Journal of Theoretical and Applied Information Technology

www.jatit.org

COMPLEXITY OF CLASS AND DEVELOPMENT TIME: A STUDY

KUMAR RAJNISH, VANDANA BHATTACHERJEE
Department of Computer Science and Engineering,
Birla Institute of Technology, Ranchi-834 001, Jharkhand, India

ABSTRACT

Software development time depends on various attributes of the software product. In this paper, attempt has been made to define an empirical relation between software development time with respect to its dependence on lines of code, variables and methods. We have attempted to analyze the various dependencies of development time of a program upon its member functions, instance variables and the number of non-blank, non-comment lines of code. Statistical techniques have been used to assign weights to the independent variables to arrive at the results.

Keywords: Classes, Metrics, Complexity, Object-Oriented Design.

INTRODUCTION

Software metrics are units of measurement, which are used to characterize software engineering products, processes and people. By careful use, they can allow us to identify and quantify improvement and make meaningful estimates. Developers in large projects use measurements to help them understand their progress towards completion. Managers look for measurable milestones so that they can assess schedule and other commitments. The metrics gathered from historical data also provide an estimate of future similar projects.

Program complexity plays an important role in the amount of time spent on development of the program. This paper presents the result of a research conducted to study the effect of program complexity (measured in terms of its member functions, instance variables and lines of code, viz., non-blank, non-comment lines) upon the development time of various C++ classes. We consider one metric for OO design and incorporate our own variations in it to study its effect on the development time of various C++ classes. Statistical techniques have been used to assign weights to the independent variables to arrive at the results. An earlier version of this work has been presented in [25].

The recent trend towards OO technology has forced the growth of OO software metrics as mentioned by Booch [4]. Several such metrics have been proposed and their reviews are available in literature such as [3][5][8][10][12-13][15][18-19][21-22]. The metrics suite proposed by Chidamber et al is one of the best-known OO (object-oriented) metrics [12-13]. We shall henceforth call them C&K metrics.

Various researchers have also conducted empirical studies to validate the OO metrics for their effects upon program attributes. Alshayeb and Li have presented an empirical study of OO metrics in two processes [1]. They predict that OO metrics are effective in predicting design efforts and lines of source code added, changed and deleted in one case and ineffective in other. Emam, Benlarbi, Goel and Rai validate the various OO metrics for effects of class size [16]. This view is however not agreed to by Evanco [17]. Other researchers have also conducted empirical studies for investigating relationship between metrics and quality factors such as development/ maintenance effort [7][20]. Arisholm, Briand and Foyen study various Java classes to empirically evaluate the effect of dynamic coupling measures with the change proneness of classes [2]. Chae, Kwon and Bae investigated the effects of dependent instance variables on cohesion metrics for object-oriented programs [11]. They also proposed an approach to

identify the dependency relations among instance variables.

A metric may be validated mathematically using measurement theory, or empirically by collecting data. Measurement theory attempts to describe fundamental properties of all measures. Weyuker concentrated on finding desirable properties that these measures should satisfy [24]. Weyuker proposed nine properties that partially characterize good software measures. We must however mention, that not all measures satisfy all of the nine properties as shown by Chidamber et. al. [13]. In this paper we consider a metric from the Chidamber and Kemerer(C&K) metric suite and use it to derive another metric to be used in our study. We first present an analytical evaluation of the derived metric against Weyuker's properties and secondly, empirical evidence that the development time of a class (and thus of an OO program) is highly correlated to the methods in a class, the number of variables and the lines of code in it.

The rest of the paper is organized as follows. In section 2I we discuss the WMC (Weighted Method per Class) measure of the C&K metric suite. In section 3, we list out Weyuker's nine properties to make the discussion complete. In section 4, we present the proposed metrics to be used in our study. Section 5 presents statistical analyses of data from different data sets. Sections 6 present the discussion and future scope respectively.

WMC METRIC OF C & K

Consider a class C1 with methods M1, M2, M3,….Mn that are defined in the class. Let c1, c2, c3,…..cn be the complexity of the methods.

Then,

$$\text{WMC} = \sum_{i=1}^{n} ci$$

If all method complexities are considered to be unity, then WMC = n, the number of methods.

Complexity of an individual as defined by Bunge is the "numerosity of its composition" [9]. Thus it can be said that the complexity of an object is the cardinality of its set of properties. In object oriented terminology, the properties of an object include the instance variables and its methods. As mentioned in Chidamber et al, WMC relates directly to Bunge's definition of complexity of a thing, since methods are properties of object classes and complexity is determined by the cardinality of its set of properties. The number of methods is therefore, a measure of class definition as well as being attributes of a class since attributes correspond to properties. They further mention that the number of instance variables has not been included in the definition of the metric since it was assumed that methods are more time consuming to design than instance variables [13].

WEYUKER'S PROPERTIES

The basic nine properties proposed by Weyuker's are as follows:

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 OOD metrics. In our discussion, therefore, these three mentioned properties shall not be considered.

PROPOSED METRIC

The metric FVL (Functions, Variables and Lines of code) that we have derived from the WMC metric for measuring the complexity of a class is based upon the following assumptions:
The mental exercise required to design and code a class depends not only upon the numbers of methods but also upon the distinct variable names. This means when a developer needs to use a new variable, he spends some amount of time in framing it out.

The number of methods is a predictor of how much time and effort is required to develop and maintain the class.
Method names are counted as distinct variable names.
A local variable of same name used in two different blocks is considered to have two distinct variable names.

To calculate FVL, we take the Lines Of Code of the entire class (LOC), the number of Methods Per Class (MPC) and the Distinct Variable Names (DVN). The formula for FVL is:
FVL = k + w1*MPC + w2 *DVN + w3*LOC
Where, the weights w1, w2, w3 and the constant k are derived at by least square regression analysis.
Note that when all method complexities are considered to be unity, the WMC metric proposed by C&K is obtained from MPC.

We give an outline of our approach. The variables of interest in our study are: MPC, DVN, LOC and the Development Time DEV in minutes, which is to be modeled by our metric. The above-mentioned four values were collected for classes from four different categories. The first data set was collected by sampling classes from third year Post Graduate students' projects and will be called Set A. The second data set was from samples of classes from laboratory exams of second year Post Graduate students and is called Set B. The data of Set C was from miscellaneous applications. The fourth data set was collected by sampling classes from second year Post Graduate M.SC (IT) students and second year Post Graduate MCA student's of Birla Institute of Technology, and will be called Set D.

RESULTS

Multivariate regression analysis was applied on all the five data sets, and correlation coefficients were calculated. In each case 75% of the data was used to derive at parameter values and 25% was used for validation. The statistical analysis of the data in the following tables has been generated with the aid of MATLAB [23]. The statistical distribution of different data sets is given in Appendix.

Table 1. Data from classes of set A

	MPC	DVN	LOC	DEV
Mean	3.1795	3.6410	67.102	2.64
Median	3.0	3.0	35.0	1.9
Std. Dev.	1.4667	2.3674	87.870	2.270
Max	8	13	400	12
Min	1.0	1.0	14.0	0.5

Table 2. Data from classes of set B

	MPC	DVN	LOC	DEV
Mean	2.9167	5.8333	52.1667	0.8156
Median	2.5000	5	37.5000	0.7885
Std. Dev.	1.8320	3.4597	39.3327	0.3880
Max	7	15	166	1.7500
Min	1	3	20	0.2500

Table 3. Data from classes of set C

	MPC	DVN	LOC	DEV
Mean	13.556	28.889	341.07	42.73
Median	5	10	116	4.500
Std. Dev.	35.9694	45.980	471.26	95.92
Max	191	175	1867	360
Min	1	3	20	0.25

Table 4. Data from classes of set D

	MPC	DVN	LOC	DEV
Mean	3.0702	4.0526	66.5439	32.5965
Median	3	4	53	30
Std. Dev.	1.5337	2.3408	36.5431	18.2364
Max	7	11	177	90
Min	1	1	14	12

Table 5. Values of the coefficients for the Three Independent Variables used in FVL Metric from four different data sets by Least Square Regression Analysis

	w1	w2	w3
SET A	0.2846	-0.0102	0.0287
SET B	0.0113	0.1662	-0.0044
SET C	1.9193	0.4535	-0.0122
SET D	1.5287	0.5800	0.3797

Table 6. Correlation coefficient with respect to development time (DEV).

	MPC	DVN	LOC	FVL
SET A	0.5538	0.9312	0.9427	0.9288
SET B	0.9936	0.9231	0.8330	0.9484
SET C	0.9575	0.9411	0.6818	0.9677
SET D	0.5295	0.3944	0.7049	0.7177

DISCUSSION AND FUTURE SCOPE

We first present an analytical evaluation of our measure against Weyuker's axioms. Properties 1 (Non-coarseness) and 3 (Non-uniqueness) are satisfied because we assume a statistical distribution of methods and variables amongst the classes. Property 4 (Design details are important) is satisfied because the choice of methods and attributes is design implementation dependent. When two classes are combined, the number of methods and variables can never exceed that of the individual classes. The same is true for LOC. Hence, property 5 (Monotonicity) is satisfied. Consider three classes P, Q and R. Let the metric values for P and Q are the same. Also let R have common methods and variables with class P but not with class Q. Thus a combination of P and R will have a smaller metric value than a combination of Q and R. Thus, property 6 (Non-equivalence of interaction) is satisfied. Let MPC, DVN and LOC values for class P be m, v and l, for class Q be m', v' and l', and for class P+Q be m'', v'' and l''.

Because of the common methods, $m'' \leq m+m'$, $v'' \leq v+ v'$ and $l'' \leq l+l'$.
Hence property 9 (Interaction increases complexity) is not satisfied. Properties 2 (Granularity) is trivially satisfied by any metric defined for a class, so will be property 8, namely, when the name of the measured entity changes, the metric should remain unchanged.

The reason for our metric not satisfying the one property of Weyuker is that by splitting a class, there is an overall increase in the MPC, DVN and LOC value for all the sub classes created. In other words, complexity has increased.

We make certain observations from Table 6. The first three columns list out the correlation coefficient (CORRCOEF) obtained when MPC, DVN and LOC are independently related with DEV. The fourth column lists out the correlation coefficient (CORRCOEF) obtained when all the three (MPC, DVN, LOC) are combined for regression with DEV. In the first case (Data Set A), the data had been collected from a well-defined similar group of programmers (with very similar programming experiences), and the FVL metric turned out to be a better predictor of development time than MPC and slightly less predictor of development time than DVN and LOC, but still the correlation value of FVL metric is shown high. In the second case (Data Set B), since the allotted time was small (duration of laboratory exam varied from 50 minutes to a maximum of 1 hour 40 minutes), FVL metric is a better predictor of development time than DVN and LOC but less predictor of DEV than MPC. In the third and fourth cases (Data Set C and Data Set D), FVL metric has turned out to be the best predictor of development time than any of the three parameters (MPC, DVN, LOC) taken individually.

We have attempted to define a generalized metric based upon the number of functions, variables and the lines of code, to measure the complexity of a class. In the work presented here, the goal was to find the effect of the number of methods, number of distinct variables and the lines of code in a class upon the development time of the class. The approach taken was empirical. The WMC metric of C&K was used to derive at our FVL metric. The OO (Object-Oriented) language used in all the cases was C++.

The high correlation of FVL with DEV (Table 6) shows that it may be used as an effective predictor of development time. For any system, once the data for a representative set of classes have been analyzed to arrive at the FVL value, this could be used to predict the development time for similar classes.

Reusability of a given class is another attribute which needs to be investigated upon in relation to our complexity metric. The larger the number of methods in a class, the greater the potential impact on children; children inherit all the methods defined in the parent class. Classes with large numbers of methods are likely to be more application specific, limiting the possibility of reuse. The same is being studied upon as our ongoing work.

We must nevertheless mention that the programs used for the study were very small compared to large industry systems. For very large systems, the dependency of development time upon FVL and/ or any other factors needs further verification, and is also the future scope of our ongoing work.

APPENDIX

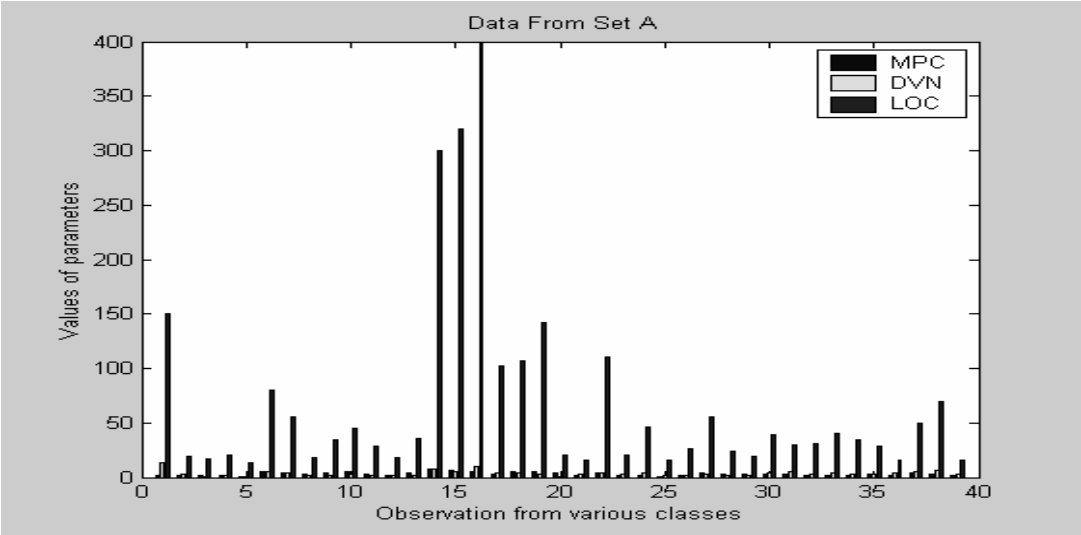

Figure 1. Observation of methods per classes, distinct variable names & Lines of code from various C++ classes for the data set A

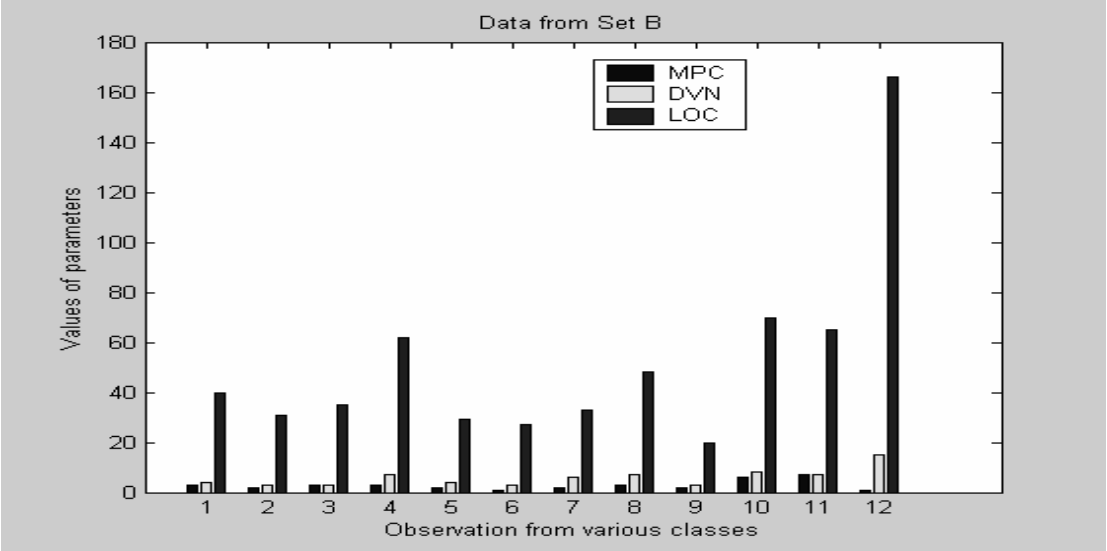

Figure 2. Observation of methods per classes, distinct variable names & Lines of code from various C++ classes for the data set B

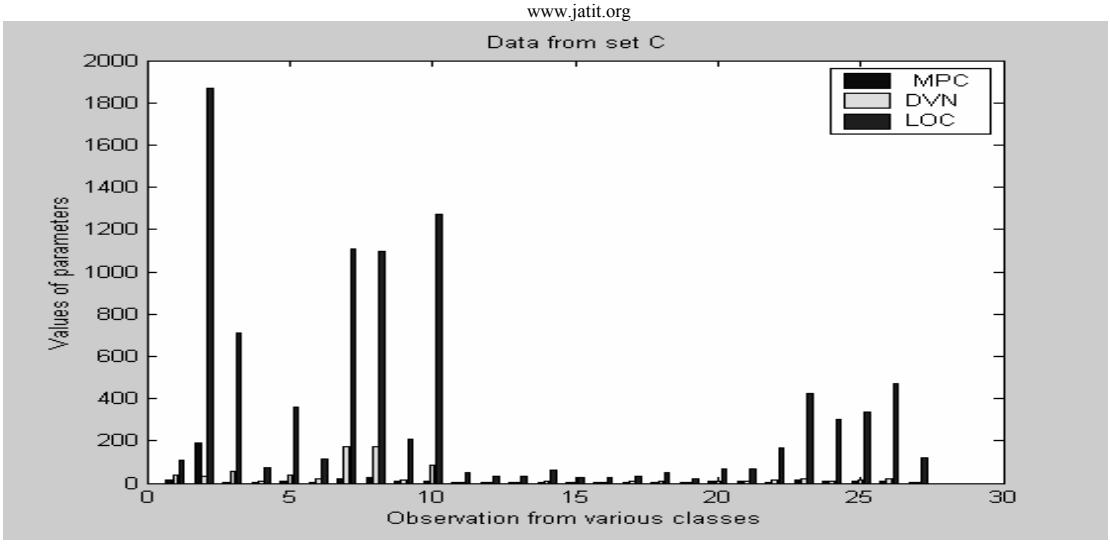

Figure 3. Observation of methods per classes, distinct variable names & Lines of code from various C++ classes for the data set C

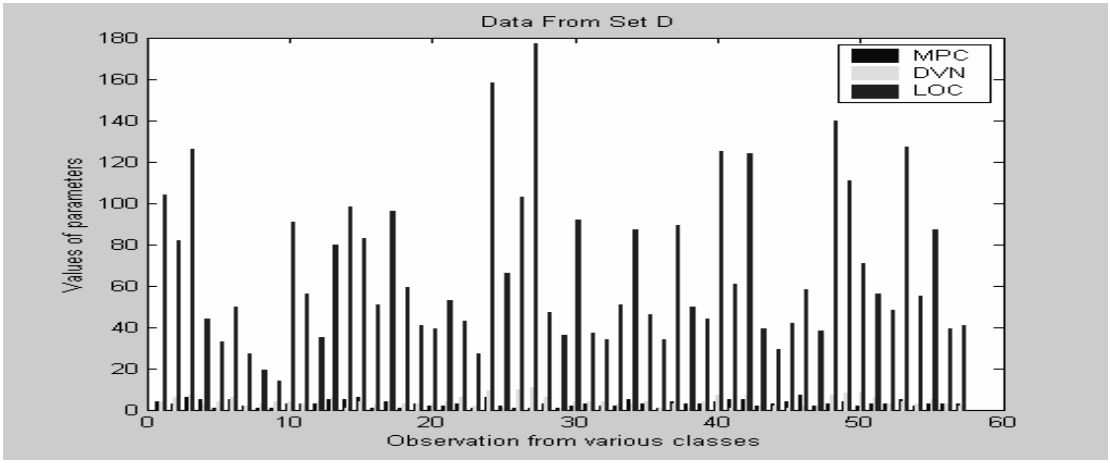

Figure 4. Observation of methods per classes, distinct variable names & Lines of code from various C++ classes for the data set D

REFERENCES

[1] 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),1043 – 1049.

[2] E. Arisholm, L. C. Briand and A. Foyen , "Dynamic Coupling Measures for Object-Oriented Software", IEEE Trans. on Software Engineering, 30, 8(2004), 491 – 506.

[3] J. M. Bieman and B. K. Kang , "Cohesion and Reuse in an Object–Oriented System", in Proceedings of Symp. Software Reliability,1995, 259 – 26.

[4] G. Booch , Object Oriented Design with Applications, Benjamin/ Cummings, Menlo Park, CA, 1995.

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

[6] L. C. Briand, J. K. Wust, J. W. Daly and D. V. Porter, "Exploring the Relationship between Design Measures and Software Quality in Object Oriented Systems", J. Systems and Software, 51, 3(2000), 245 – 273.

[7] L. C. Briand and J. K. Wust, "Modeling Development Effort in Object – Oriented Systems Using Design Properties", IEEE Trans. Software Eng., 27, 11 (2001), 963 – 986.

[8] F. Brotoeabreu, "The MOOD Metrics Set", in Proceedings of ECOOP '95 Workshop Metrics, 1995.

[9] M. Bunge, Treatise on Basic Philosophy: Ontology 1: The Furniture of the World, Boston: Riedel, 1997.

[10] H. S. Chae, Y. R. Kwon and D. H. Bae, "A Cohesion Measure for Object – Oriented Classes", Software practice and Experience, 30, 12 (2000), 1405 – 1431.

[11] 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, 30, 11 (2004) , 826 – 832.

[12] S. R. Chidamber and C. F. Kemerer, "Towards a Metrics Suite for Object Oriented Design", in Proeedings of Sixth OOPSLA Conference, 1991, 197 – 211.

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

[14] S. R. Chidamber, D. P. Darcy and C. F. Kemerer, "Managerial Use of Metrics for Object Oriented Software: An Exploratory Analysis", IEEE Trans. Software Eng., 24, 8 (1998) , 629 - 639.

[15] N. I. Churcher and M. J. Shepperd, Comments on " A Metrics Suite for Object Oriented Design", IEEE Trans. on Software Engineering, 21, (1995) , 263 – 265.

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

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

[18] B. Henderson Sellers and J. M. Edwards, Book Two of Object oriented Knowledge: The Working Object, Prentice Hall, Sydney, 1994.

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

[20] H. Kabaili, R. K. Keller and F. Lustman , "Cohesion as Changeability Indicator in Object – Oriented Systems", in Proceedings.of Fifth European Conf. Software Maintenance and Reeng,2001.

[21] W. Li and S. Henry, "Maintenance Metrics for the Object Oriented Paradigm", Proc. First Int'l. Software Metrics Symp.,(May 21-22, 1993), 52-60.

[22] M. Lorenz and J. Kidd , "Object – Oriented Software Metrics: A Practical Guide", 1994.

[23] R. Pratap, Getting Started with MATLAB – V,Oxford University Press, 1998.

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

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