International Journal of Innovation, Management and Technology, Vol. 1, No. 5, December 2010
ISSN: 2010-0248

Empirical Analysis of CK & MOOD Metric Suit

Amandeep Kaur, Satwinder Singh, Dr. K. S. Kahlon and Dr. Parvinder S. Sandhu

***Abstract*—With the rise of the OO paradigm has come the acceptance that conventional software metrics are not adequate to measure object-oriented systems. This has inspired a number of software practitioners and academics to develop new metrics that are suited to the OO paradigm. The MOOD metrics have been subjected to much empirical evaluation, with claims made regarding the usefulness of the metrics to assess external attributes such as quality and maintainability. We evaluate the MOOD metrics on a theoretical level and show that any empirical validation is premature due to the majority of the MOOD metrics being fundamentally flawed. The metrics either fail to meet the MOOD team's own criteria or are founded on an imprecise, and in certain cases inaccurate, view of the OO paradigm. One of the suite of OO design measure was proposed by Chidamber and Kemerer. The author of this suite of metrics claim that these measure can aid users in understanding object oriented design complexity and in predicting external software qualities such as software defects, testing, and maintenance effort. Use of the CK set of metrics and other complexity measures are gradually growing in industry acceptance. This is reflected in the increasing number of industrial software tools, such as Rational Rose, that enable automated computation of these metrics. Even though this metric suite is widely, empirical validations of these metrics in real world software development setting are limited. Various flaws and inconsistencies have been observed in the suite of six class based metrics. We validate some solutions to some of these anomalies and clarify some important aspects of OO design, using Six projects in particular those aspects that may cause difficulties when attempting to define accurate and meaningful metrics. These suggestions are not limited to the MOOD and CK metrics but are intended to have a wider applicability in the field of OO metrics.**

***Index Terms*—CK Metric, MOOD Metric Suit, Cohesion, Coupling, Object Oriented.**

I. Introduction

A metric is a standard unit of measure, such as meter or mile for length, or gram or ton for weight, or more generally, part of a system of parameters, or systems of measurement, or a set of ways of quantitatively and periodically measuring, assessing, controlling or selecting a person, process. A software metric (noun) is the measurement of a particular characteristic of a program's performance or efficiency. A rule for quantifying some characteristic or attribute of a computer software entity. Metrics can be used for software entities such as requirements documents, design object models, or database structure models. Metrics for programs can be used to support decisions about testing and maintenance and as a basis for comparing different versions of programs. Ideally, metrics for the development cost of software and for the quality of the resultant program are desirable.

For Software measurements, it is numerical ratings to measure the complexity and reliability of source code, the length and quality of the development process and the performance of the application when completed.

II. Need to Develop the Metrics

As today‘s software applications are more complex and software failure is more critical, potentially resulting in economic damage or even threatening the health or lives of human beings, a means of effectively measuring the quality of software products is needed. Effective management of the software development process requires effective measurement of that process.

III. Related Work

Chidamber and Kemerer, in 1994, developed a set of six metrics to identify certain design and code traits in OO software.

MOOD Metrics by Abreu et al introduces that these attributes can express the quality of internal structure, thus being strongly correlated with quality characteristics like analyzability, changeability, stability and testability, which are important to software developers and maintainers.

Rosenberg, et al evaluated the Quality Assurance of Object Oriented Assurance and Risk Assessment of Object Oriented Metrics.

Briand, et al gives framework for cohesion and coupling measurement in Object Oriented System.

Linda, et al concluded that, as the fundamental building block of metric is object not algorithm, the approach to S/W metrics for Object Oriented Program.

Aggarwal et al proposed a set of metrics that are related to various constructs like class, coupling, cohesion, information hiding, polymorphism, reusability.

IV. Analysis and Results

The research was done by surveying the literature on object oriented metrics which are used for measuring design and code quality of software code. Many object-oriented metrics have been used specifically for the purpose of assessing the design of a software system. MOOD and CK set of metrics cover every aspect Object Oriented Paradigm.

But many of flaws have been observed in MOOD and CK set of metrics which are illustrated with the help of examples and live projects applied on them.

A. Flaws in CK Metrics Definitions

1) Weighted Methods per Class (WMC)

WMC is a count of sum of complexities of all methods in a class.

$$WMC = \sum_{i=0}^{n} C_i \tag{4.1}$$

Where n = No. of methods in one class

C_i = Complexity of every class

WMC break an elementary rule of measurement theory that a measure should be concerned with a single attribute . This is also not clear whether the inherited method is to be counted in base class (which defines it), in derived classes or in both.

2) Response For a Class (RFC)

It is number of methods in the set of all methods that can be invoked in response to a message sent to an object of a class. It includes all methods accessible within the class hierarchy. It looks at the combination of the complexity of a class through the number of methods and the amount of communication with other classes.

$$RS = \{M\}\ all\ i\{R_i\} \tag{4.2}$$

where{ R_i} = set of methods called by method i and

{ M } = set of all methods in the class.

The response set of a class is a set of methods that can potentially be executed in response to a message received by an object of that class. But here the point to be noted is that because of practical considerations, Chidamber and Kermerer recommended only one level of nesting during the collection of data for calculating RFC. This gives incomplete and ambiguous approach as in real programming practice there exists "Deeply nested call-backs" that are not considered here.

3) Depth of Inheritance Tree (DIT)

It is defined as the maximum length from the node to the root of the tree and measured by the number of ancestral classes. But the definition should measures the maximum ancestor classes from the class-node to the root of the inheritance tree.

NOC Metric Number of children (NOC): of a class is the number of immediate sub-classes subordinated to a class in the class hierarchy. The definition of NOC metric gives the distorted view of the system as it counts only the immediate sub-classes instead of all the descendants of the class as illustrated by the figure 1.

Where Both A and B classes have *NOC* value of two, but there are nine classes that inherits the properties of class A and a total of seven classes inherit class B's properties. So the *NOC* value of a class should reflect all the subclasses that share the properties of that class.

$$NOC(class) = Number\_of\_immediate\_subclasses + \sum_i NOC(i) \tag{4.3}$$

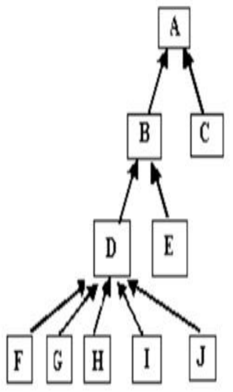

Figure 1. Example showing the distorted view of NOC metric

4) Coupling Between Object Classes (CBO)

According to this metric "Coupling Between Object Classes" (CBO) for a class is a count of the number of other classes to which it is coupled. Theoretical basis of CBO relates to the notion that an object is coupled to another object if one of them acts on the other, i .e. methods of one use methods or instance variables of another.

As Coupling between Object classes increases, reusability decreases and it becomes harder to modify and test the software system. But for most authors coupling is reuse, which raises ambiguity. So there is the need to find out the coupling level that implies the goodness of design

5) Lack of Cohesion in Methods (LCOM)

Consider a Class C1 with n methods M1 , M2 ..., Mn . Let {Ij } = set of instance variables used by method Mi .There are n such sets {I1},{I2}... {In}. Let P = { (Ii ,Ij) | Ii ∩ Ij = ∅ } and Q = { (Ii ,Ij) | Ii ∩ Ij ≠ Ø }. If all n sets {I1},{I2}... {In} are Ø then let P = Ø [4]. Lack of Cohesion in Methods (LCOM) of a class can be defined as:

$$LCOM = |P| - |Q|, if |P| \rangle |Q| \tag{4.4}$$
$$LCOM = 0\ OTHERWISE$$

The high value of LCOM indicates that the methods in the class are not really related to each other and vice versa. According to above definition of LCOM the high value of LCOM implies low similarity and low cohesion, but a value of LCOM = 0 doesn't implies the reverse .

Consider the example in figure 11 (a) the value of LCOM is 8 (as | P | =9 and | Q | = 1). Whereas in figure 2 (b) the value of LCOM is also 8 (as | P | =18 and | Q | = 10), but figure 2 (a) example is more cohesive than figure 2 (b) example. So the above said definition of CK metric for LCOM is not able to distinguish the more cohesive class from the less ones. This is simple violation of the basic axiom of measurement theory, which tells that a measure should be able to distinguish two dissimilar entities. So this deficiency offends the purpose of metric.

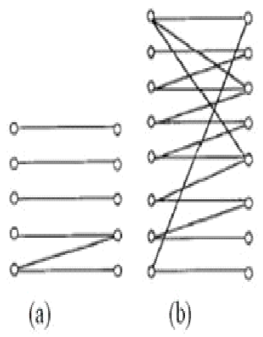

Figure 2. Examples of (a) less cohesive (b) densely cohesive class

B. Flaws in MOOD Metrics Definitions

1) Metric1: Method Inheritance Factor

Definition of the MIF is inconsistent with the 0-1 scale. Consider the following system with the hierarchical structure: A->B->C.

Class A {public void x(); public void y();}

Class B extends A { //no methods defined}

Class C extends B {//no methods defined}

B and C both inherit the two methods defined in class A and define no further methods. This is the maximum possible method inheritance in this sytem (i.e. all methods that can be inherited have been inherited, by all classes that are ble to inherit them). Intuitively, it seems that the MIF value fro this system should be 100%, but in fact it is 66.6%

$$Mi(A)=0, Mi(B)=2, Mi(C)=2, Total=4 \qquad (4.5)$$
$$Ma(A)=2, Ma(B)=2, Ma(C)=2, Total=6$$

This can be further illustrated. If class C in the above example had a new method added it should not change the MIF value fro the system. This is consistent with our intuitive understanding of the method inheritance. If fact we find that it does, with Ma(C) 3 MIF becomes 4/7 0.57 (57%).

2) Metric2: Attribute Inheritance factor

The metric Ai (Ci) is meaningless in the sense that the concept of the inheritance concerns the behavior defined in a method, an attribute does not have behavior, and thus cannot be overridden or inherited. It is certainly possible for a class B to define an attribute named x even if its parent class A already has an attribute (of the same type) named x but the attribute B.x does not override A.x. The methods of A that refer to x will use A.x and the methods of B that refer to x will use B.x. If B.x is counted as an 'overriding' attribute, rather than a new attribute when calculating Ad(B) then A.x would not be counted as an inherited attribute when calculating Ai(B). This would further result in an inaccurate value being returned for Aa(B).

Inherited Factors Solution: The definitions of the MIF and AIF need to be amended to remove this inconsistency. However, even if this can be done the main problem with the MIF and AIF metrics is that they not really telling us anything of the interest, especially if it is accepted that all private methods and attributes are inherited. It may be more interesting to develop a metric that measures inheritance at a class level, rather than separate metrics to capture method and attribute inheritance. After all, it is classes that are extended; methods and attributes just come as part of the package.

A class Inheritance factor (CIF) metric could be defined as the total count of all ancestors for all classes divided by the maximum possible inheritance for the system. Inheritance is one-way. If class A extends class B then it is possible for class B to also extend class A. This means that the maximum inheritance level for a system with n classes will be 0+1…(n-1). Therefore, a more formal definition for a class inheritance factor metric would be:

$$CIF = \frac{\sum_{i=1}^{TC} AC(C_i)}{TC*(TC-1)/2} \qquad (4.6)$$

Where *TC*-Total Classes. This value will be 100% when the classes are all arranged in a linear hierarchy. It will be 0% when there is no inheritance.

3) Metric3: Method Hiding Factor

It is recommended that MHF should not be lower than a particular (as yet undefined) value but suggest that there is no upper limit, thus implying that it is 'good' for all methods in a class to be hidden (private). However, the number of private methods in a class doesn't tell us anything about the degree of information hiding in a class. It may tell us that a particular method (or methods) has been broken down into a number of smaller methods to avoid duplication or for clarity of understanding. Such methods would only need to be visible to the containing class. But whether or not a method is broken down this way the containing class's implementation is still hidden. In the following example both classes have equal 'information hiding' levels:

In class A all of method m0's behavior is contained in the body of m0. In class B the behavior has been separated into three smaller methods which are called by m0. Both classes have identical interfaces and their respective implementations are equally well hidden from client classes. A count of the number of private methods in a class is not a particularity useful metric, and certainly does not contribute anything to our knowledge of a class's encapsulation level.

4) Metric4: Attribute Hiding Factor

This is a clearly defined metric with no apparent inconsistencies. Its use is in determining the level of visibility of a class's data.

5) Metric5: Polymorphism factor

It is possible, indeed highly likely, that a sub-system will consist of a set of classes that extends a framework. This may be a set of library classes or a framework of low(er) level system classes. When measuring the sub-system it should be only the Classes that belong to the sub-system that are measured; classes outside of its boundaries (which is where the framework or library classes will lie) should not be considered. In such cases the denominator for the POF measure may be less than the numerator, resulting in a value greater than 1. An example will make this clear. Sub-system "S" produces a value for POF which is outside the range 0-1 a shown in figure below

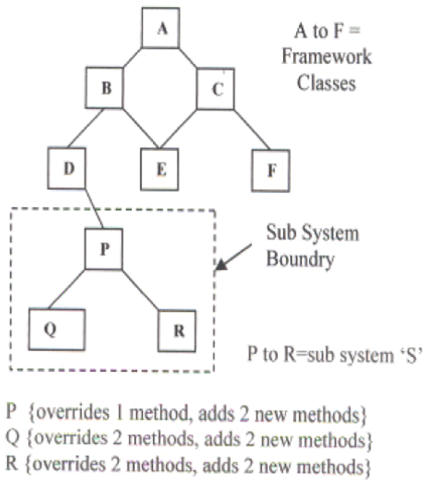
A to F = Framework Classes
A
B
C
D
E
F
Sub System Boundry
P
Q
R
P to R=sub system 'S'
P {overrides 1 method, adds 2 new methods}
Q {overrides 2 methods, adds 2 new methods}
R {overrides 2 methods, adds 2 new methods}

Figure 3: Example for POF metric

$$Mo(P) = 1, Mo = (Q)2, Mo(R) = 2 \qquad (4.6)$$
$$(Mn(P) = 2 * DC(P) = 2) = 4$$
$$(Mn(Q) = 2 * DC(Q) = 0) = 0$$
$$(Mn(R) = 2 * DC(P) = 0) = 0$$
$$Therefore, POF\ for\ subsystem\ "S" =$$
$$(1+2+2)/(4+0+0) = 5/4 (and\ 5/4 > 1)$$

Taking this concept one step further, it is possible for an entire system to be built using an existing framework. This is especially likely in languages that are shipped with large class libraries, such as Java or Smalltalk. In such cases the whole system could produce POF>1.

The definition of POF can only be applied to complete hierarchies. Therefore, a formula needs to be proposed which should be applied to sub-systems. The formula for the same was proposed by Mayer et al.

The new formula for the POF metric is:

$$\frac{\sum_{i=1}^{TC} MOV(C_i)}{\sum_{i=1}^{TC} MOV(C_I)} \qquad (4.7)$$

The numerator is unchanged. The denominator is the sum of all Mov(C_i) where Mov(C_i) is the count of all methods that can potentially be overridden by class C_i.

This will consist of all the methods in the parent class or classes excluding private methods and class-wide (static) methods. The new definition remove the anomaly described above and means that it is now down scalable to sub-systems. This definition still contains the discontinuity concerning systems with no inheritance identified by Harrison.

6) *Metric6: Coupling Factor*

This metric is intended to count all client-supplier relationships in a system. The important point here is that the relationship between any two classes in a system is not constrained to just one or the other of these relationship types. As an example consider the two classes, Component and container, from the Java java.awt library package. Component is the super class of all graphical components and container is one of its subclasses. Thus the two classes are in an inheritance relationship: Container is-a Component. However, each class also contains an attribute of the other class type, i.e. Component has an attribute of type Container and Container has an (array) attribute of type Component. Container's use of a set of Components has nothing to do with the fact that Component is its super class, indeed if the hierarchy was redesigned to alter this fact it would not alter a container's need to maintain references to all the components that reside in it.,

The question that the MOOD team does not adequately answer is whether a client supplier relationship under these conditions is counted. There is probably no 'correct' way of dealing with this situation is terms of the COF metric but a decision needs to be made one way or the other and it needs to be explicit in the metric's definition.

C. *Results*

The metrics chosen for analysis can be divided into 7 categories viz. size, coupling, cohesion, inheritance, information hiding, polymorphism and reuse metrics. Figure 4 to Figure 11 shows the Bar Chart of Metrics. Table 1 and Table 2 shows the Comparison of values of Projects before and after removing Inconsistencies from CK and MOOD set of metrics.

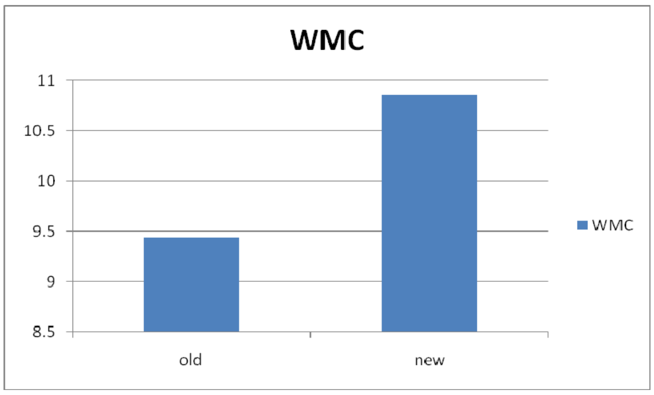

Figure 4. Bar Chart for WMC values before and after removing inconsistencies

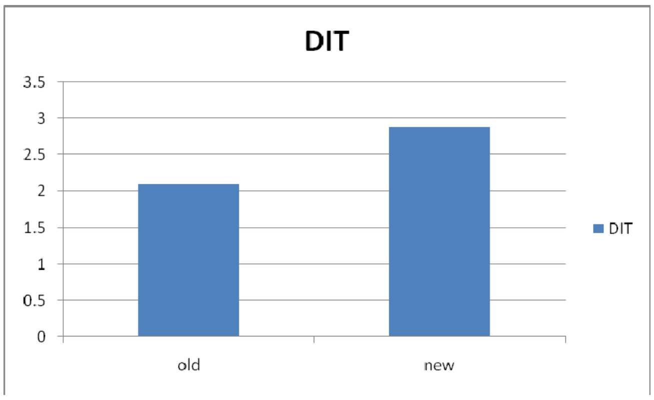

Figure 5. Bar Chart for DIT values before and after removing inconsistencies

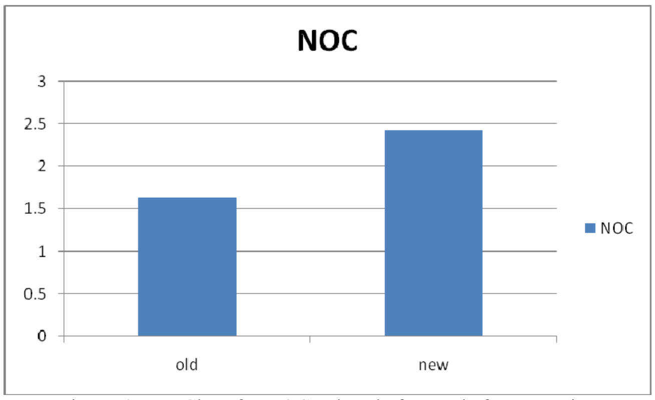

Figure 6. Bar Chart for NOC values before and after removing inconsistencies

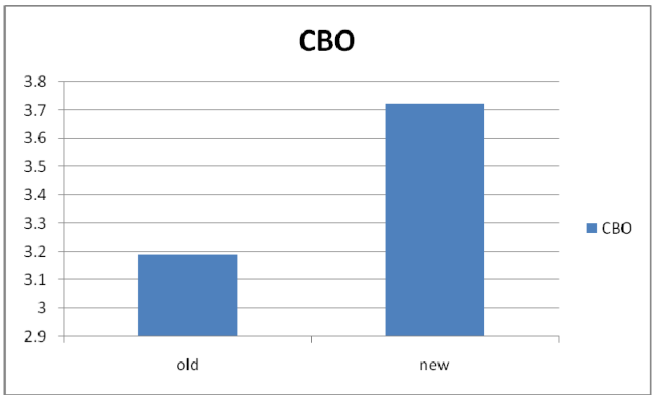

Figure 7. Bar Chart for CBO values before and after removing inconsistencies

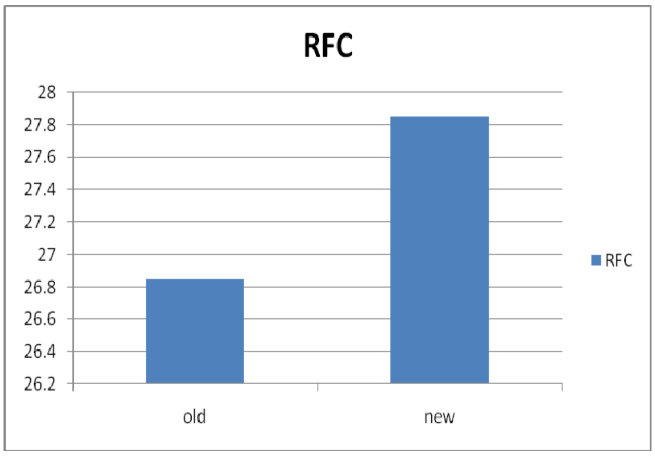

Figure 8. Bar Chart for RFC values before and after removing inconsistencies

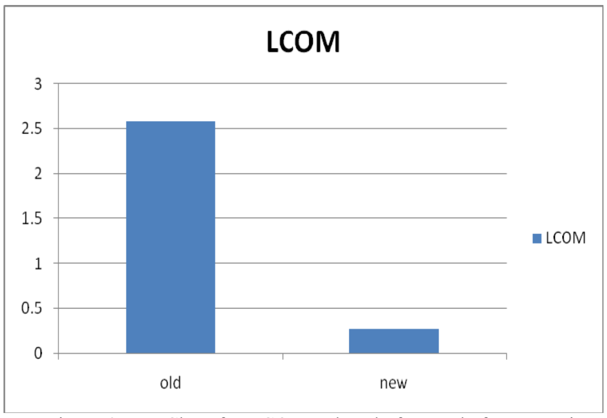

Figure 9. Bar Chart for LCOM values before and after removing inconsistencies

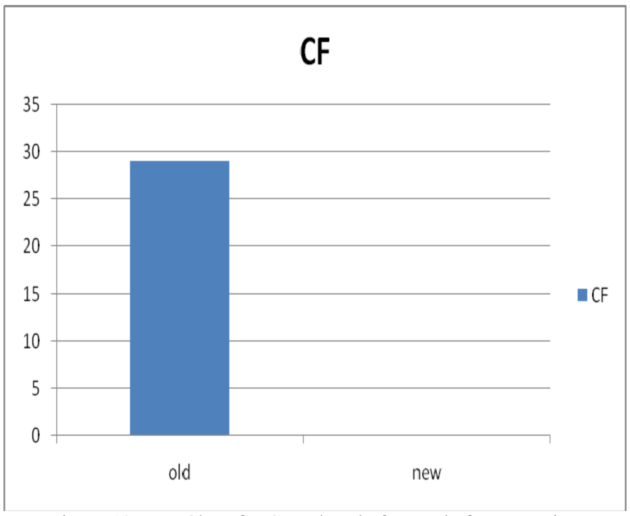

Figure 10. Bar Chart for CF values before and after removing inconsistencies

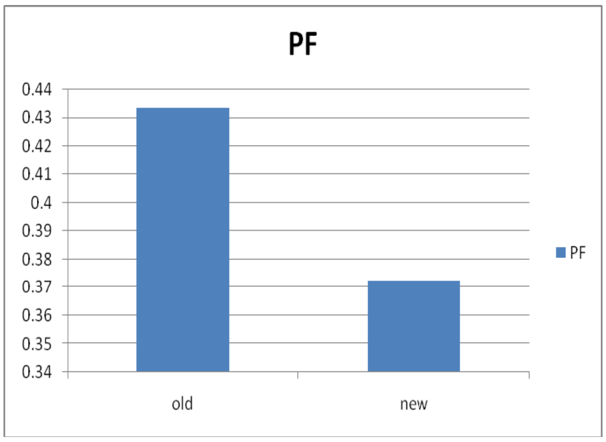

Figure 11. Bar Chart for PF values before and after removing inconsistencies

V. Discussion

Following are the observations made from applying the metrics on projects

1) RFC measures the complexity of the software by counting the no. of methods in the class and also capture the information about the coupling of the class to other classes
2) CBO value is generally less in sample data ,hence classes are easy to understand ,reuse and maintain
3) LCOM values are zero because the no. of pairs of methods having access to common attributes is more than the no. of pairs of method having no attributes. It implies that classes are cohesive
4) The DIT and NOC values are medium in all projects. This shows that inheritance is used in all the classes in the optimum level
5) The MIF value is null for the project 1 &4.it is observed that there are very less methods in a super class. They contain only abstract methods which are overridden in subclass.
6) MIF and AIF measures can provide overall system view about amount of information hiding incorporated by software designers
7) MHF has nil values indicating that methods are declared public by developers
8) PF value is null for all projects. This shows that more

overloading is used in project1 as compared to other projects.

REFERENCES

[1] F.B. Abreu and R. Carapuca, ”Candidate Metrics for Object- Oriented Software within a Taxonomy Framework,”]. System and Software, vol. 26, no. l, pp. 87-96, Jan. 1994.

[2] L. Briand, S. Morasca, and V. Basili, De$ning and Vdidating High-Level Design Metrics, Techtucal Report CS-TR-3301, Univ. of Maryland, Dept. of Computer Science, College Park, Md., 1994.

[3] L. Briand, S. Morasca, and V. Basili, ”Property Based Software Engineering Measurement,” IEEE Trans. Software Eng., vol. 22, no. 1, p. 68-86, Jan. 1996.

[4] I.Brooks, “Object-Oriented Metrics Collection and Evaluation with a Software Process,” Proc. OOPSLA ’93 Workshop Processes and Metrics for Object-Oriented Software Development, Washington, D.C., 1993.

[5] S.R. Chidamber and C.F. Kemerer, “A Metrics Suite for Object-Oriented Design,” IEEE Trans. Software Eng., vol. 20, no. 6, pp. 476493, June 1994.

[6] S.R. Chidamber and C.F. Kemerer, ”Authors Reply,” lEEE Trans. Software Eng., vol. 21, no. 3, p. 265, Mar. 1995.

[7] L.Briand , W.Daly and J. Wust, Unified Framework for Cohesion Measurement in Object-Oriented Systems. Empirical Software Engineering, 3 65-117, 1998.

[8] L.Briand , W.Daly and J. Wust, A Unified Framework for Coupling Measurement in Object-Oriented Systems. IEEE Transactions on software Engineering, 25, 91 121,1999.

[9] L.Briand , W.Daly and J. Wust, Exploring the relationships between design measures and software quality. Journal of Systems and Software, 5 245-273, 2000.

[10] S.R.Chidamber and C.F.Kamerer, A metrics Suite for Object-Oriented Design. IEEE Trans. Software Engineering, vol. SE-20, no.6, 476-493, 1994.

[11] N.Fenton et al, Software Metrics: A Rigorous and practical approach. International Thomson Computer Press, 1996.

[12] R.Harrison, S.J.Counsell, and R.V.Nithi, An Evaluation of MOOD set of ObjectOriented Software Metrics. IEEE Trans. Software Engineering, vol.SE-24, no.6, pp. 491-496 June1998.

[13] B.Henderson-sellers, Object-Oriented Metrics, Measures of Complexity.Prentice Hall, 1996.

[14] Lorenz, Mark & Kidd Jeff, Object-Oriented Software Metrics, Prentice Hall, 1994.

[15] McCabe and Associates, Using McCabe QA 7.0, 1999, 9861 Broken Land Parkway 4th Floor Columbia, MD 21046.

[16] McCabe, T. J., “A Complexity Measure”, IEEE Transactions on Software Engineering, SE-2(4), pages 308-320, December 1976.

[17] Moreau, D. R., “A Programming Environment Evaluation Methodology for Object-Oriented Systems”, Ph.D. Dissertation, University of Southwestern Louisiana, 1987.

[18] Moreau, D. R., and Dominick, W. D., “Object-Oriented Graphical Information Systems: Research Plan and Evaluation”, Journal of Systems and Software, vol. 10, pp. 23-28, 1989.

[19] Moreau, D. R., and Dominick, W. D., “A Programming Environment Evaluation Methodology for Object-Oriented Systems: Part I – The Methodology”, Journal of Object-Oriented Programming, vol. 3, pp. 38-52, 1990.

[20] Rosenberg, L., “Metrics for Object-Oriented Environment”, EFAITP/AIE Third Annual Software Metrics Conference, 1997.

TABLE I. BEFORE REMOVING INCONSISTENCIES

	Source code 1	**Source code 2**	**Source code 3**	**Source code 4**	**Source code 5**	**Source code 6**	**Mean**	**Median**	**Std dev.**
WMC	20	31	35	10	17	28	**23.5**	**24**	**9.43928**
RFC	79	12	16	8	15	18	**24.66667**	**15.5**	**26.84524**
CBO	2	5	1	4	8	9	**4.833333**	**4.5**	**3.188521**
LCOM	2	1	1	4	6	7	**3.5**	**3**	**2.588436**
DIT	2	2	4	3	7	6	**4**	**3.5**	**2.097618**
NOC	2	3	2	4	6	5	**3.666667**	**3.5**	**1.632993**
CF	78	25	29	2	5	3	**23.66667**	**15**	**29.07691**
MIF	0.491	1.5	2.5	0	0.4	0.13	**0.836833**	**0.4455**	**0.97121**
AIF	0.676	1	1.5	0.3	0.5	0.4	**0.729333**	**0.588**	**0.450647**
MHF	0.305	0.897	0.834	0	0	0	**0.339333**	**0.1525**	**0.424808**
AHF	0.375	0.667	0.444	0.16	0.94	0.86	**0.574333**	**0.5555**	**0.300765**
PF	0	0.8	1	0	0.8	0.4	**0.5**	**0.6**	**0.43359**

TABLE II. AFTER REMOVING INCONSISTENCIES

Metric	**Source code 1**	**Source code 2**	**Source code 3**	**Source code 4**	**Source code 5**	**Source code 6**	**Mean**	**Median**	**Std dev.**
WMC	20	39	40	14	20	30	**27.16667**	**19.33333**	**10.85204**
RFC	85	14	19	15	17	20	**28.33333**	**0.641167**	**27.85438**
CBO	4	6	4	8	13	11	**7.666667**	**0.519323**	**3.723797**
LCOM	0.42	0.1979	0.5777	0.123	0.578	0.889	**0.464267**	**0.245917**	**0.281274**
DIT	4	6	11	3	7	8	**6.5**	**0.564917**	**2.880972**
NOC	6	7	11	4	6	5	**6.5**	**0.466795**	**2.428992**
CF	0.0166	0.333	0.0394	0	0	0	**0.064**	**0.132285**	**0.132285**
MIF	MIF and AIF is replaced with CIF								
AIF									
MHF	These are clearly defined metrics with no apparent inconsistency								
AHF									
PF	0	0.4	1	0	0.4	0.2	**0.333**	**0.37238**	**0.37238**