Some Observations based on Comparison of MOOD and CK Software Metrics suites for Object Oriented System

[1]Ganesh Chandra, [2]D. L. Gupta, [3]A. K. Malviya
[1,2,3]Dept. of CSE, KNIT, Sultanpur, India

Abstract
Software metrics measure both quantitative and qualitative aspects of a system. In this paper we have calculated and compared the MOOD and C K metrics suits for object oriented system on a given data set. We have found that encapsulation property of object oriented system is high in MOOD metric suite in comparison to CK metric suites.

Keywords
MOOD Metrics, CK Metrics, Cohesion and Coupling

I. Introduction

In recent years, there has been a growing interest in the study of software metrics to increase the quality of software. Many experiments have been made to understand how metrics can be used to characterize to improve the quality of software [1-2]. Software measurement will need to play an increasingly important role in software engineering. Over the past twenty years, a significant number of software metrics have been proposed to better control and understand software at closely from a measurement method perspective to analyze the quality of software.

The software engineers think that the isolating objects makes their software easier to manage but many of them have the reverse views that software becomes more complex to maintain and document. Because of this, engineers move towards the Object-Oriented Paradigm (OOP) as it could increase the capability of programming through its reusability function. By the implementation of OOP the researchers modified and validated the conventional metrics theoretically or empirically. Sizing and complexity metrics were the most impressive contributions for effort and cost estimation in project planning.

Object-Oriented (OO) techniques have been widely popular in software development since the early 1990s. To ensure the quality of OO software, researchers have proposed many metrics such as Chidamber & Kemerer metrics (CK) [8] and metrics for object-oriented design (MOOD) set [5]. There are many ways of applying these metrics to real projects, and successful experiences from software development have proved the validity of these metrics.

This paper is organised as follows, section II, consists of MOOD metrics suites, in section III, we have described about CK metrics suites followed by conclusion in section IV.

II. Mood Metrics Set

MOOD set of metrics are proposed by F.B Abreu describes use of objects oriented paradigm in software code. These metrics help to assess quality and productivity of an object oriented system. MOOD refers to basic structural mechanism of the object-oriented paradigm as encapsulation (MHF, AHF) [3], inheritance (MIF, AIF) [4], polymorphism (POF), and message passing (COF). In MOOD metrics model two main features are used in every metrics: methods and attributes. To perform several kinds of operations on objects such as obtaining of modifying the status of objects, methods are used. To represent the status of each object in the system, attributes are used. Each feature (method and attributes) is either visible or hidden from a given class .The MOOD (Metrics for Object Oriented Design) set includes the following metrics:

- Method Hiding Factor (MHF)
- Attribute Hiding Factor (AHF)
- Method Inheritance Factor (MIF)
- Attribute Inheritance Factor (AIF)
- Polymorphism Factor (PF)
- Coupling Factor (CF)

A. Method Hiding Factor

The MHF metrics states the sum of the invisibilities of all methods in all classes [3]. The invisibility of a method means how much the method is hidden from the percentage of the total class. Abreu et.al [7], states, the MHF denominator is the total number of methods defined in the system under consideration. If the value of MHF is high (100 %) it means all the methods are private which indicates very little functionality. Thus it is not possible to reuse methods with high MHF. MHF with low (0%) value indicate all the methods are public that means most of the methods are unprotected. The MHF metrics is defined as follows:

$$MHF = \frac{\sum_{i=1}^{TC} M_h(C_i)}{\sum_{i=1}^{TC} M_d(C_i)}$$

$$M_d(C_i) = M_V(C_i) + M_h(C_i)$$

$M_d(C_i)$ = methods defined in a class.

$M_V(C_i)$ = visible methods in a class.

$M_h(C_i)$ = hidden methods in a class.

TC = total classes.

B. Attribute Hiding Factor

The AHF metrics shows the sum of the invisibilities of all the attributes in all classes [3]. The invisibility of an attribute represents the percentage of the total classes from which the attributes are hidden. MHF and AHF represent the average amount of hiding among all the classes in the system [7]. If the value of AHF is high (100 %), it means all attributes are private. AHF with low (0%) value indicates all attributes are public. The AHF metric is defined as follows:

$$AHF = \frac{\sum_{i=1}^{TC} A_h(C_i)}{\sum_{i=1}^{TC} A_d(C_i)}$$

$$A_d(C_i) = A_V(C_i) + A_h(C_i)$$

$A_d(C_i)$ = attributes defined in a class.

$A_V(C_i)$ = visible attributes in a class.

$A_h(C_i)$ = hidden methods in a class.

C. Method Inheritance Factor

The MIF metrics states the ratio of the sum of inherited methods in all classes of the system under consideration to the total number of available methods for all classes of the system [4-5]. If the value of MIF is low (0%), it means that there is no methods exists in the class as well as the class lacking an inheritance statement. MIF is defined as follows:

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$$

$$M_a(C_i) = M_d(C_i) + M_i(C_i)$$

$$M_d(C_i) = M_n(C_i) + M_o(C_i)$$

$M_a(C_i)$ = available methods in class.
$M_d(C_i)$ = methods defined in class.

$M_n(C_i)$ = new methods in class.
$M_o(C_i)$ = overriding methods in class.
$M_i(C_i)$ = methods inherited in class.

D. Attribute Inheritance Factor

The ratio of the sum of inherited attributes in the sum to the available attributes in all classes of the system is called attribute inheritance factor. AIF denominator is defined in an analogous manner and provides an indication of the impact of inheritance in the object oriented software [4]. If the value of AIF is low (0%), it means that there is no attributes exists in the class as well as the class lacking an inheritance statement. AIF is defined as follows:

$$AIF = \frac{\sum_{i=}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}$$

$$A_a(C_i) = A_d(C_i) + A_i(C_i)$$

$$A_d(C_i) = A_n(C_i) + A_o(C_i)$$

$A_a(C_i)$ = attribute available in class.

$A_d(C_i)$ = attribute defined in class.

$A_n(C_i)$ = new attributes in class.

$A_o(C_i)$ = overriding attributes in class.

$A_i(C_i)$ = attributes inherited in class.

E. Polymorphism Factor

The numerator and denominator of POF is used to represents the actual number of possible different polymorphic situation and the maximum number of possible distinct polymorphic situation for class C_i [3, 5]. The POF is defined as follows:

$$PF = \frac{\sum_{i=}^{TC} M_o(C_i)}{\sum_{i=1}^{TC} [M_n(C_i) \times DC(C_i)]}$$

where
$M_n(C_i)$ = new methods in class C_i.

$M_0(C_1)$ = overriding methods in class.

$DC(C_i)$ = number of descendants of class C_i

The value of POF can be varies between 0% and 100%. If a project have 0% POF, it indicates the project uses no polymorphism and 100% POF indicates that all methods are overridden in all derived classes.

F. Coupling Factor

The COF is defined as the ratio of the maximum possible number of couplings in the system to the actual number of coupling which is not imputable to inheritance [6]. The value of COF can be varies between 0% and 100%. The 0% COF indicates no classes are coupled and 100% COF indicates all classes are coupled with all other classes. High values of COF should be avoided. The COF is defined as follows.

$$COF = \frac{\sum_{c=1}^{TC}\left[\sum_{s=1}^{TC} is\_client(C_c, C_s)\right]}{TC^2 - TC - 2 \times \sum_{i=1}^{TC} DC(C_i)}$$

where is_client (C_c , C_s) = { 1 [iff] C_c =>C_c $\Lambda C_s \neq C_s$} or {0 otherwise}
where C_c =>C_s, represents the relationship between client class Cc and supplier class C_s.

Table 1: MOOD Metrics Result

Metrics	MHF	AHF	MIF	AIF	PF	CF
Values	17/39= 0.435	9/9=1	24/63= 0.38	6/15= 0.4	5/31= 0.161	0

In Table 1, we have calculated MOOD metrics suite applied on data set (Appendix-1 [14]) which has four classes named Basic Component, Application, UI Component and Clock. We observed that Clock class and Application class inherits the properties of UI Component class. From table1the value of MHF is 0.435 and AHF is 1, which shows that the information of attributes and methods in a system are hidden and secure.
The values of MIF & AIF are 0.38 & 0.4 respectively, which shows that in a data set classes utilizing the properties of their parents classes to reduce the cost of the system and also to increase the quality. Here it is also observed that coupling between classes is

0, so the reliability of software process will increases. Therefore quality will also increases because of less complexity.

III. C & K Metrics

One of the most widely referenced sets of object oriented software metrics has been proposed by Chidamber and Kemerer in 1991 at Object Oriented Programming Systems Languages and Applications conference (OOPSLA). The metric suite provided by C & K have been used in this study [8] are as follows:

A. Weighted Methods Per Class (WMC)

WMC is a measure of the number of methods implemented within class [8-9, 12]. This metric measures understandability, maintainability, and reusability as follows:

The number of methods in class reflects the time and efforts required to develop and maintain the class.

The higher the number of methods indicates the greater potential impact on children, since children inherit all of the methods defined in a class.

Consider a class C_i with number of methods $M_1, , M_2$,………………… M_n . Let C_1 , C_2 …………. C_n be the static complexities of the methods. Then:

$$WMC = \sum_{i}^{n} C_i$$

where n is the number of methods in the class. The value of WMC = n if all static complexities are considered to be unity [8].

B. Depth of Inheritance Tree (DIT)

DIT is equal to the maximum length from the class node to the root of the tree. It is measured by the number of ancestor classes [8]. This metric measures understandability, reusability and testability as follows:

1. The deeper a class is, within the hierarchy, greater the number of methods to be inherited. This makes the deep class more complex to predict its behavior.
2. Deeper tree constitute greater design complexity, since more methods and classes are involved.
3. The deeper the inheritance tree means more potential is required for reuse.

C. Number of Children (NOC)

NOC is the number of immediate subclasses of a class in the hierarchy. This explains how much the potential influence a class can have on the design and on the system hierarchy. This metric measures efficiency, reusability, and testability as follows:

1. The higher the number of children means greater the improper abstraction of the parent and may be a case of misuse of sub-classing.
2. The higher the number of children means higher reusability of class since inheritance is a form of reuse.

D. Response For a Class (RFC)

The RFC is the number of functions or procedures that can be potentially executed in a class. So this is the number of operations directly invoked by member operation in a class plus the number operations themselves [5, 13].

E. Coupling Between Object Classes (CBO)

CBO for the target class is count of the number of other classes to which it is coupled. A class is coupled to another if it uses the methods or instance variables of another class.

F. Lack of Cohesion in Methods (LCOM)

LCOM measures the dissimilarities between methods in a class by looking at the instance variable or attributes used by methods [8, 10-11]. Let M_1, M_2…., M_n are n methods in a class. Let (I_j) consists of set of all instance variables used by method M_i. There are n such sets $\{I_1\}$,…….$\{I_n\}$.Let

$P = \{(I_i, I_j) \mid I_i \cap I_j = 0\}$

$Q = \{((I_i, I_j) \mid I_i \cap I_j \neq 0\}$.

If all n sets $\{(I_1\}$ ... $(I_n)\}$ are 0 then p=0.

LCOM =| P | - | Q |, if | P | > | Q |

Table 2: C & K Metrics

Name of Class Metrics	Basic Component	UI Component	Application	Clock
WMC	5	9	13	11
RFC	5	14	27	25
DIT	0	1	2	2
NOC	1	2	0	0
LCOM	10	34	52	49
CBO	0	0	0	0

In Table 2, we have calculated the C K metrics suit applied on data set (Appendix-1 [14]). We observed that maximum inheritance level is 2 and the complexity between classes is not so high which increases the reliability and decreases the maintenance cost of the software. As we know that C K metrics applied on class level, from the result it seen that out of four classes UI Component class have NOC value 2 which increases the reusability of code.

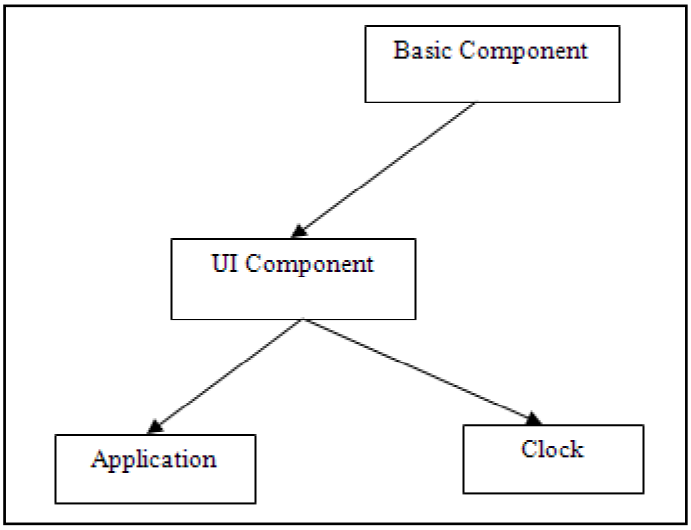

Fig. 1: Class Hierarchy

There is no metrics available in MOOD suit to find out the coupling and cohesion relationship between the classes where as in C K metrics suite one metric available that is CBO. Furthermore we can also observe that both metric suites are applicable at class, attribute, method and inheritance.

ISSN : 0976-8491 (Online) | ISSN : 2229-4333 (Print)

Table 3: Comparison Table for MOOD and C & K Metrics

Category	MOOD	Chidamber & Kemerar
Class	MHF, AHF, MIF, AIF, POF, COF	WMC, RFC, LCOM
Attribute	AHF, AIF	LCOM
Method	MHF, MIF, POF	WMC, RFC, LCOM
Cohesion/ Coupling		CBO
Inheritance	MIF, AIF	DIT, NOC

IV. Conclusion

Our result confirms that the MOOD metrics cannot be applicable for the message passing between classes where as C K metrics are applicable for the message passing between classes in a system via CBO therefore the C K metrics suite is better for the analysis of complexity between various classes in a system in compared to MOOD metrics suite for object oriented system.

The total hiding value (sum of AHF and MHF) is 1.435 in MOOD metric suite where as the hiding value in C & K metrics is 0 because this metric suite has no metrics for hiding factor. Hence it revals that the encapsulation is high in MOOD metric suite in comparison to C & K metric suite. So MOOD metric suite is much secure for object oriented system. As per our investigation we found that there is a large scope of research work for all kind of dynamic relationship between classes in message delivering ,activation , etc. for object oriented approach is possible in future.

References

[1] H.Barkmann, R.Lincke, W.Lowe,"Quantitative Evaluation of Software Quality Metrics in Open-Source Projects", in QuEST' 09, Bradford, UK. IEEE, 2009.

[2] V.R.Basili, L.C. Briand, W.L.Melo,"A Validation of Object-Oriented Design as Quality Indicators", IEEE TSE, Vol. 22, No. 10, pp.751-761, 1996.

[3] D.L.Parnas,"On the Criteria to be Used in Decomposing Systems into Modules", Comm. ACM, Vol.15, No.12, pp. 1,053-1058, 1972.

[4] B.A. Kitchenham, N.Fenton, S. Lawrence Pfleeger, "Towards a Framework for Software Measurement Validation", IEEE Trans Software Eng., Vol. 21, No. 12, pp. 929-944, 1995.

[5] R.Harrison, S. J. Counsell, R. V. Nithi,"An Evaluation of the MOOD Set of Object-Oriented Software Metrics", IEEE Transactions on Vol. 24, Issue 6, pp. 491-496, Jun 1998.

[6] G.Alkadi, L.D Craver,"Application of Metrics to Object-Oriented Designs", Proceedings of IEEE Aerospace Conference, Vol. 4, pp. 159-163, March 1998.

[7] F.B. Abreu, W. Melo,"Evaluating the Impact of Object-Oriented Design on Software Quality", Proc. 3rd International Software Metrics Symposium (METRICS' 96), IEEE, Berlin, Germany, Mar. 1996.

[8] S.R. Chidamber, C.F. Kemerer,"A Metrics Suite for Object-Oriented Design", IEEE Trans. Software, Vol. 20, No. 6, pp. 476-493, June 1994.

[9] N. Fenton, N.Ohlsson,"Quantitative Analysis of faults and failure in a complex software system", IEEE Transactions on Software Engineering, 26(8), pp. 797-814, 2000.

[10] T.Gyimothy, R. Ference, I. Siket,"Emperical Validation of Object-Oriented Metrics on Open Source Software for Faults Prediction", IEEE Transactions on Software Engineering, 2005.

[11] S.R. Chidamber, D.P. Darcy, C.F. kemerer,"Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis", IEEE Trans. Software Eng., Vol. 24, pp. 629-639, 1998.

[12] T.J. Mccabe,"A Complexity Measure, "IEEE Transactions on Software Engineering", Vol. 2, pp. 308-320, December 1976.

[13] R. Subramanyam, M.S Krishnan,"Emperical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects", IEEE Computer Society, 2003.

[14] D.A.Young,"Object-Oriented Programming with C++ and OSF/MOTIF", Prentice-Hall, 1992.

Source Code Appendix- 1

```
Class BasicComponent {
protected:
char *_name;
Widget _w;

BasicComponent (const char *); //Constructor
public:
virtual ~BasicComponent(); // Destructor
virtual void manage ();
virtual void unmanage();
const Widget baseWidget() { return _w; }
};

class UI Component : public Basic Component {
private:
static void widgetDestroyedCallback
( Widget, XtPointer, XtPointer );
protected:
UIComponent ( const char * ); //Constructor
void installDestroyHandler();
virtual void widgetDestroyed();
void setDefaultResources
( const Widget , const String *);
void getResources
( const XtResourceList, const int );
public:
virtual ~UIComponent(); // Destructor
virtual void manage();
virtual const char *const className()
{ return "UIComponent"; }
};

class Application : public UIComponent {
// Allow main and MainWindow to access protected
member functions
#if (XlibSpecificationRelease>=5)
friend void main ( int, char ** );
#else
friend void main ( unsigned int, char ** );

#endif
friend class MainWindow;
private:
void registerWindow ( MainWindow * );
void unregisterWindow ( MainWindow * );
protected:
```

```
Display *_display;
XtAppContext _appContext;
// Functions to handle Xt interface
#if (XlibSpecificationRelease>=5)
virtual void initialize ( int *, char ** );
#else
virtual void initialize(unsigned int *,char **);
#endif
virtual void handleEvents();

char *_applicationClass;
MainWindow **_windows;
int _numWindows;
public:
Application ( char * ); // Constructor
virtual ~Application(); // Destructor
void manage();
void unmanage();
void iconify();
Display *display()
{ return _display; }
XtAppContext appContext()
{ return _appContext; }
const char *applicationClass()
{ return _applicationClass; }
virtual const char *const className()
{ return "Application"; }
};
```

class Clock : public UIComponent {

```
private:
int _delta; // The time between ticks
XtIntervalId _id; // Xt Timeout identifier
virtual void timeout(); // Called every delta
virtual void speedChanged ( int );
static void timeoutCallback
( XtPointer, XtIntervalId * );
static void speedChangedCallback
( Widget, XtPointer, XtPointer );
protected:
virtual void tick()= 0;
public:
Clock ( Widget, char *,
int // Minimum speed
int ); // Maximum speed
~Clock (); // Destructor
void stop(); // Stop the clock
void pulse(); // Make the clock tick once
void start(); // Start or restart the clock
virtual const char *const className()
{ return ( "Clock" ); }
};
```

Ganesh Chandra was born at Kanpur, India. He received the B.Tech. Degree in Computer Science and Engineering in 2009 from Dr. Ambedkar Institute of Technology for Handicapped Kanpur, India. He is currently pursuing M. Tech in Computer Science and Engineering from Kamla Nehru Institute of Technology, Sultanpur, U.P, India.

Dharmendra Lal Gupta is currently working as an Assistant Professor in the Department of Computer Science & Engineering at KNIT, Sultanpur (U.P.) India. And he is also pursuing his Ph.D. in Computer Science & Engineering form Mewar University, Chittorgarh (Rajasthan). He received B. Tech. (1999) from Kamla Nehru Institute of Technology (KNIT) Sultanpur, in Computer Science & Engineering, M.Tech. Hon's (2003) in Digital Electronics and Systems from Kamla Nehru Institute of Technology (KNIT) Sultanpur. His research interests are Cryptography and Network Security, Software Quality Engineering, and Software Engineering.

Dr. Anil Kumar Malviya is an Associate Professor in the Computer Science & Engineering Department at Kamla Nehru Institute of Technology, (KNIT), Sultanpur. He received his B.Sc. & M.Sc. both in Computer Science from Banaras Hindu University, Varanasi respectively in 1991 and 1993 and Ph.D. degree in Computer Science from Dr. B.R. Ambedkar University; Agra in 2006.He is Life Member of CSI, India. He has published about 25 papers in International/ National Journals, conferences and seminars. His research interests are Data mining, Software Engineering, Cryptography & Network Security.