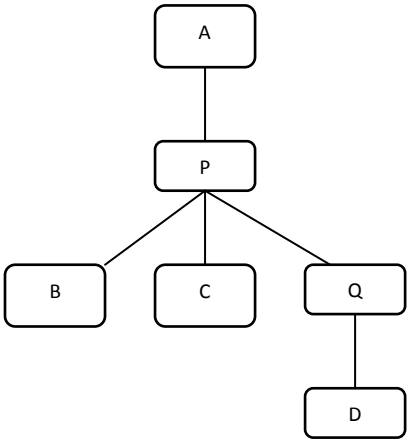

Figure 5

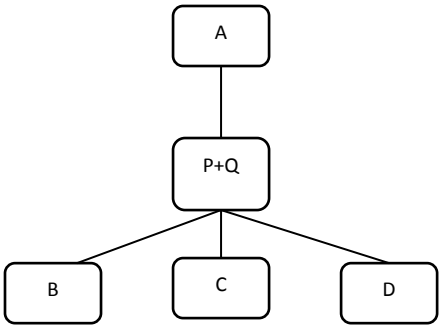

Figure 6

From Figure 5 *NAC (P) =x* and *NAC (Q) =y* and *y > x*. From Figure 6, *NAC (P+Q) = x*. So, *NAC (P) = NAC (P+Q)* and *NAC (P+Q) ≤ NAC (Q)*. Therefore, *property 5 (monotonicity)* is not satisfied.

See Figure 7, Figure 8 and Figure 9.

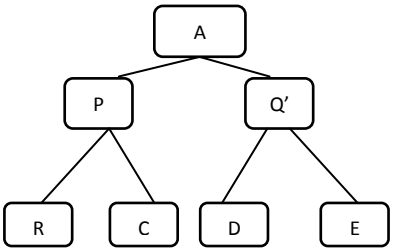

Figure 7

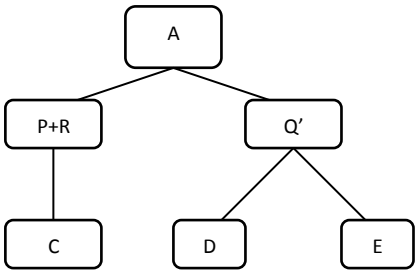

Figure 8

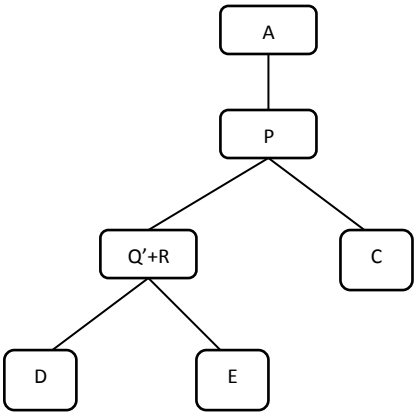

Figure 9

Let *class P* and *class Q* be sibling's i.e. *NAC (P) = NAC (Q') = n* from Figure 7, and let *class R* be a child of *class P* then *NAC (P+R) = n* from Figure 8 and from Figure 9, *NAC (Q'+R) = n+1* i.e. *NAC (P+R)* is not equal to *NAC (Q'+R)*. Therefore, *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 tree will have no effect in eventual execution. Hence, *property 7* is satisfied. But, it has been suggested that *property 7* is not appropriate for Object-Oriented Design (OOD) 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 tree or the methods.

For any two classes *P* and *Q*, $NAC\ (P+Q) = NAC\ (P)$ or $NAC\ (Q)$. Therefore, $NAC\ (P+Q) \leq NAC\ (P) + NAC\ (Q)$ i.e. *property 9* is not satisfied.

5.2 Analytical Evaluation of Number of Descendent Class (NDC) Metric

Let $X_P = NDC$ for *class P* and $X_Q = NDC$ for *class Q*. X_P and X_Q are the functions of the number of methods in the class inheritance tree 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.

It follows from assumption 3 [as shown in Section 2] the inheritance hierarchy has a root and leaves. Let *class P* and *class R* be leaves (see Figure 10), $NDC\ (P) = 0$ and $NDC\ (R) = 0$. Let *class Q* be the root (see Figure 10), $NDC\ (Q) > 0$, $NDC\ (P) \neq NDC\ (Q)$. Therefore, *property 1* is satisfied. Since $NDC\ (R) = NDC\ (P)$, *property 3* is also satisfied.

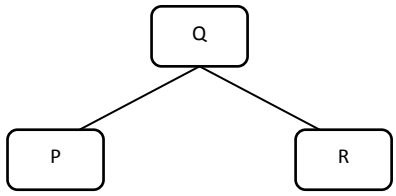

Figure 10

Property2 (Granularity) is trivially satisfied by any metric defined for a class. There is a finite number of cases where at any level in the class inheritance tree the *NDC* metric have the same value.

Design of a class involves decision in the scope of the methods declared within the class i.e. the sub classing for the class. The number of subclass is therefore dependent upon the design implementation of the class. Therefore, *property 4* is satisfied.

Let *class P* and *class Q* be two classes with N_P and N_Q subclasses respectively (i.e. $NDC\ (P) = N_P$ and $NDC\ (Q) = N_Q$). Combining *class P* and *class Q* will yield a single class with $N_P + N_Q - \delta$ subclasses, where δ is the number of children *class P* and *class Q* have in common. Clearly, δ is *0* if either N_P or N_Q is *0*. If *class Q* is a subclass of *class P*, then *class P+Q* will have $N_P + N_Q - 1$ subclasses. Therefore, the number of subclasses of *class P+Q* is $N_P + N_Q - \beta$, where $\beta = 1$ or δ. Now, $N_P + N_Q - \beta \geq N_P$ and $N_P + N_Q - \beta \geq N_Q$. This can be written as $NDC\ (P+Q) \geq NDC\ (P)$ and $NDC\ (P+Q) \geq NDC\ (Q)$ for all *class P* and all *class Q*. Therefore, *property 5* is satisfied.

From Figure 7, Figure 8, and Figure 9. From Figure 7, let *class P* and *class Q'* each have *n* children and *class R* be a child of *class P* which has *r* children. Then, $NDC\ (P) = n$, and $NDC\ (Q') = n$. The class obtained by combining *class P* and *class R* will have $(n-1) + r$ children (see Figure 8), whereas class obtained by combining *class Q'* and *class R* will have $n + r$ children (see Figure 9), which means that $NDC\ (P+R) \neq NDC(Q'+R)$. Therefore, *property 6* 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 tree will have no effect in eventual execution. Hence, property 7 is satisfied. But, it has been suggested that property 7 is not appropriate for Object-Oriented design (OOD) metrics.

Property 8 is trivially satisfied by any metric, which is measured at the class level.

Given any two classes *P* and *Q* with N_P and N_Q children respectively, the following relationship holds:

$NDC\ (P) = N_P$ and $NDC\ (Q) = N_Q$

$NDC\ (P+Q) = N_P + N_Q - \delta$.

Where, δ is the number of common children. Therefore, $NDC\ (P+Q) \leq NDC\ (P) + NDC\ (Q)$. *Property 9* is not satisfied.

6. CONCLUSION

Since the analytical evaluation of Li's NAC and NDC metrics [19] is not done in the literature. In this paper we have attempted to present the analytical evaluation of NAC and NDC metrics against Weyuker's properties [24] From Table 5 and Section 5 it is observed that *property 9* is not satisfied by any of the inheritance metrics. Failing to meet *property 9* means that complexity cannot be reduced by dividing a class. Since complexity cannot be reduced, it may increase (but also may stay the same). The widely criticized *property 9* of Weyuker's axiomatic system is shown to be applicable to OO systems through counter examples to earlier arguments. Whether Weyuker's *property 9* is applicable to inheritance metrics is under debate in the literature. Based on the observation that none of the metrics proposed in [11] [[33] [34]

satisfies this property, Gursaran and Roy try to prove that some classes of inheritance metrics can never satisfy *property 9* [37]. However, there are some discrepancies in their proofs and, therefore, exceptions can be found to their conclusion. However, two counter examples to their formalization were presented by Zhang and Xie [38]. They gave an inheritance metric, which satisfied *property 9*. However, they were not be able to find any practical example for their metric that satisfied *property 9*. Rajnish and Bhattcaherjee had proposed one method for the applicability of Weyuker's property 9 to inheritance metrics as in [32]. They gave an inheritance method which satisfied Weyuker property 9.

REFERENCES

[1] G.Booch, "*Object-Oriented Design and Application"*, Benjamin/Cummings, Mento Park, CA, 1991

[2] J. M. Bieman.and B. K. Kang, "Cohesion and Reuse in an Object-Oriented System", in *Proc. Symp. Software Reliability*, (1995), pp.259-26.

[3] L. C. Briand, W. J. Daly and J. K. Wust, "A Unified Framework for Cohesion Measurement in Object-Oriented Systems", *Empirical Software Eng.*, 1, 1 (1998), pp.65-117.

[4] F. Brotoeabreu, "The MOOD Metrics Set", in *Proc. ECOOP'95 Workshop Metrics*, 1995.

[5] H.S.Chae, Y. R. Kwon and D. H. Bae, "A Cohesion Measures for Object-Oriented Classes", Software practice and Experiences, 30, 12 (2000), pp.1405-1431.

[6] 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.

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

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

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

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

[11] 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.

[12] L. C. Briand and J. K. Wust, "Modeling Development Effort in Object-Oriented Systems Using Design Properties", *IEEE Trans. on Software Engineering.* , 27, 11(2001), pp.963-986.

[13] 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.

[14] 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.

[15] H. S. Chae, Y. R. Kwon and D. H. Bae, "Improving Cohesion Metrics for Classes by considering Dependent Instance Variables", *IEEE Trans. on Software Engineering*, 20, 6 (1994), pp.476-493.

[16] E. Arisholm, L. C. Briand and Foyen, "A Dynamic coupling measures for Object-Oriented Software", *IEEE Trans. on Software Engineering*, 30, 8 (2004) pp. 491-506.

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

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

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

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

[21] 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.

[22] 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.

[23] V. Bhattacherjee and K. Rajnish, " Class Complexity-A Case Study", *Proceedings of First International Conference on Emerging Application of Information Technology (EAIT-2006)*, Kolkata,India,2006, pp. 253-258.

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

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

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

[27] 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.

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

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

[30] 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.

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

[32] 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. http://ICIT2007.home.comcast.net/, http://www.computer.org.

[33] A. F. Brito and R. Carapuca, "Candidate Metrics for Object-Oriented Software within a Taxonomy Framework, *Journal of System Software*, vol. 26, 87-96.

[34] 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.

[35] K. Rajnish and V. Bhattacherjee, "Interaction among classes and its effect on Cohesion Metric", *proceedings of Second National Conf. on Methods and Models in Computing (NCM2C-2007)*, School of Computer System Sciences, JNU, New Delhi, India, 2007, pp.17-26.

[36] 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.

[37] 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.

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

[39] 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.

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