International Journal of Hybrid Information Technology
Vol.7, No.2 (2014), pp.203-216
http://dx.doi.org/10.14257/ijhit.2014.7.2.19

Another New Complexity Metric for Object-Oriented Design Measurement

Kumar Rajnish

Department of Information Technology, Birla Institute of Technology, Mesra, Ranchi-835215, India

krajnish@bitmesra.ac.in

Abstract

This paper presents a new complexity metric for Object-Oriented (OO) design measurement to find at the design stage whether the classes become more complex, Moderate complex or less complex. The proposed metric is theoretically evaluated against the Weyuker's properties as well as empirically evaluated against three open source software system. Furthermore, for validating the validity of new complexity metric, the paper has also presented the comparison of proposed metric with some well known complexity metrics like Weighted Method per Class (WMC) metric of Chidamber and Kemerer (CK), Class Complexity (CC) metric of Balasubramanian, and Complexity Metric for OO Design (CMOOD) metric of Rajnish and Bhattacherjee (RB) against the same three open source software system..Automated tool were used to generate the metric values and for analyzing the results, IBM SPSS software used. The results in this paper indicates that the new complexity metric is correlated well with existing complexity metrics and may be used as predictors of complexity of class.

Keywords: *Weyuker's Properties, Object-Oriented, Complexity, Metrics, Classes, Complex, Object-Oriented Design*

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. 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 [1]. 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 [2], *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

ISSN: 1738-9968 IJHIT
Copyright ⓒ 2014 SERSC

follows that more complex the code is the higher the development effort and development time needed to develop or maintain this code.

Various OO complexity and quality metrics have been proposed and their reviews are available in the literature. RB [3] 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.* [4] presents a case study of applying design measures to assess software quality. Dapeng *et al.* [5] 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 *et al.* [6] presents the results of study in which they empirically investigated the suite of OO design metrics introduced in [7] 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.* [8] 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.* [9] 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.* [10] 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.* [11] 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.* [12] presents that metrics have been used to analyze various features of software component. 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. Sandip *et al.* [13, 14] presented in his paper to analytically evaluate against the Weyuker's property [15] 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 deals with the Weyuker's properties. Section 3 presents a definition of proposed metric, its analytical evaluation against Weyuker's properties and discussion on results from three open source software system. Section 4 deals with the comparison of new complexity metric with existing complexity metrics for analyzing the validity of new metric. Section 5 deals with conclusion and future scope respectively.

2. Weyuker's Properties

The basic nine properties proposed by Weyuker's [15] 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 [16], *i.e.*,

 Copyright ⓒ 2014 SERSC

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.

Copyright ⓒ 2014 SERSC

Assumptions. Some basic assumptions used in Section 3 have been taken from Chidamber and Kemerer [7] 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*

X_i, Y_i, Z_i are discrete random variables each characterized by some general distribution functions. Further, all the X_is are independent and identically distributed. The same is true for all the Y_is, and Z_is. This suggests that the number of methods and variables follow a statistical distribution that is not apparent to an observer of the system. Further, that observer cannot predict the variables and methods of one class based on the knowledge of the variables and methods of another class in the system

Assumption 2: In general, two classes can have a finite number of "identical" methods in the sense that a combination of the two classes into one class would result in one class's version of the identical methods becoming redundant. For example, a class "*foo_one*"has a method "*draw*" that is responsible for drawing an icon on a screen; another class "*foo_two*"also has a"*draw*" method. Now a designer decides to have a single class "*foo*" and combines the two classes. Instead of having two different "*draw*" methods the designer can decide to just have one "*draw*" method.

Assumption 3: The inheritance tree is "full", *i.e.*, there is a root, intermediate nodes and leaves. This assumption merely states that an application does not consist only of standalone classes; there is some use of sub classing.

3. Proposed Complexity Metric

3.1. Definition

A new complexity metric named "Attribute Method Complexity (AMC) "is proposed for measuring the complexity of class at the design stage. AMC may be defined as follows:

AMC=A' + M'

Where,

A' represents the attributes range values based on the sum of the actual attributes (private, public and protected) of a class and can be represented as follows:

Actual Attributes of a class	Attributes range values (A')
0 or 1 - 5	1
6 - 10	2
11 - 15	3
16 - 20	4
21 or more	5

 Copyright ⓒ 2014 SERSC

M' represents the methods range values based on the sum of the actual methods (private, public and protected) of a class and can be represented as follows:

Actual methods of a class	Methods range values (M')
0 or 1 - 5	1
6 - 10	2
11- 15	3
16- 20	4
. 21 or more	5

3.2. Intutive ideas of AMC Metric

- When classes contains more number of methods and attributes, there is increasing in AMC value, resulting in increases the complexity of class in terms of time and development effort.
- In case of inheritance, child classes accesses the attributes and methods of the parent classes, if parent classes have more AMC then there is also enhancement of AMC in the children classes, resulting in greater reuse of methods and attributes of a classes. Since inheritance is the form of reuse and it also effect on design complexity.
- Very high value of AMC *i.e.* when AMC=10 that indicate, more mental exercise and understandability is required for the designers for framing out attributes and methods of a class at the design stage.
- The mental discriminations required to design and code a class depends not only upon the numbers of methods but also upon the attributes names.
- The number of methods and number of attributes is a predictor of how much time and effort is required to develop and maintain the class.

3.3. Criteria for Analysis

Certain criteria have been defined for AMC for determining the complexity of class at design stage whether the class is more complex, moderate complex or less complex. The paper also tried to create the relationship between AMC with quality factors [chosen quality factors is "understandability", "Testability", "Risk"].

C_0: when AMC = 2, 3, and 4

Table 1. Effects on the Quality Factors for C_0 [Result: Less Complex]

Understandability	*Testability*	*Risk*
- Clear - Clarity of Code - Less mental exercise is required for framing out Methods and Attributes.	- Easy (Less Expensive) - Testing of Simple, well structured and stable methods. - More development time and effort required.	- Low

C_1: when AMC = 5, 6, 7, 0r 8

Table 2. Effects on the Quality Factors for C_1 [Result: Moderate Complex]

Understandability	*Testability*	*Risk*
- Difficult - Less clarity of code. - Less mental exercise is required for framing out Methods and Attributes.	- Less Stable (Expensive) - Testing of more complex methods. - More development effort and time is required.	- Moderate

C_2: when AMC = 9 or 10

Table 3. Effects on the Quality Factors for C_2 [Result: More Complex]

Understandability	*Testability*	*Risk*
- More Complex Behavior.	- Not Stable (More Expensive). - Untestable of Methods	- High/ Very High

3.4. Analytical Evaluation of AMC against Weyuker's Properties

This section present an analytical evaluation of AMC metric against Weyuker's axioms [15] is done. *Property 1 (Non-coarseness)* and *Property 3 (Non-uniqueness)* are satisfied because it is assumed that there is a statistical distribution of methods and attributes 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 attributes can never exceed that of the individual classes. Hence, *Property 5 (Monotonicity)* is satisfied. Consider three *classes P, Q and R*. Let the metric values for *class P* and *class Q* be the same. Also let *class R* has common methods and attributes with *class P* but not with *class Q*. Thus a combination of *class P* and *class R* will have a smaller metric value than a combination of *class Q* and *class R*. Thus, *property 6 (Non-equivalence of interaction) is satisfied*. Let *A'* and *M'* values for *class P* be *a* and *m*, for *class Q* be *a'* and *m'*, and for *class P+Q* be *a''* and *m''*.

Because of the common methods,

$a'' \leq a+a'$ and $m'' \leq m+m'$.

Hence, *Property 9 (Interaction increases complexity)* is not satisfied. *Property 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 *AMC* metric not satisfying the one property of Weyuker is that by splitting a class, there is an overall increase in the *A'* and *M'* value for all the sub classes created. In other words, complexity has increased. See in Table 4 the results of analytical evaluation results of *AMC*.

 Copyright ⓒ 2014 SERSC

Table 4. Analytical Evaluation results for AMC against Weyuker's properties

Property Number	*AMC*
1	√
2	√
3	√
4	√
5	√
6	√
7	NOT APPLICABLE
8	√
9	×
√: this indicates that metric satisfy the corresponding property. ×: this indicates that metric does not satisfy the corresponding property.	

3.5. Analysis of AMC on Open Source Software System

3.5. 1. Data Collection and Results: In order to effectively evaluate the proposed OO complexity metric and their relationship with qulaity factors, medium-size open source software systems is chosen for experiments. The following three open source sotware system were selected: http://freecode.com/projects/jadvisor/ [JADVISOR], http://sourceforge.net/projects/jmetric/ [JMETRIC], http://jcckit.sourceforge.net/ [JCCKIT].

All the programs were implemented in Java. Table 5 shows the basic information of the three systems. Please notice that the number of lines indicates all lines in Java files, including comments and blanks. File sizes are round up to integers.

The graph, summary statistics and Correlation coefficients for propose complexity metric for all three open source software systems are shown in Figure 1, Figure 2, Figure 3, Table 6, Table 7, Table 8, and Table 9.

Table 5. Details of the Subject Programs

	JADVISOR	*JCCKIT*	*JMETRIC*
#Lines	6145	27103	31397
#Classes	33	112	252
#Methods	383	1894	2242
#Attributes	224	1023	1765
Size (in KB)	191	939	943

Copyright ⓒ 2014 SERSC

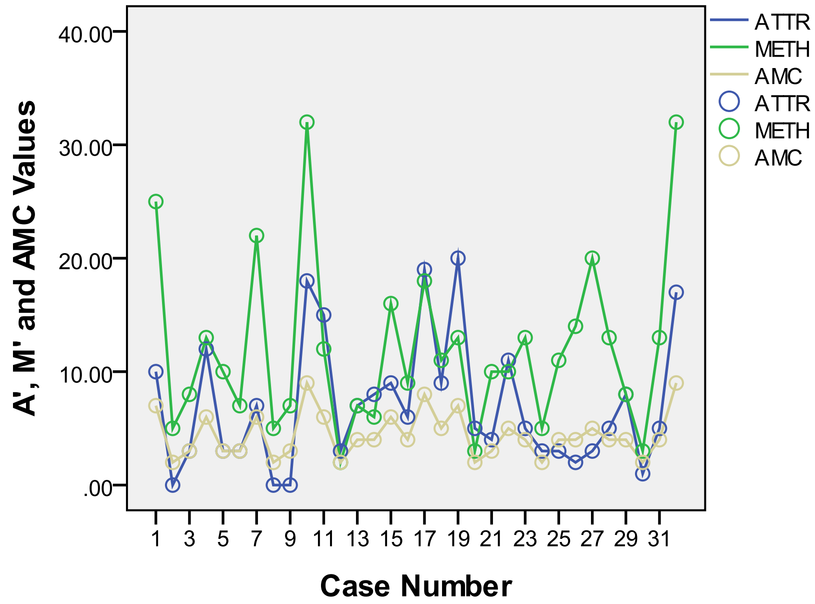

Figure 1. Parametric Values for JADVISOR System

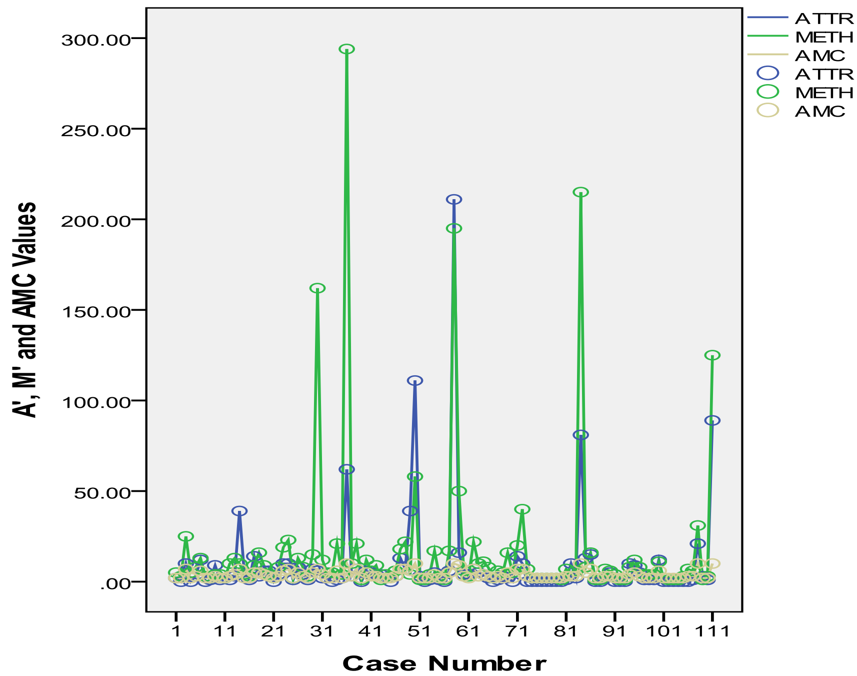

Figure 2. Parametric Values for JCCKIT System

Copyright ⓒ 2014 SERSC

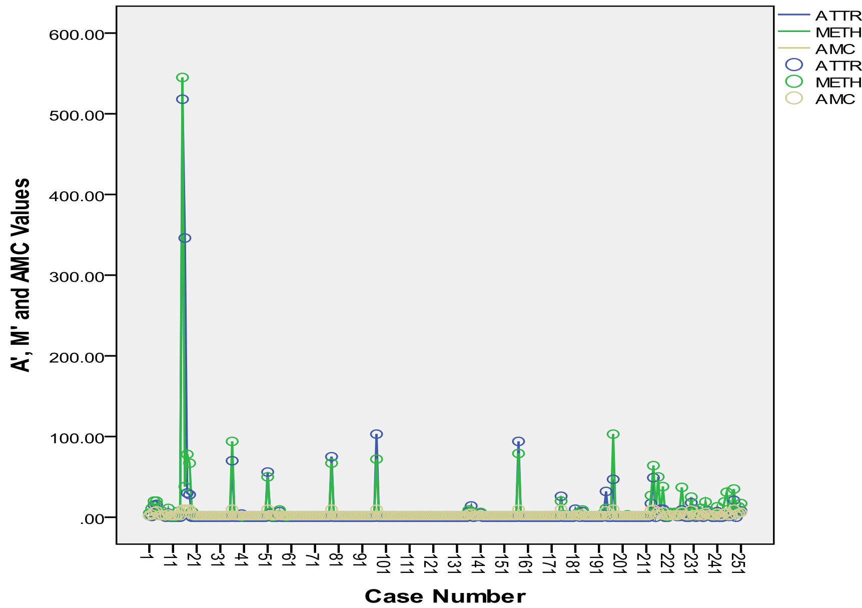

Figure 3. Parametric Values for JMETRICSystem

Table 6. Summary Statistics for the Parameters used in Proposed Complexity Metric for JADVISOR System

	Average	*Mimimum*	*Maximum*
A'	7.0000	0.00	20.00
M'	11.9688	2.00	32.00
AMC	4.4375	2.00	9.00

Table 7. Summary Statistics for the Parameters used in Proposed Complexity Metric for JCCKIT System

	Average	*Mimimum*	*Maximum*
A'	9.2162	0.00	211.00
M'	17.0631	1.00	294.00
AMC	3.7297	2.00	10.00

Table 8. Summary Statistics for the Parameters used in Proposed Complexity Metric for JMETRIC System

	Average	*Mimimum*	*Maximum*
A'	7.0000	0.00	518.00
M'	8.9323	1.00	545.00
AMC	2.8127	2.00	10.00

Table 9. Correlation Coefficient of AMC with A' and M' for all three open source software system

	A'	*M'*
JADVISOR	0.886	0.885
JCCKIT	0.658	0.674
JMETRIC	0.535	0.554

Copyright ⓒ 2014 SERSC

3.5.2. Discussion: From Figure 1, Figure 2, Figure 3, Table 6, Table 7, Table 8 and Table 9 following observations made which are as follows:

- ✓ From Figure 1, Figure 2 and Figure 3 it is observed that in *JADVISOR* system *61%* of *classes* whose *AMC* value lies between *2 or 3 or 4* as comparing to *JCCKIT* and *JMETRIC* systems *70%* found in *JCCKIT* and *88%* found in *JMETRIC* whose *AMC* value lies between *2 or 3 or 4*. The nature of the distribution of the *AMC* metric values in all the three systems is slightly different. In *JMETRIC* and *JCCKIT* systems approximately *78%* of classes contains *AMC* with value *2* (in the case of *JMETRIC*) and *45%* of classes contains *AMC* with value *2* (in case of the *JCCKIT*). So, there may be a chance of improvement by merging these classes in some other classes within the same package without affecting the abstraction and encapsulation of the classes (that increases Understandability of code). There may be question on the requirements of such classes that are with *AMC=2* in both *JMETRIC* and *JCCKIT* systems that needs to necessity of re-design for improving the software quality.

- ✓ From Table 6, Table 7 and Table 8 it is observed that the *average value* of *AMC (around 4.4375)* which is larger than the *average value* of *AMC* (*around 3.7297)* for *JCCKIT* and the *average value* of *AMC* (*around 2.8127)* for *JMETRIC* systems, still the number of classes involved in *JMETRIC* and *JCCKIT* is larger than the number of classes involved in *JADVISOR*. It may be of because the nature of distribution of methods and attributes among classes at early design stage in *JMETRIC* and *JCCKIT* systems is not proper.

- ✓ From Table 9 it is observed that the proposed complexity metric *AMC* has a very good correlation with *A'* and *M'* in *JADVISOR* system. This indicates that designer had given more effort and time for framing out methods and attributes of classes at early stages of the development of the programs and associated risk is low with *JADVISOR* because of the proper distribution of methods and attributes, whereas *AMC* has also good correlation with the same in *JCCKIT* and *JMETRIC* systems but not well like *JADVISOR* because of may be messy distribution of methods and attributes among classes.

The overall observations about all three systems is that, since more percentage of classes whose *AMC* value lies between *2 or 3 or 4* almost appears in all the three systems. The reslut mentioned above indicates that all the three systems are *less complex* in nature and needs *redesign* on the requirements of *JCCKIT* and *JMETRIC* systems.

4. Validity of AMC Metric

This section presents the discussion of *AMC* with existing complexity metrics like *CC*, *CMOOD* and *WMC* metrics [brief description is shown in Table 16] have been done for the validity or the correctness of the proposed metrics. Correlation coefficients were calculated for *AMC* with *CC, CMOOD* and *WMC* for all open source software systems and focus is on how proposed metric were correlated with existing complexity metrics.

The summary statistics and the correlation coefficients of the existing as well as proposed complexity metrics are shown in Table 10, Table 11, Table 12, Table 13, Table 14 and Table 15.

Table 10. Summary Statistics of complexity Metrics for JADVISOR system

	Minimum	*Maximum*	*Mean*	*Std. Dev.*	*Variance*
WMC	2	32	11.9688	7.55404	57.064
CC	4	93	30.2188	22.65696	513.388
CMOOD	5	130	42.22	30.099	902.370
AMC	2	9	4.44	2.015	4.060

Copyright ⓒ 2014 SERSC

Table 11. Summary Statistics of complexity Metrics for JCCKIT system

	Minimum	*Maximum*	*Mean*	*Std. Dev.*	*Variance*
WMC	1	294	17.0360	42.38094	1796.144
CC	0	1014	54.7568	152.08916	23131.113
CMOOD	0	1308	73.2252	196.90038	38769.758
AMC	2	10	3.7297	2.30393	5.3

Table 12. Summary Statistics of complexity Metrics for JMETRIC system

	Minimum	*Maximum*	*Mean*	*Std. Dev.*	*Variance*
WMC	1	545	8.96	37.135	1378.998
CC	0	5233	38.91	337.049	113601.848
CMOOD	0	5277	42.22	340.732	116098.070
AMC	2	10	2.81	2.061	4.249

Table 13. Correlation Coefficients for the different OO Complexity Metrics for JADVISOR systems (P: Pearson Correlation Coefficients and S: Spearman's RHO Correlation Coefficients)

	WMC		*CC*		*CMOOD*		*AMC*	
	P	S	P	S	P	S	P	S
WMC	1.000	1.000	0.911	0.887	0.907	0.872	0.885	0.875
CC	0.911	0.887	1.000	1.000	0.984	0.975	0.925	0.915
CMOOD	0.907	0.872	0.984	0.975	1.000	1.000	0.881	0.842
AMC	0.885	0.875	0.925	0.915	0.881	0.842	1.000	1.000

Table 14. Correlation Coefficients for the different OO Complexity Metrics for JCCKIT systems (P: Pearson Correlation Coefficients and S: Spearman's RHO Correlation Coefficients)

	WMC		*CC*		*CMOOD*		*AMC*	
	P	S	P	S	P	S	P	S
WMC	1.000	1.000	0.982	0.875	0.986	0.875	0.674	0.912
CC	0.982	0.875	1.000	1.000	0.999	0.990	0.659	0.867
CMOOD	0.986	0.875	0.999	0.990	1.000	1.000	0.666	0.854
AMC	0.674	0.912	0.659	0.867	0.666	0.854	1.000	1.000

Table 15. Correlation Coefficients for the different OO Complexity Metrics for JMETRIC systems (P: Pearson Correlation Coefficients and S: Spearman's RHO Correlation Coefficients)

	WMC		*CC*		*CMOOD*		*AMC*	
	P	S	P	S	P	S	P	S
WMC	1.000	1.000	0.941	0.793	0.942	0.724	0.554	0.872
CC	0.941	0.793	1.000	1.000	1.000	0.896	0.341	0.745
CMOOD	0.942	0.724	1.000	0.896	1.000	1.000	0.353	0.681
AMC	0.554	0.872	0.341	0.745	0.353	0.681	1.000	1.000

Copyright ⓒ 2014 SERSC

Table 16. Existing Complexity Metrics

	Description	*References*
WMC metric of CK	Consider a class C_1 with methods $M_1, M_2, M_3, ...M_n$ that are defined in the class. Let $c_1, c_2, c_3, ...c_n$ be the complexity of the methods. Then, *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.	[7]
CC metric of Balasubramanian	Class Complexity (*CC*) 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 [19] 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.	[17]
CMOOD metric of RB	$CMOOD = V_{pt} + V_{pr} + V_{pu} + PARAMS + SCM$ Where, V_{pt}: number of private instance variables in a class. V_{pr}: number of protected instance variables in a class. V_{pu}: number public instance variables in a class. *PARAMS*: sum of the formal parameters used in all local methods of a class. *SCM*: sum of the weighted static complexity of local methods in the class. To measure the static complexity RB uses McCabe's Cyclomatic Complexity [19] where the weighted result is the number of nodes subtracted from the sum of the number of edges in a program flow graph plus 2.	[18]

4.1. Discussion

From Table 10, Table 11, and Table 12, it is observed that statistics like *Variance, Std.Dev, Mean,* and *Max* is highest for *CMOOD* in all the three systems (*JADVISOR, JCCKIT,* and *JMETRIC*). *CC* has the second highest statistical values for the same. *AMC* has the lowest statistical values for the same. From *CMOOD*, *CC*, and *WMC* values it is little bit difficult to predict whether the systems is becoming *more complex*, *moderate complex* or *less complex*. But with *AMC* statistical values it can be easily predict about the complexity of classes in the systems at early stages of the development of a programs. In all the systems *mean* values for *AMC* is lowest than *CC, CMOOD* and *WMC*.

Certain interesting observations also made from Table 13, Table 14, and Table 15. From Table 13 it is observed that *AMC* correlates (both *P* and *S*) very well with *CMOOD, CC*, and *WMC*. Especially *AMC* with *CC* (*P: 0.925 S: 0.915*). Because in *JADVISOR* system values for all complexity metrics gives a proper prediction about the complexity of classes. From Table 14, it is observed that *P-correlation* is slightly well with *CC, CMOOD* and *WMC* and *S-correlation* correlates very well with *CC, CMOOD* and *WMC*. In all the columns *S-correlation* is highest (*S: 0.912*) for *1st column* than *CC, CMOOD* and *WMC*. From Table 15, it is observed that in *JMETRIC* system *P-Correlation* of *AMC* is not correlated very well with *CC, CMOOD* and *WMC* metrics. The one possible reason may be that in *JMETRIC* system, there may be a lack of requirements.

 Copyright ⓒ 2014 SERSC

5. Conclusion and Future Scope

In this paper, an attempt has been made to define new Complexity Metric (*AMC*) to measure the Complexity of class at the design stage of the software systems. On evaluating *AMC* against a set of standard criteria *AMC* 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 *AMC* 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 *AMC* along with *CC, WMC* and *CMOOD* from three open source software systems. Both systems are developed in Java. From Table 13, Table 14 and Table 15 it is found that the values of *S* and *P* obtained for each of the Complexity measures of all three open source software systems and in all cases *AMC* correlates well with all, which indicates that *AMC* is the best Complexity Measure than the alternatives.

In this study, the *AMC* is used for predicting how much effort would be required to reuse a large system, how much easy to understand and how much complex the design. Through *AMC* one can chose to measure reusability, understandability and complex design. The more classes required denotes the lower reusability, more design complex and harder to understand.

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.

Acknowledgements

The Author would like to thanks anonymous reviewers for their valuable suggestions and comments.

References

[1] W. Harrison, “Software Measurement: A Decision-Process approach”, Advances in Computers, vol. 39, (**1994**), pp. 51-105.
[2] IEEE Std 1061-1998, “Standard for software Quality Metrics Methodology”, IEEE Computer society, (**1998**).
[3] 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), Scopus (Elsevier), vol. 3, no. 1, (**2006**) June, pp. 63-70.
[4] U. L. Kulkarni, Y. R. Kalshetty and G. V. Arde, “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] 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, (**2007**), pp. 870-875.
[6] V. R. Basilli and L. W. Melo, “A Validation of Object-Oriented Design Metrics as Quality indicators”, IEEE Transaction on Software Engineering, vol. 22, no. 10, (**1996**) October, pp.751-761.
[7] S. R. Chidamber and C. F. Kemerer, ”A Metric Suite for Object-Oriented Design”, IEEE Transaction on Software Engineering, vol. 20, no. 6, (**1994**), pp. 476-493.

[8] S. Yacoub, T. Robinson and H. H. Ammar, " Dynamic Merics for Object-Oriented Design", Proceedings of 6th International Conf. on Software Metrics Symposium, (**1999**), pp. 50-61.
[9] J. Bansiya and G. C. Davis", A Hierarchical Model for Object-Oriented Design Quality Assessment", IEEE Transaction on Software Engineering, vol. 28, no. 1, (**2002**) January, pp. 4-17.
[10] J. C. Munson and T. M. Khoshgoftaar", Measuring Dynamic program Complexity", IEEE Software, vol. 9, no. 6, (**1992**) December, pp. 48-55.
[11] K. A. Mayo, A. S. Wake and M. S. Henry", Static and Dynamic Software Quality Metric tools", Department of Computer Science, Virginia Tech, Blacksburg, Technical Report, (**1990**).
[12] R. V. S. J. Sastry, V. K. 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**) May.
[13] S. Mal and K. Rajnish, "Applicability of Weyuker's Property 9 to Inheritance Metric", International Journal of Computer Application", Foundation of Computer Science, USA, vol. 66, no. 12, (**2013**) March.
[14] S. Mal and K. Rajnish, "New Quality Inheriatnce Metrics for Object-Oriented Design", International Journal of Software Engineering and its Application, SERSC publishers, Scopus, vol. 7, no. 6, (**2013**) November, pp. 185-200.
[15] E. J. Weyuker, "Evaluating Software Complexity Measures", IEEE Trans. on Software Engineering, vol. 14, (**1998**), pp. 1357-1365.
[16] B. Abreu and W. Melo, "Evaluating the Impact of OO Design on Software Quality", presented at Third International Software Metrics Symposium, Berlin, (**1996**).
[17] N. V. BalaSubramanian, "Object-Oriented Metrics", Asian Pacific Software Engineering Conference (APSEC-96), (**1996**) December, pp. 30-34.
[18] 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, (**2007**) July, pp. 36-45.
[19] T. J. McCabe, "A Complexity Measure", IEEE Transaction on Software Engineering, vol. 2, (**1976**), pp. 308-320.

Author

Kumar Rajnish

He 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 30 International and National Research Publications. His Research area is Object-Oriented Metrics, Object-Oriented Software Engineering, Software Quality Metrics, Programming Languages, and Database System.

 Copyright ⓒ 2014 SERSC