Measuring System Complexity Using New Complexity Metric

[1]Sandip Mal, [2]Kumar Rajnish

Department of computer Science and Engineering, Bit, Mesra, Ranchi, India-835215

[1]Sandip.mal1987@gmail.com, [2]krajnish@bitmesra.ac.in

Abstract - **This paper presents a new complexity metric for Object-Oriented (OO) design that helps the researchers, practitioners, and designers to measure the system complexity of a software system. Theoretical validation of new complexity metric has been done using Weyuker's properties. This paper also presents a better correlation of the new proposal comparatively with some existing complexity metrics to show better prediction of Complexity (Reusability, Understandability, Design Complex). A statistical analysis of two open source system (JFreeChart, JEdit) has been evaluated.**

Keywords - **Attributes, Classes, Complexity, Methods, Metrics, Object-Oriented.**

1. INTRODUCTION

The development of a large software system is a time and resource-consuming activity. Even with the increasing automation of software development activities, resources are still scarce. Therefore, we need to be able to provide accurate information and guidelines to managers to help them make decisions, plan and schedule activities, and allocate resources for the different software activities that take place during software development. Software metrics are, thus, necessary to identify where the resources are needed; they are a crucial source of information for decision making (Harrison, 1994). For measuring the complexity of large systems is one example of a resource and time consuming activities. 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 (IEEE Std, 1998) 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. Software complexity is an estimate of the amount of effort needed to develop, understand or maintain the code. It follows that more complex the code is the higher the development effort and development time needed to develop or maintain this code. Software quality is the degree to which software possesses a desired combination of quality attributes (IEEE Std, 1998). The purpose of software metrics is to make assessments throughout the software life cycle as to whether the software quality requirements are being met. The use of software metrics reduces subjectivity in the assessment and control of software quality by providing a quantitative basis for making decisions about software quality. Design Complexity, understandability, and Reusability depends on the classes, methods of a class, super class etc (Jagdish, 2002, Munson et al, 1992). More classes or methods indicates complex design, more effort required to understand the design and less use of modules in next version. The metric WMC (Chidamber et al, 1994) and CC (Class Complexity) (Balasubramanian, 1996) have been used for comparative study with CM. The use of WMC and CC metrics are more acceptable in literature, and industry. So comparing CM with CC and WMC is more acceptable and useful. Weyuker (Weyuker, 1998) proposed nine important properties for validating metrics. Most of the metric in the literature have been validated using Weyuker's property. CM metric is also validated using Weyuker's nine properties for acceptning the metric in literature and research purpose.

Various OO complexity and quality metrics have been proposed and their reviews are available in the literature. (Rajnish et al, 2006a, 2007a, 2007b) 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. (Kulkarni et al, 2010) presents a case study of applying design measures to assess software quality. (DapengLiu et al, 2007) 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. Victor (Victor, 1996) presents the results of study in which they empirically investigated the suite of OO design metrics introduced in (Chidamber et al, 1994) 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, 1999) 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, 2002) described an improved hierarchical model for the assessment of high-level design quality attributes in OO design. This paper shows how quality factors are affected by the various metrics. 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. (Mayo et al, 1990) 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. (Sastry et al, 2011) presents that metrics have been used to analyze various features of software component. (Munson et al, 1990) describes the application of a Relative Complexity Metric for Software Project Management. Complexity of methods involved is a predictor of how much time, effort, cost is required to develop and maintain the class. If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes more complicated since it requires a greater level of understanding on the part of the tester.

The rest of the paper is organized as follows: Section 2 deals with the Weyuker's Properties and limitation of these properties and brief description of existing OO complexity metrics for comparative study with other metrics. Section 3 presents proposed complexity metric and its analytical evaluation against Weyuker's Property. Section 4 presents methodology of work and results. Section 5 presents conclusion and future scope respectively.

2. WEYUKER PROPERTIES AND EXISTING METRICS

2.1 Weyuker's Property

The basic nine properties proposed by Weyuker (Weyuker, 1998) are listed below. The notations used are as follows: P, Q, and R denote combination of classes ∑ses 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.

TABLE 1: WEYUKER PROPERTY

No	Name	Description
1	Noncoarseness	$(\exists P)(\exists Q)(\mu(P) \neq \mu(Q))$
2	Granularity	Let c be a non-negative number. Then there are finitely many programs of complexity c
3	Non-Uniqueness	There are distinct programs P and Q such that $\mu(P) = \mu(Q)$
4	Design Details Matter	$(\exists P)(\exists Q)(P \equiv Q)$ and $\mu(P) \neq \mu(Q)$
5	Monotonicity	For all classes P and Q such that $\mu(P) \leq \mu(P+Q)$ and $\mu(Q) \leq \mu(P+Q)$
6	Non-equivalence of interaction	$(\exists P)(\exists Q)(\exists R)$ such that $\mu(P) = \mu(Q)$ does not imply that $\mu(P + R) = \mu(Q + R)$
7	Interaction among statements	Not consider for Object-Oriented metrics.
8	No change on renaming	If P is a remaining of Q then $\mu(P) = \mu(Q)$
9	Interaction increases complexity	$(\exists P)(\exists Q)(\mu(P) + \mu(Q) < \mu(P+Q)$

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.

2.2 Weighted Method Per Class (WMC) Metric Of Chidamber And Kemerer

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

Then,

$$\text{WMC (Weighted Method per Class)} = \sum_{i=1}^{n} c_i$$

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

Theoretical Basis. WMC metric relates directly to complexity of an individual as defined by Bunge as the "numerosity of its composition" (Bunge, 1997). Thus it can be said that the complexity of an object is the cardinality of its set of properties. In OO terminology, the properties of an object include the attributes and its methods. As mentioned in (Chidamber et al, 1994), 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 an attribute of the class since attributes correspond to properties. They further mention that the number of attributes 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.

2.3 Class Complexity (CC) Metric of Balasubramanian

In Balasubramanian's CC (Class Complexity) metric (Balasubramanian, 1996), Class Complexity is calculated as the sum the number of instance variables in a class and the sum of the weighted static complexity of a local method in the class. To measure the static complexity Balasubramanian uses McCabe's Cyclomatic Complexity (McCabe, 1976) where the weighted result is the number of nodes subtracted from the sum of the number of edges in a program flow graph and the number of connected components.

$$CC= \sum_{i=0}^{n} INST(i) + CyclomaticComplexity(i)$$

3. PROPOSED METRICS AND ITS ANALYTICAL EVALUATION AGAINST WEYUKER'S PROPERTY

From literature survey , one can say complexity of a class depends on method, instance variable, no of class related with the class etc. Based on this the Complexity Metric (CM)has been proposed for measuring complexity of a software system and is defined as follows:

CM= NOM + INST + EXT + NSUP + TCC + NSUB

Where,

NOM = Number of methods in a class.
INST = Number of instance variables in a class.
EXT = Number of external variable in a class.
NSUP = Number of super class of a class.
TCC = Total Cyclomatic Complexity of a Class
NSUB = Number of subclass of a class.

Complexity metric CM is based upon the following intuitive ideas:

- Number of methods, number of instance variables, number of external variables, number of super class, number of subclass, and total Cyclomatic Complexity in a class is important for the creation of the class in an Object-Oriented Design (OOD) because the complexity of the information hiding gives an indication of the amount of time needed to design, and implement the class.

- CM directly relates to the time needed to design and implement a class. Higher the value of CM, the more the mental exercise is required to design and code the class and vice versa with low CM.

- Larger the number of methods in a class the greater the potential impact on the subclass, since subclass will inherit methods and variables defined in the superclass.

3.1 Theoretical Evaluation of CM against Weyuker's Property

From assumption 1, the number of methods, number of instance variables, number of external variables 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 CM (P) ≠ CM (Q), therefore Property 1 (Non-coarseness) is satisfied. Similarly, there is a nonzero probability that there exist R such that CM (P) = CM (R). Therefore Property, Non-uniqueness (notion of equivalence) is satisfied. There is finite number of cases in the system having the same CM values for classes. Since 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, number of super classes, and number of subclasses is a design decision and independent of the functionality of the class, therefore property 4 design details matter is satisfied. From assumptions 1, 2, and 3 and let CM (P) = X_P and CM (Q) =XQ, then CM (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, same number of subclasses at any level between P and Q, so the maximum value of y is min (X_P, X_Q). Therefore CM (P+Q) ≥ $X_P + X_Q - \min(X_P, X_Q)$. It follows that CM (P+Q) ≥ CM (P) and CM (P+Q) ≥CM (Q), thereby satisfying property 5(monotonicity).Now, let CM (P) = x, CM (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, same number of subclasses at any level α in common with Q (as per assumption 1, 2 and 3) and μ methods, variables, external variables, Cyclomatic Complexity, same number of subclasses at any level in common with P, where$\alpha \neq \mu$. Let CM (R) = r;

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

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

Therefore CM (P+R) $\neq$CM (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 8is 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. CM (P+Q) < CM (P) + CM (Q) for any P and Q. Therefore, property 9 (Interaction increases complexity) is not satisfied.

4. METHODOLOGY AND RESULTS

4.1 Data Collection and flow of proposed quality model

We have collected two open source software system (JFreeChart, JEdit) for validating and analyzing proposed complexity metric [22] [23]. JHawk automated tool has been used to find out NOM, INST, EXT, NSUP, TCC, NSUB.

4.2 Algorithm

Proposed work of finding Quality of a software system has been given in the following steps:

```
Step 1 : Propose Quality Metric.
Step 2 : Identify Quality factors to be effected.
Step 3 : Collect data of open source software system.
Step 4 : LOOP: every class of every version of JFreeChart
         and JEdit
         Generate metric value of CM, CC, WMC using
         JHawk automated tool
         End LOOP
Step 5 : LOOP: every version of JFreeChart and JEdit
         Find Average Quartile (Average of 25%, 50%,
         75%), Mean, and Median.
         End LOOP
Step 6 : LOOP: every Software System
         a) Find the R2, Spearman Correlation, of CM,
            CC, WMC with no of classes for three cases
            (Average Quartile, Mean, Median).
         b) Plot graph and Analysis of statistical data
            End LOOP
```

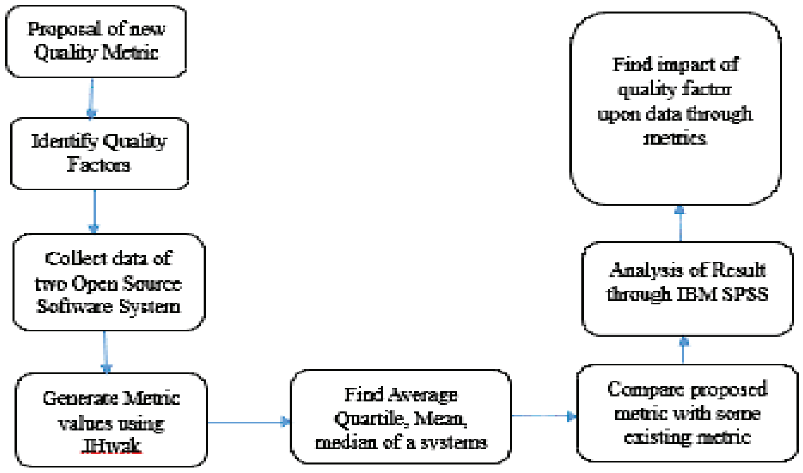

Proposal of new Quality Metric
Identify Quality Factors
Collect data of two Open Source Software System
Generate Metric values using JHwak
Find Average Quartile, Mean, median of a systems
Compare proposed metric with some existing metric
Analysis of Result through IBM SPSS
Find impact of quality factor upon data through metrics

FIGURE 1: DATA FLOW DIAGRAM OF PROPOSED WORK

4.3 Empirical Data

The R2 (Co-efficient), spearman correlation, summary statistics, and graph for two systems (JFreeChart, JEdit) are shown in Table 2, Table 3, Table 4, Table 5 and Table 6, Table 7 and Figure 2, Figure 3, Figure 4, Figure 5, Figure 6, Figure 7.

TABLE 2: R2 VALUES FOR COMPLEXITY MEASURE OF JFREECHART

	CM	**CC**	**WMC**
Average Quartile	0.897	0.814	0.915
Mean	0.913	0.626	0.774
Median	0.903	0.709	0.473

TABLE 3: R2 VALUES FOR COMPLEXITY MEASURE OF JEDIT

	CM	**CC**	**WMC**
Average Quartile	0.812	0.625	0.585
Mean	0.883	0.915	0.879
Median	0.750	0.750	0.107

TABLE 4: SPEARMAN CORRELATION VALUES FOR COMPLEXITY MEASURE OF JFREECHART

	CM	CC	WMC
Average Quartile	0.947	0.902	0.903
Mean	0.958	0.807	0.884
Median	0.95	0.847	0.688

TABLE 5: SPEARMAN CORRELATION VALUES FOR COMPLEXITY MEASURE OF JEDIT

	CM	CC	WMC
Average Quartile	0.901	0.791	0.362
Mean	0.945	0.956	0.943
Median	0.866	0.866	0.354

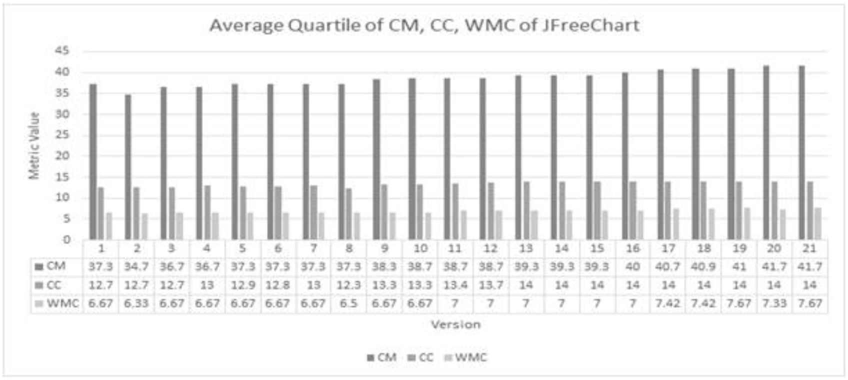
TABLE 6: CM METRIC VALUES OF JEDIT

Version	**No of Classes**	**No of Classes Remove**	**No of Classes added**	**Complexity Metric(CM)**		
				Average Quartile	**Mean**	**Median**
4.3.0 pre4	957			19.66667	38.97811	15
4.3.0 pre7	1054	0	97	19.66667	39.19548	15
4.3.0 pre11	1120	0	66	19.75	39.23463	15
4.3.0 pre 14	1159	0	39	**19.66667**	**39.33679**	**15**
4.5.1	1237	0	78	20.33333	40.55496	16
4.5.2	1239	0	2	20.66667	40.96581	16
5.0.0	1280	0	41	20.66667	41.74791	16
	Total Classes added finally **323**	**0**	**323**	**1**	**2.7698**	**1**

TABLE 6: CM METRIC VALUES OF JEDIT

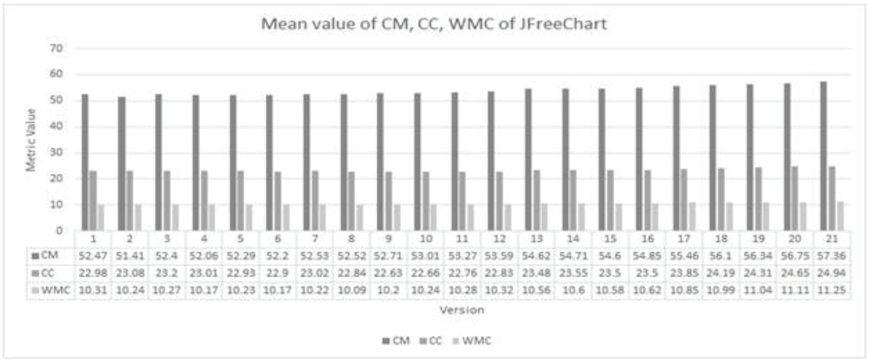
Version	**No of Classes**	**No of Classes Remove**	**No of Classes added**	**Complexity Metric(CM)**		
				Average Quartile	**Mean**	**Median**
1.0.0	774			37.33333	52.46572	35
1.0.0 pre1	668	106	0	**34.66667**	**51.41339**	**33**
1.0.0 pre2	711	0	43	36.66667	52.40141	34
1.0.0 rc1	759	0	48	36.66667	52.06069	35
1.0.0 rc2	775	0	16	37.33333	52.28811	35
1.0.0 rc3	777	0	2	37.33333	52.20232	35
1.0.1	776	1	0	37.33333	52.53032	35
1.0.2	832	0	56	37.33333	52.52106	35
1.0.3	914	0	82	38.33333	53.2895	36
1.0.4	946	0	32	38.66667	53.464	36
1.0.5	961	0	15	38.667	53.669	36
1.0.6	976	0	15	38.66667	53.98667	36
1.0.7	987	0	11	39.33333	54.41663	37

1.0.8	989	0	2	39.33333	54.71053	37
1.0.8 a	992	0	3	39.33333	54.6004	37
1.0.9	992	0	0	40	54.84763	37
1.0.10	1009	0	17	40.66667	55.36329	38
1.0.11	1029	0	20	40.91667	55.80019	38
1.0.12	1033	0	4	41	56.33915	38
1.0.13	1064	0	31	41.66667	56.74976	38
1.0.14	1081	0	17	41.66667	57.35556	38
	Total Classes added finally **307**	**107**	**414**	**3.3334**	**1.333**	**3**



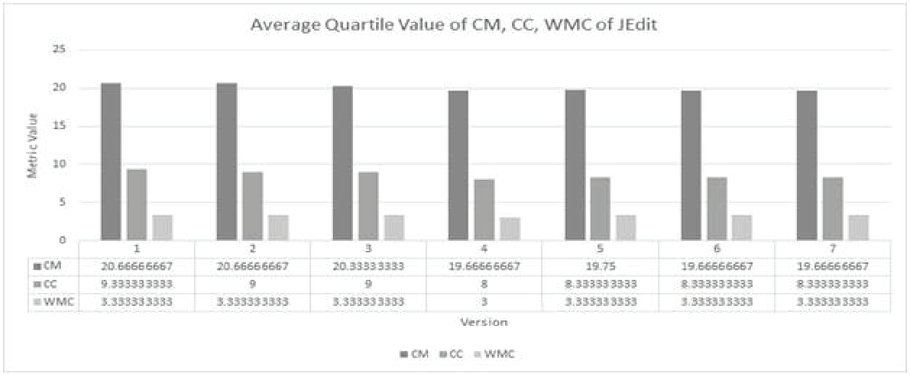
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
CM	37.3	34.7	36.7	36.7	37.3	37.3	37.3	37.3	38.3	38.7	38.7	38.7	39.3	39.3	39.3	40	40.7	40.9	41	41.7	41.7
CC	12.7	12.7	12.7	13	12.9	12.8	13	12.3	13.3	13.3	13.4	13.7	14	14	14	14	14	14	14	14	14
WMC	6.67	6.33	6.67	6.67	6.67	6.67	6.67	6.5	6.67	6.67	7	7	7	7	7	7	7.42	7.42	7.67	7.33	7.67

FIGURE 2: BAR CHART OF AVERAGE QUARTILE VALUES OF CM, CC, AND WMC METRICS OF JFREECHART



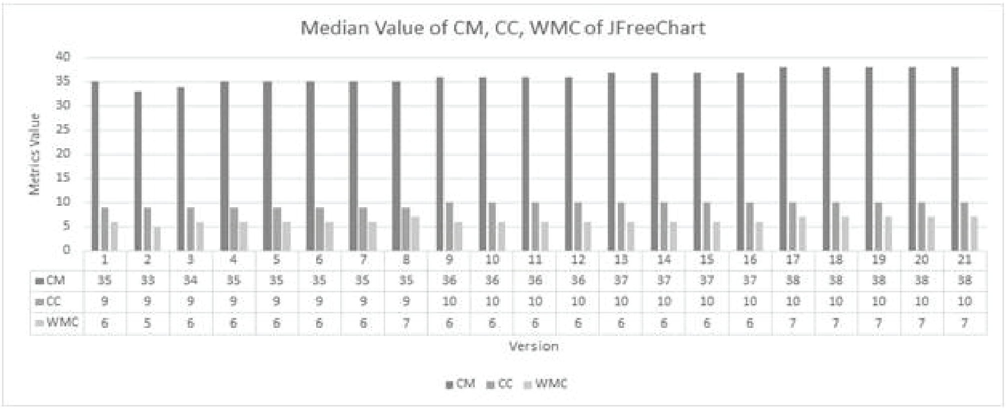
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
CM	52.47	51.41	52.4	52.06	52.29	52.2	52.53	52.52	52.71	53.01	53.27	53.59	54.62	54.71	54.6	54.85	55.46	56.1	56.34	56.75	57.36
CC	22.98	23.08	23.2	23.01	22.93	22.9	23.02	22.84	22.63	22.66	22.76	22.83	23.48	23.55	23.5	23.5	23.85	24.19	24.31	24.65	24.94
WMC	10.31	10.24	10.27	10.17	10.23	10.17	10.22	10.09	10.2	10.24	10.28	10.32	10.56	10.6	10.58	10.62	10.85	10.99	11.04	11.11	11.25

FIGURE 3: BAR CHART OF MEAN VALUES OF CM, CC, AND WMC METRICS OF JFREECHART



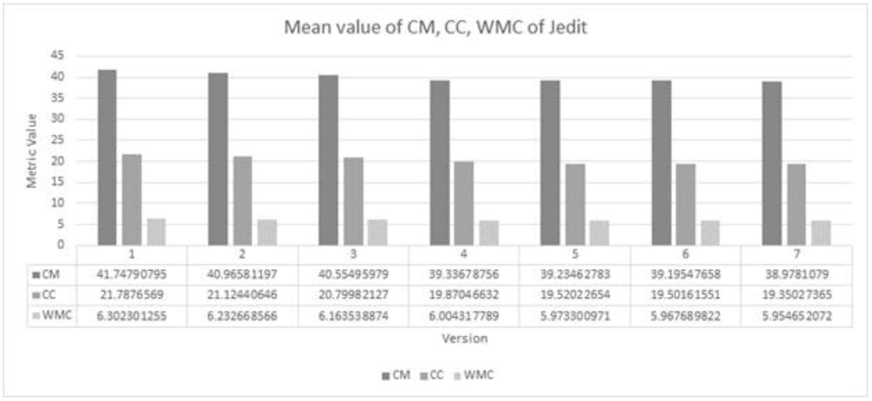
	1	2	3	4	5	6	7
CM	20.66666667	20.66666667	20.33333333	19.66666667	19.75	19.66666667	19.66666667
CC	9.333333333	9	9	8	8.333333333	8.333333333	8.333333333
WMC	3.333333333	3.333333333	3.333333333	3	3.333333333	3.333333333	3.333333333

FIGURE 4: BAR CHART OF MEDIAN VALUES OF CM, CC, AND WMC METRICS OF JFREECHART



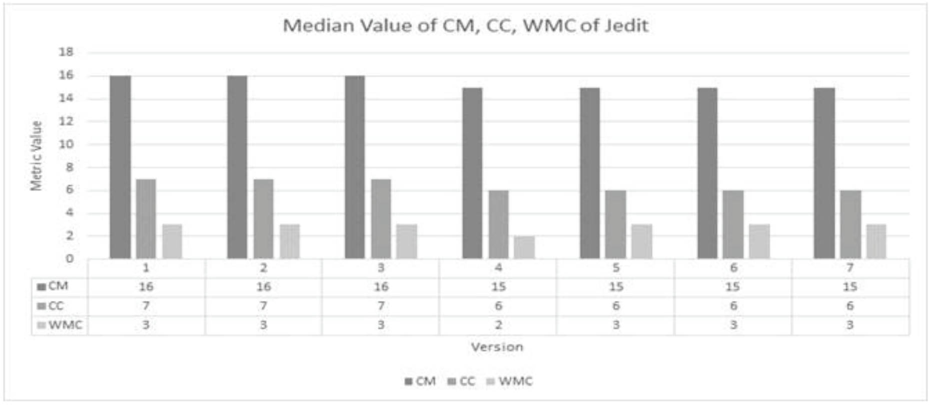
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
CM	35	33	34	35	35	35	35	35	36	36	36	36	37	37	37	37	38	38	38	38	38
CC	9	9	9	9	9	9	9	9	10	10	10	10	10	10	10	10	10	10	10	10	10
WMC	6	5	6	6	6	6	6	7	6	6	6	6	6	6	6	6	7	7	7	7	7

FIGURE 5: BAR CHART OF AVERAGE QUARTILE VALUES OF CM, CC, AND WMC METRICS OF JEDIT



	1	2	3	4	5	6	7
CM	41.74790795	40.96581197	40.55495979	39.33678756	39.23462783	39.19547658	38.9781079
CC	21.7876569	21.12440646	20.79982127	19.87046632	19.52022654	19.50161551	19.35027365
WMC	6.302301255	6.232668566	6.163538874	6.004317789	5.973300971	5.967689822	5.954652072

FIGURE 6: BAR CHART OF MEAN VALUES OF CM, CC, AND WMC METRICS OF JEDIT



	1	2	3	4	5	6	7
CM	16	16	16	15	15	15	15
CC	7	7	7	6	6	6	6
WMC	3	3	3	2	3	3	3

FIGURE 7: BAR CHART OF MEDIAN VALUES OF CM, CC, AND WMC METRICS OF JEDIT

4.4 Discussion

The coefficient of determination R2 of metrics value with no of classes added or removed, provides a measure of how much of the variation of complexity by various versions. Table 2 and Table 3 display the values of R2 obtained for each of the complexity measures on all versions of JFreeChart and JEdit. In each case, CM gives the largest value of R2, which indicates that it is the better linear complexity measure of a system as compare to other complexity metrics (CC, WMC). Though R2 indicates that CM is a good complexity measure of a system, the Spearman rank correlation coefficients between the rankings determined by versions of JFreeChart, JEdit have been computed in Table 4 and Table 5 and these results also provides a strong indication that CM is better predictors of Complexity of a system than the alternatives. This study shows that if number of class increases then CM also increases and there is a possibility of increasing design complexity. So it is hard to understand the design also. Increasing of CM denotes CM factors like NOM, INST, EXT, TCC, NSUB have increased and the class is less reused.

From table 7 it is observed that out of 21 version 17 version added new class. Average Quartile value of CM has increased in 11 versions which indicates 65 % versions added new classes and also increases complexity value. Remaining 35% of the versions added very few classes and introducing new classes does not affect the complexity value. Finally in 21 version total classes added is 307. Complexity increases 3.334(Average Quartile), 1.333(Mean), 0.999(Median). On an average one can say adding every 15 classes, complexity increase 0.15(Average Quartile), 0.07(Mean) and 0.05(Median). It is also observed that removing 106 classes from 1.0.0, Average quartile complexity decrease 2.67 (80%), mean decreases 1.052(79%) and median decrease (67%) the maximum decrease of complexity as well as maximum removing of classes. Maximum complexity increases in version 1.0.0 pre2 which has been made from version 1.0.0pre1, which has minimum classes and also made from a version with maximum removing of classes. Beside version 1.0.0 pre1, maximum class added in version 1.0.2 is 82 and Average quartile complexity increases 1(30%), mean increases 0.77(58%), median increases 1(33.33) the maximum increasing classes as well as introducing maximum new classes.

From table 6, it is observed that out of 7 versions all versions added new class. Average Quartile value of complexity metric has decreased in version 4.3.0 pre 14, although classes have been increased. One can say that version 4.3.0 pre 14 is the maturity level. Finally in 7 version total classes added are 323. Complexity increases 1(Average Quartile), 2.7698(Mean), 1(Median). On an average one can say that adding every 46 class, complexity increase 0.14(Average Quartile), 0.396(Mean) and 0.14(Median).

5. CONCLUSION AND FUTURE SCOPE

In this paper, an attempt has been made to define new Complexity Metric (CM) to measure the Complexity of a software system at system level. On evaluating CM against a set of standard criteria CM 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 CM satisfy the majority of the properties presented by Weyuker (Weyuker, 1998) 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 CM along with CC, WMC from two open source systems. Both systems are developed in Java. From Table 2 and Table 3 it is found that the values of R2 obtained for each of the Complexity measures of all versions of the JFreeChart and JEdit and in case CM gives the largest value of R2 which indicates that CM is the best linear Complexity Measure of a system. From Table 4 and Table 5 it is seen from results, which provide a strong indication that CM is better prediction of complexity of a system than the alternatives.

In This study, the CM is used for predicting the possibility of reuse a class in a large system, how much easy to understand and how much complex the design. Through CM one can chose to measure reusability, understandability and complex design as simply the number of classes that were added, modified or deleted in order to release its next version. The more classes required denotes the lower more design complex and harder to understand. More CM of a class denotes less reuse of the class.
The future scope includes some fundamental issues:-

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

2. 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] N.V. Bala Subramanian, "Object-Oriented Metrics", Asian Pacific Software Engineering Conference (APSEC-96), pp. 30-34, 1996.

[2] M. Bunge, "Treatise on Basic Philosophy Ontology1", The Furniture of the World, Boston:Reidel, 1997.

[3] S.R. Chidamber and C.F. Kemerer, "A Metric Suite for Object-Oriented Design", IEEE Transaction on Software Engineering, Vol. 20, No. 6, pp. 476-493, 1994.

[4] D. Liu and S. Xu, "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, pp. 870-875, 2007.

[5] W. Harrison, "Software Measurement: A Decision-Process approach", Advances in Computers, Vol. 39, pp 51-105, 1994.

[6] IEEE Std 1061-1998.:"Standard for software Quality Metrics Methodology", IEEE Computer Society, 1998.

[7] J. Bansiya and C. Davis Carl, "A Hierarchical Model for Object-Oriented Design Quality Assessment", IEEE Transaction on Software Engineering, Vol. 28, No. 1, pp. 4-17, 2002.

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

[9] A. Mayo Kevin, A. Wake Steven and M. Henry Sallie, " Static and Dynamic Software Quality Metric tools", Department of Computer Science, Virginia Tech, Blacksburg, Technical Report, 1997.

[10] T.J. McCabe, "A Complexity Measure", IEEE Transaction on Software Engineering, Vol. 2, pp. 308-320, 1976.

[11] C. John Munson and T.M. Khoshgoftaar, "Measuring Dynamic program Complexity", IEEE Software, Vol. 9, No. 6, pp. 48-55, 1992.

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

[13] K. Rajnish and V. Bhattacherjee, "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, pp.36-45, 2007.

[14] K. Rajnish and V. Bhattacherjee, "Class Cohesion: An Empirical and Analytical Approach", International Journal of Science and Research (IJSR), Victoria, Australia, Vol. 2, No. 1, pp. 53-62, 2007.

[15] K. Rajnish, A.K. Choudhary, and A.M. Agrawal, "Inheritance Metrics for Object-Oriented Design", International Journal of Computer Science and Information Technology (IJCSIT), AIRCC, Vol. 2, No. 6, pp. 13-26, 2010.

[16] http://www.msquaredtechnologies.com/m2rsm/rsm_demo.php. . (Accessed 11 January 2013)

[17] J.S.V.R.S. Sastry, K.V. Ramesh and M. Padmaja, "Measuring Object-Oriented Systems based on the Experimental Analysis of the Complexity Metrics", International Journal of Engineering Science and Technology (IJEST), Vol. 3, No. 5, 2011.

[18] http://www.stan4j.com. (Accessed 11 January 2013)

[19] V.R. Basilli and W.L. Melo, "A Validation of Object-Oriented Design Metrics as Quality indicators", IEEE Transaction on Software Engineering, Vol.22, No. 10, pp.751-761, 1996.

[20] S. Yacoub, T. Robinson and H. Hany Ammar, "Dynamic Metrics for Object-Oriented Design", proceedings of 6th International Conf. on Software Metrics Symposium, pp. 50-61, 1999.

[21] E.J. Weyuker , "Evaluating Software Complexity Measures", IEEE Trans. on Software Engineering, Vol. 14, PP: 1357-1365, 1998.

[22] http://sourceforge.net/projects/jfreechart/files/1.%20JFreeChart (Accessed 11 January 2013).

[23] http://sourceforge.net33/projects/jedit/files/jedit (Accessed 11 January 2013).

[24] 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, pp. 253-258, 2006.

[25] C. Munson and T.M. Khoshgoftaar, "Application of a Relative Complexity Metric for Software Project Management", Journal of System Software, 12, pp. 283 – 291, 1990.

ABOUT THE AUTHORS

Mr. Sandip Mal received B.Tech degree in the department of Computer Science and Engineering from West Bengal University of Technology in the year 2008. He has also Completed ME (Software Engineering) from Birla Institute of Technology, Mesra, Ranchi, Jharkhand, India in the year of 2012. Currently, he is pursuing Ph.D. on Software Quality Metrics from Birla Institute of Technology, Mesra, Ranchi, Jharkhand, India. His Research area is Object-Oriented Metrics, Software Engineering, Database System, and Image Processing.

Dr. Kumar Rajnish is an Assistant Professor in the Department of Information Technology at Birla Institute of Technology, Mesra, Ranchi, Jharkahnd, India. He received his PhD in Engineering from BIT Mesra, Ranchi, Jharkhand, India in the year of 2009. He received his MCA Degree from MMM Engineering College, Gorakhpur, State of Uttar Pradesh, India. He received his B.Sc Mathematics (Honours) from Ranchi College Ranchi, India in the year 1998. He has 23 International and National Research Publications. His Research area is Object-Oriented Metrics, Object-Oriented Software Engineering, Software Quality Metrics, Programming Languages, and Database System.