I.J. Information Engineering and Electronic Business, **2014, 1, 69-76**
Published Online February 2014 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijieeb.2014.01.08

Class Complexity Metric to Predict Understandability

Kumar Rajnish
Department of Information Technology, Birla Institute of Technology, Mesra, Ranchi, India
E-mail: krajnish@bitmesra.ac.in

Abstract — This paper presents a new class complexity metric of an Object-Oriented (OO) program which is used to predict the understandability of classes. The propose complexity metric is evaluated theoretically against Weyuker's properties to analyze the nature of metric and empirically evaluated against three small projects developed by Post Graduate (PG)/Under Graduate (UG) teams. Least Square Regression Analysis technique is performed to arrive at the result and find correlation coefficient of propose metric with the Degree of Understandability. The result indicates that the propose metric is a good predictor of understandability of classes. JHAWK TOOL (Java Code Metrics Tool) were used to evaluate the parameters values involved in propose metric and for analyzing the results of projects, Matlab6.1 and IBM SPSS software were used.

Index Terms **—** Complexity, Metrics, Object-Oriented, Classes, Understandability, Methods, Instance variables.

1. Introduction

Program complexity plays an important role in the amount of time spent on development of the program. 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.

Software complexity is defined as the degree to which a system or component has a design or implementation that is difficult to understand and verify [1] i.e. complexity of a code is directly depend on the understandability. All the factors that makes program difficult to understand are responsible for complexity.

Various OO complexity and quality metrics have been proposed and their reviews are available in the literature. Rajnish et al [2] has 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. Rajnish et al [3] has proposed a complexity metric which is used to measure the complexity of class at the design stage. Kulkarni et al [4] presents a case study of applying design measures to assess software quality. Sanjay et al [5] applied their proposed metric on a real project for empirical validation and compared it with Chidamber and Kemerer metrics suites [6] and their theoretical, practical and empirical validations and the comparative study prove the robustness of the measure. Alshayeb and Li have presented an empirical study of OO metrics in two processes [7]. 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 [8]. This view is however not agreed to by Evanco [9]. Churcher et al [10] show some of the ambiguities associated with the seemingly simple concept of the number of methods per class. K. K. Agarwal et al [11] presented a set metrics which measure the robustness of the design. Koh et al [12] attempts to review the 12 OO software metrics proposed in 90s' by Chidamber and Kemerer [6] and Li [13]. Arisholm, Briand and Foyen study various Java classes to empirically evaluate the effect of dynamic coupling measures with the change proneness of classes [14]. Chae, Kwon and Bae investigated the effects of dependent instance variables on cohesion metrics for object-oriented programs [15]. They also proposed an approach to identify the dependency relations among instance variables. Liu et al [16] proposed new quality metrics that measure the method calling relationships between classes and they also conducted experiments on five open source systems to evaluate the effectiveness of the new measurement. Basilli et al [17] presents the results of study in which they empirically investigated the suite of OO design metrics introduced in [6] and their goal is to assess these metrics as predictors of fault-prone classes and determine whether they can be used as early quality indicators. Yacoub et al [18] defined two metrics for object coupling (Import Object Coupling and Export Object Coupling) and operational complexity based on state charts as dynamic complexity metrics. The metrics are applied to a case study and measurements are used to compare static and dynamic metrics. Jagdish et al [19] described an improved hierarchical model for the assessment of high-level design quality attributes in OO design. In their model, structural and behavioral design properties of classes, objects, and their relationships are

evaluated using a suite of OO design metrics. Their model relates design properties such as encapsulation modularity, coupling and cohesion to high-level quality attributes such as reusability, flexibility, and complexity using empirical and anecdotal information. Munson et al [20] showed that relative complexity gives feedback on the same complexity domains that many other metrics do. Thus, developers can save time by choosing one metric to do the work of many. Mayo et al [21] explained the automated software quality measures: Interface and Dynamic metrics. Interface metrics measure the complexity of communicating modules, whereas Dynamic metrics measure the software quality as it is executed. Sandip et al [22-23] presented in his paper to analytically evaluate against the Weyuker's property [24] and empirically validate a proposed inheritance metrics (against a three versions of the same project) that can be used to measure the quality (especially focus on the quality factors "Reuse" and "Design Complexity") of an OO systems in terms of the using class inheritance tree.

The rest of the paper is organized as follows: Section 2 presents a Weyuker's properties. Section 3 presents description of proposed metric and its analysis on data sets. Section 4 presents Conclusion and Future scope respectively.

2. Weyuker's Property

The basic nine properties proposed by Weyuker's [24] 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, $\mu(P)$ denotes the value of the metric for class *P*, and $P \equiv Q$ (*P* is equivalent to *Q*) means that two class designs, *P* and *Q*, provide the same functionality. The definition of combination of two classes is taken here to be same as suggested by [25], 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 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 vales for *P* and *Q* will be same.

Property 5. Monotonicity: For all classes *P* and *Q* the following must hold: $\mu(P) \leq \mu(P+Q)$ and $\mu(Q) \leq (P+Q)$ 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 the properties has been criticized by some researchers; however, it is widely known formal approach and serves as an important measure to evaluate metrics. In the above list however, property 2 and 8 will trivially satisfied by any metric that is defined for a class. Weyuker' 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 3 have been taken from Chidamber and Kemerer [6] regarding the distribution of methods and instance variables in the discussions for the metric properties.

***Assumption 1*:**

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

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

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

Copyright © 2014 MECS

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.

3. Propose Metric and its Analysis

3.1 Class Complexity Metric (CCM)

The metric CCM is proposed for class level and will be used in this study for predicting the understandability of classes. To calculate CCM, Total Cyclomatic Complexity (TCC) of a class, Number of Methods (NOMT) of a class, Number of Instance Variables (INST) declared, Number of External Methods (EXT) called, Number of Local Methods (LMC) called, and Total Lines of Code (NLOC) have been taken. The formula for CCM is:

$$CCM = k + w_1 * TCC + w_2 * NOMT + w_3 * INST + w_4 * EXT + w_5 * LMC + w_6 * NLOC$$

Where, the weights w_1, w_2, w_3 w_4, w_5, w_6 and the constant k are derived at by least square regression analysis.

CCM is based upon the following assumptions:

- The number of methods, number of variables, total cyclomatic complexity, total lines of code, number of external methods called, number of local methods called is predictor of understandability (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 in two different blocks is considered to have two distinct variable names.
- CCM directly relates to understandability of classes. Higher the value of CCM, less understandability (more complex) and more mental exercise is required to design and code the class and vice-versa with low CCM.

The CCM is directly related to Total Cyclomatic Complexity (TCC) of a class, Number of Methods (NOMT) of a class, Number of Instance Variables (INST) declared, Number of External Methods (EXT) called, Number of Local Methods (LMC) called, and Total Lines of Code (NLOC). So, more relation increases the understandability and a good design should have less complex classes in nature. So the objective is to find the better correlation coefficients between the number of relation and propose complexity measure. The number of relation is calculated by multiplying the Total Number of Methods in a Classes (TNMC) and the Total Number of Instance Variables in a Classes (TNVC) and named it Degree of Understandability (DU).

Based on the above fact two hypotheses has been designed to test the results:

HU_0: the positive correlation of CCM with DU increases understandability of class's i.e. direct relation with DU which is less complex in nature.

HU_1: the negative correlation of CCM with DU decreases understandability of class's i.e. inverse relation with DU which is more complex in nature.

To test these hypotheses correlation coefficient of CCM with DU has been calculated for understanding the classes in projects/ or software system.

3.2 Analytical Evaluation of CCM against Weyuker properties

From *assumption 1*, the number of methods, number of instance variables, number of external variables, total lines of code, and number of local methods called in class *P* and another class *Q* are independent and identically distributed, this implies that there is a nonzero probability that there exist *Q* such that C*CM (P)* ≠ *CCM (Q*), therefore *Property 1 (Non-coarseness*) is satisfied. Similarly, there is a nonzero probability that there exist *R* such that C*CM (P) = CCM (R).* Therefore *Property3, Non-uniqueness (notion of equivalence)* is satisfied. There is finite number of cases in the system having the same C*CM* values for classes. Since C*CM* is measured at the class level so *Property 2, Granularity* is satisfied. The choice of number of methods, number of instance variables, number of external variables, total lines of code, and number of local methods is a design decision and independent of the functionality of the class, therefore *property 4 design details matter* is satisfied. From assumptions *1, and 2* and let $CCM(P) = X_P$ and $CCM(Q) = X_Q$, then $CCM(P+Q) = X_P + X_Q - y$, where y is the number of common methods, number of common instance variables, number of common external variables, cyclomatic complexity of the common method, total lines of code, and number of local methods

between *P* and *Q*, so the maximum value of *y* is $min (X_P, X_Q)$. Therefore $CCM (P+Q) \geq X_P + X_Q - min (X_P, X_Q)$. It follows that $CCM (P+Q) \geq CCM (P)$ and $CCM (P+Q) \geq CCM (Q)$, thereby satisfying *property 5(monotonicity).* Now, let $CCM (P) = x$, $CCM (Q) = x$ and there exist a class *R* such that it has a number of common methods, number of common instance variables, number of common external variables, cyclomatic complexity of the common method, total lines of code, and number of local methods α in common with *Q* (as per assumption 1and *2)* and μ methods, variables, external variables, cyclomatic complexity, total lines of code, and number of local methods in common with *P*, where $\alpha \neq \mu$. Let $CCM (R) = r;$

$$CCM (P+R) = x + r - \mu$$

$$CCM (Q+R) = x + r - \alpha,$$

Therefore $CCM (P+R) \neq CCM (Q+R)$ and *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 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. *Property 8* is satisfied because when the name of the measured entity changes, the metric should remain unchanged. For any two classes *P* and *Q*, $X_P + X_Q - y < X_P + X_Q$ i.e. $CCM (P+Q) < CCM (P) + CCM (Q)$ for any *P* and *Q*. Therefore, property *9 (Interaction increases complexity)* is not satisfied. Table 1 presents the results of analytical evaluation of CCM against Weyuker's Property.

TABLE 1： Analytical Evaluation Results for CCM against Weyuker's Properties

Property Number	CCM
1	√
2	√
3	√
4	√
5	√
6	√
7	√
8	√
9	×
√: Metric satisfies the properties ×: Metric does not satisfy the properties	

3.3 Analysis on Data

This section presents the description of data collection, algorithm of the proposed work, summary of graphs and tables, and their interpretation.

3.3.1 Data Collection

This section presents an outline of applied approach. The variables of interest in this study are: TCC, NOMT, INST, EXT, LMC, NLOC, which is to be modeled by CCM. The above-mentioned six values were collected for classes from three different project categories. In each project categories the author had given the responsibility to the team members of each project to frame out the parameters/variables used in CCM.

The first project is related to *"Account Department"(Named it Set A).* This project had been developed by Well experienced Post Graduate (PG)/Under Graduate (UG) teams, they had developed the project in Java Language. The project involves 5 team members and containing 85 Java classes.

The second project is related to *"Bio-Technology Department"* (*named it Set B).* This project had been developed by PG teams. They have a sound knowledge of Java Programming. They had developed a small tool for the Department Research work. The project involves 2 team members and containing 20 Java classes.

The third project is related to "*Corporate Department" (named it Set C).* This project had been developed by experienced PG teams. This project had been developed in Java and for faculties for On-Line Shopping. The project involves 3 team members and containing 20 Java Classes.

3.3.2 Algorithm of Propose work

This section presents the algorithm of the proposed work which is represented with the following steps:

1. Propose Quality metric.
2. Identify Quality factors (predicting understandability of classes in software projects which is to be used in this study).
3. Collect data of three different categories (Named as Data Set A, Data Set B, and Data Set C).
4. LOOP: for each data sets perform following actions:
   a) Generate TCC, NOMT, INST, EXT, LMC, and NLOC values used in CCM using Java Tool (named JHAWK TOOL (Named JAVA CODE METRIC)
   b) Generate values for weights w1, w2, w3 w4, w5, w6 and the constant k used in CCM using Least Square Regression Analysis by MATLAB6.1 TOOL.

   END LOOP;
5. LOOP: for each data sets do the following:
   a) Find summary statistics TCC, NOMT, INST, EXT, LMC, NLOC and DU using IBM SPSS Software.

   END LOOP;
6. LOOP: for each data sets do the following:
   a) Find the Correlation Coefficients of CCM with DU and also find the Correlation Coefficients

TCC, NOMT, INST, EXT, LMC, and NLOC with DU using MATLAB6.1 TOOL.

b) Plot graph and analysis of data using IBM SPSS Software.

END LOOP;

3.3.3 *Empirical Data*

Multivariate Regression Analysis was applied on all three data sets and correlation coefficients were calculated. The summary statistics, correlation coefficients, and graphs used for CCM for three different data sets are shown (at the end of this paper in Appendix) in Table 2, Table 3, Table 4, Table 5, Table 6, Fig. 1, Fig, 2 and Fig. 3.

3.3.4 *Discussion*

The CCM has been applied to each class of three software projects. Total 125 Java classes have given as input to JHAWK tool to calculate the values of CCM, TCC, NOMT, INST, EXT, LMC, NLOC and DU for each data set. Correlation Coefficient approach was used to validate the performance of the proposed metric for predicting understandability of classes. The proposed complexity metric is directly related to TCC, NOMT, INST, EXT, LMC, NLOC and relation between them. So, more relation increases the understandability of classes and a good design should have less complex classes in nature.

Certain observations made from Table 6. The first six columns list out the correlation coefficient obtained when TCC, NOMT, INST, EXT, LMC, NLOC are independently related with DU. The seven column lists out the correlation coefficient obtained when all the six (TCC, NOMT, INST, EXT, LMC, NLOC) are combined for regression with DU. In all the cases this column entry has the highest values in each row. In the first case, the data had been collected from a well-defined similar group of PG/UG teams (with very similar programming experiences), and the CCM turned out to be a better predictor of understandability of classes. In the second case data had been collected from novice group of PG teams (with very sound knowledge of Java), CCM is turned out be good than TCC, NOMT, INST, EXT, LMC, NLOC as a predictor of DU. In the last case, since the data came from experienced PG teams and CCM turned out be a best predictor of DU.

The overall observations found that CCM has a better direct relation with DU in Data Set A and Data Set C but with Data Set B it has a direct relation with DU when combined but less direct relation with DU when measured individually. So there may be a necessity of redesign in Data Set B to predict the better understandability of classes.

4. Conclusion and Future Scope

In this paper, an attempt has been made to define new Complexity Metric CCM which is used to predict the understandability of classes in software projects. On evaluating CCM against a set of standard criteria CCM is found to possess a number of desirable properties and suggest some ways in which the OO approach may differ in terms of desirable or necessary design features from more traditional approaches. Generally, CCM satisfy the majority of the properties presented by Weyuker with one strong exception, *Property 9 (Interaction Increases Complexity).* Failing to meet *Property 9* implies that a Complexity Metric could increase rather than reduce if a class is divided into more classes. In other words complexity can increase when classes are divided into more classes.

In addition to the proposal and analytical evaluation, this paper has also presented empirical data on CCM from three software projects. All projects are developed in Java. From Table 6, it is found that the CCM is turned out to the best predictor of understandability of classes in chosen software projects.

In this study, the CCM is used for predicting the understandability of classes and through CCM one can choose to measure the same and complex design.

The future scope includes some fundamental issues:-

✓ To analyze the nature of proposed metric with performance indicators such as maintenance effort and system performance.

✓ Another interesting study would be together different complexity metrics at various intermediate stages of the project. This would provide insight into how application complexity evolves and how it can be managed/control through the use of metrics.

References

[1] IEEE Std 1061-1998.,"Standard for software Quality Metrics Methodology", *IEEE Computer society,* 1998.

[2] Rajnish. K and Bhattacherjee. V, "Complexity of class and development time: A Study", Journal of theoretical and Applied Information Technology (JATIT), Asian Research Publication Network (ARPN), Scopus (Elsevier) Index, Vol. 3, No. 1, 2006, pp. 63-70.

[3] Rajnish. K and Bhattacherjee. V, "Object-Oriented Class Complexity Metric-A Case Study", Proceedings of 5th Anuual International Conference on Information Science Technology and Management (CISTM) 2020 pennsylvania NW, Ste 904, Washington DC, publish by the Information Institute, USA, 2007, pp.36-45.

[4] Kulkarni. L, Kalshetty. R. Y and Arde. V. G, "Validation of CK metrics for Object-Oriented design measurement", proceedings of third international conf. on Emerging Trends in Engineering and Technology, IEEE Computer Soceity, 2010, pp. 646-651.

[5] Misra. S, Akman. I and Koyuncu. M, "An inheritance complexity metric for object-oriented

code:A cognitive approach", Indian Academy of Sciences, Vol. 36, Part 3, 2011, pp. 317–337.
[6] Chidamber. R. S and Kemerer. F. C, "A Metric Suite for Object-Oriented Design", IEEE Transaction on Software Engineering, Vol. 20, No. 6, 1994, pp. 476-493.
[7] Alshayeb. M and Li. W, "An Empirical Validation of Object – Oriented Metrics in Two Different Iterative Software Processes", IEEE Trans. on Software Engineering, Vol. 29, No. 11, 2003, pp.1043 – 1049.
[8] Emam. EL. K, Benlarbi. S, Goel. N and Rai. N. S, "The Confounding Effect of Class Size on the Validity of Object – Oriented Metrics", IEEE Trans. Software Eng., Vol. 27, No.7, 2001, pp. 630 – 650.
[9] Evanco. M. W, Comments on "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics", IEEE Trans. on Software Engineering, Vol. 29, No.7, 2003, pp.670 – 672.
[10] Neville. I. C, Martin. J. S, "Comment on : A Metric Suite for Object-Oriented Design", IEEE Transaction on Software Engineering, Vol. 21, No. 3, 1995, pp.263-265.
[11] Agarwal. K. K, Singh. Y, Kaur. A and Malhotra. R, "Software Design Metrics for Object-Oriented Design", Journal of Object Technology, Vol. 6, No. 1, 2006, pp. 121-138.
[12] Koh. W. T, Selamat. H. M, Ghani. A. A. A, and Abdullah. R, "Review of Complexity Metrics for Object Oriented Software Products", IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.11, 2008, pp. 317.337.
[13] Li. W, "Another Metric Suite for Object-Oriented Programming", The Journal of System and Software, Vol. 44, No. 2, 1998, pp. 155-162.
[14] Arisholm. E, Briand. C. L and Foyen. A, "Dynamic Coupling Measures for Object- Oriented Software", IEEE Trans. on Software Engineering, Vol. 30, No. 8, 2004, pp. 491 – 506.
[15] Chae. S. H, Kwon. R. Y and Bae. H. D, "Improving Cohesion Metrics for Classes by considering Dependent Instance Variables", IEEE Trans. on Software Engineering, Vol. 30, No.11, 2004, pp. 826 – 832
[16] Liu. D and Xu. S, "New Quality Metrics for Object-Oriented programs", Proceedings of Eighth ACIS Int. Conf. on Software Engineering, Artificial Intelligence, Networking, and parallel/ Distributing Computng, IEEE Computer Soceity, 2007, pp. 870-875.
[17] Basilli. R. V and Melo. W. L, "A Validation of Object-Oriented Design Metrics as Quality indicators", IEEE Transaction on Software Engineering, Vol.22, No. 10, 1996, pp.751-761.
[18] Yacoub. S, Robinson. T and Ammar. H. H, "Dynamic Merics for Object-Oriented Design", Proceedings of 6th International Conf. on Software Metrics Symposium, 1999, pp. 50-61.
[19] Bansiya. J and Davis. C. G", A Hierarchical Model for Object-Oriented Design Quality Assessment", IEEE Transaction on Software Engineering, Vol. 28, No. 1, 2002, pp. 4-17.
[20] Munson. C. J and Khoshgoftaar. M. T", Measuring Dynamic program Complexity", IEEE Software, Vol. 9, No. 6, 1992, pp. 48-55.
[21] Mayo. A. K, Wake. S. A and Henry. S. M", Static and Dynamic Software Quality Metric tools", Department of Computer Science, Virginia Tech, Blacksburg, Technical Report, 1990.
[22] Mal. S and Rajnish. K, "Applicability of Weyuker's Property 9 to Inheritance Metric", International Journal of Computer Application", Foundation of Computer Science, USA, Vol. 66, No.12, 2013, pp.21-26
[23] Mal. S and Rajnish. K, " New Quality Inheriatnce Metrics for Object-Oriented Design", International Journal of Software Engineering and its Application, SERSC, Scopus (Elsevier), Vol. 7, No. 6, pp. 185-200, November 2013.
[24] Weyuker. J. E, "Evaluating Software Complexity Measures", IEEE Trans. on Software Engineering, Vol.14, 1998, pp.1357-1365.
[25] Abreu. B and Melo. W, "Evaluating the Impact of OO Design on Software Quality", presented at Third International Software Metrics Symposium, Berlin, 1996.

Appendix

TABLE 2：Summary Statistics for the Data Set A

	Minimum	Maximum	Mean	Std. Deviation
TCC	1.00	16.00	7.1294	3.18773
NOMT	1.00	10.00	3.1647	1.12172
INST	0.00	34.00	7.7765	6.34218
EXT	6.00	51.00	25.6941	9.65232
LMC	0.00	1.00	0.2353	0.42670
NLOC	30.00	157.00	67.5059	20.85587
DU	0.00	120.00	26.5294	22.57453

Copyright © 2014 MECS

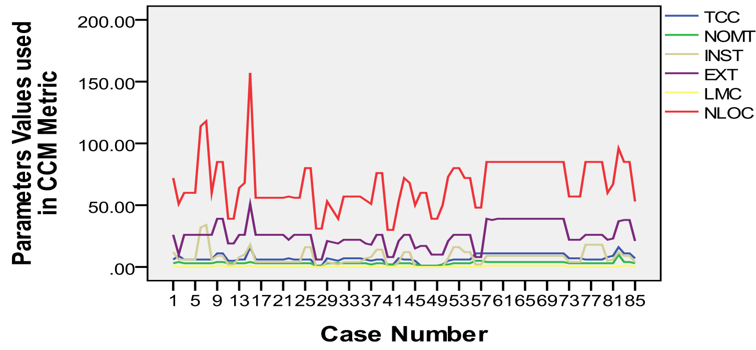

Figure 1: Parameters Values used in CCM for the Data Set A

TABLE 3： Summary Statistics for the Data Set B

	Minimum	Maximum	Mean	Std. Deviation
TCC	0.00	59.00	17.30	16.89628
NOMT	0.00	48.00	10.95	11.17080
INST	0.00	12.00	2.20	2.83957
EXT	0.00	43.00	6.70	11.61261
LMC	0.00	8.00	0.90	2.14966
NLOC	7.00	400.00	88.050	100.47910
DU	0.00	60.00	17.150	20.75490

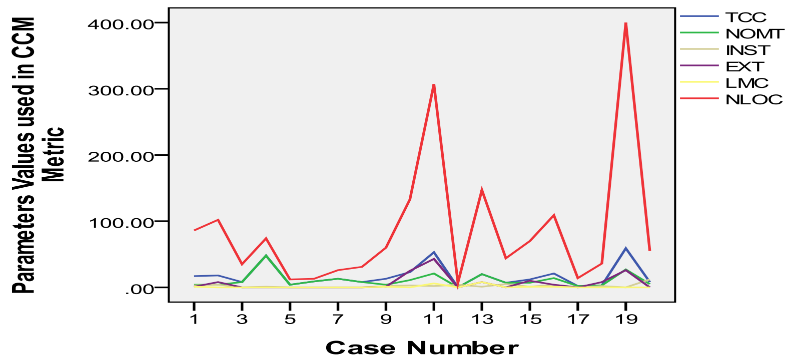

Figure 2: Parameters Values used in CCM for the Data Set B

TABLE 4： Summary Statistics for the Data Set C

	Minimum	Maximum	Mean	Std. Deviation
TCC	1.00	21.00	5.1111	5.77916
NOMT	1.00	15.00	4.3889	4.48709
INST	0.00	17.00	1.8889	4.39102
EXT	0.00	39.00	3.5556	9.31932
LMC	0.00	2.00	0.500	0.85749
NLOC	4.00	172.00	26.6111	42.18048
DU	0.00	204.00	23.0556	54.46043

Copyright © 2014 MECS

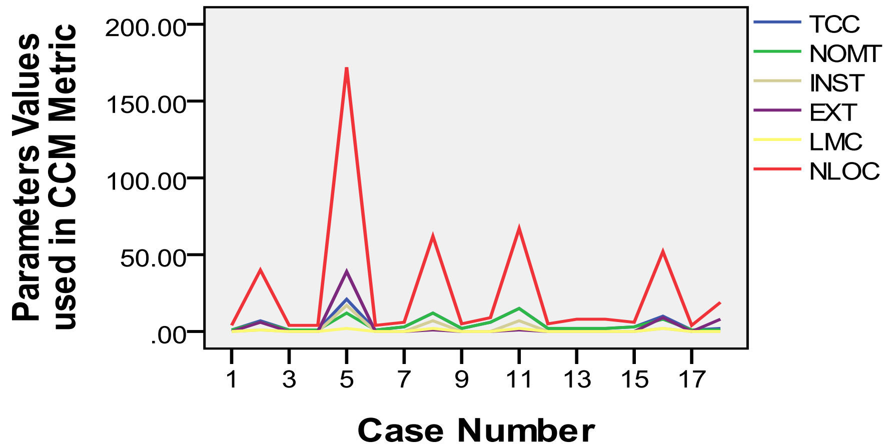

Figure 3: Parameters Values used in CCM for the Data Set C

Table 5: Values of the coefficients for the six independent variables and constant used in CCM from three different data sets by Least Square Regression Analysis

	W_1	W_2	W_3	W_4	W_5	W_6	k
SET A	-1.9865	9.7890	3.2926	-0.4374	17.4461	-0.1164	0
SET B	0.4253	0.6650	5.2092	0.6272	1.5007	-0.1403	0.0007
SET C	-6.3444	7.4854	15.2038	1.4431	-2.8707	-0.3579	0

Table 6: Correlation Coefficient with respect to DU for the three different Data Sets

	TCC	NOMT	INST	EXT	LMC	NLOC	CCM
SET A	0.5183	0.5917	0.9006	0.5187	0.2786	0.8513	0.9586
SET B	0.3533	0.3934	0.6701	0.2146	0.2198	0.2112	0.9085
SET C	0.9248	0.8121	0.9950	0.7939	0.7766	0.9567	0.9980

Author Profile

Kumar Rajnish: He is an Assistant Professor in the Department ofInformation Technology at Birla Institute of Technology, Mesra, Ranchi, Jharkahnd, India. He received his PhD in Engineering from Birla Institute of Technology Mesra, Ranchi, Jharkhand, India in the year of 2009. He received his MCA Degree from Madan Mohan Malaviya Engineering College, Gorakhpur, State of Uttar Pradesh, India in the year of 2001. He received his B.Sc Mathematics (Honours) from Ranchi College Ranchi, India in the year 1998. He has 28 International and National Research Publications. His Research area is Object-Oriented Metrics, Object-Oriented Software Engineering, Software Quality Metrics, Programming Languages, and Database System