Metrics Identification for Measuring Object Oriented Software Quality

Aman Kumar Sharma, Arvind Kalia, Hardeep Singh

***Abstract*— In the modern era many object-oriented quality suites exists for assessing software quality against features of object-oriented design and as well as against the factors of evaluating quality. This paper presents a review of quality metrics suites namely, MOOD, CK and Lorenz & Kidd, and then selects some metrics and discards other metrics based on the definition and capability of the metrics.**

***Index Terms*— CK Suite, Lorenz and Kidd Suite, Metrics, MOOD Suite, Software Quality.**

I. INTRODUCTION

The contribution of metrics to the overall objective of software quality is understood and fully recognized by the software engineering community in general [1] and particularly emphasized by the software quality community [2]. Process and product metrics can help both managing activities, such as scheduling, costing, staffing and controlling. Also, the engineering activities such as analyzing, designing, coding, documentation and testing are helped by the software metrics. Since the early days of computer science many approaches quantifying the internal structure of procedural software system have emerged [3]. Some of those traditional metrics can still be used with the object-oriented paradigm, especially at the method level such as Lines Of Code and Number of Methods [4]. However, the need to quantify the distinctive features of object-oriented paradigm gave birth, in recent years, to new metric suites. Most of these sets have yet to be experimentally validated. This validation step usually consists of correlation studies between internal (design) and external (attributes) [5]. Software metrics is the measuring property or attribute to measure quality of a software object related to software project of any size. Object-oriented approach is capable of classifying the problem in terms of objects and provides paybacks like efficiency, maintainability, reliability, portability and usability. Object-oriented metrics are useless if they are not mapped to software quality parameters. There are numerous metric sources available to predict quality of the software namely CK, MOOD and Lorenz and Kidd. The main goal of this paper is to evaluate the metrics suites by presenting theoretical study of three object-oriented metric suites namely MOOD, CK and Lorenz and Kidd.

II. REVIEW OF METRICS

The software metrics considered in this study are: MOOD, CK and Lorenz and Kidd metrics suite.

A. MOOD Metric Suite

The Metrics for Object Oriented Design (MOOD) suite was proposed by Fernando Brito and Rogerio Carpuca in 1994 with an objective to enable identify quality in Object Oriented Design (OOD) by means of quantitative measurement of the object-oriented paradigm abstractions comprising of factors responsible for internal quality and to be able to express external quality attributes as functions of these metrics. MOOD suite includes six metrics, which have values as a measure of the presence of degree of OOD attributes. Hence their values range from 0 to 1. The MOOD Metrics are as follows:

(i). Method Hiding Factor (MHF)
(ii). Attribute Hiding Factor (AHF)
(iii). Metric Inheritance Factor (MIF)
(iv). Attribute Inheritance Factor (AIF)
(v). Polymorphism Factor (PF)
(vi). Coupling Factor (CF)

Each of these metrics are associated to a basic characteristics of OOD paradigm as encapsulation is related to MHF and AHF, inheritance pertains to MIF and AIF, polymorphism to PF and message passing to CF [6]. The software quality factors are numerous of which the most important and common as proposed by researchers are maintainability, portability, usability, efficiency and reliability [7]. MOOD metric suite has no binding to any Object-Oriented Programming language, the values are computable using C++, JAVA or any other object-oriented programming language used for software development.

(i). Method Hiding Factor

Method Hidden is the sum of the invisibilities of all methods defined in all classes. The percentage of the total classes from which the method is not visible is the invisibility method. MHF is the ratio of method hidden with total number of classes.

(ii). Attribute Hiding Factor

Similar to MHF, AHF is the ratio of attributes hidden to the total data members defined. The attributes hidden are the sum of the invisibilities of all attributed are defined in all classes. The invisibility of an attribute is the percentage of the total classes from which the attributes are not visible [8].

Regarding MHF and AHF validation criteria it is directly measured that increasing values of hiding would imply less complexity, more understanding and higher degree of maintainability, thereby resulting in good quality of software. Understanding software is usable and less complex software has high efficiency.

Manuscript Received on November, 2012.

Aman Kumar Sharma, Computer Science Department, Himachal Pradesh University, Shimla, India.
Dr. Arvind Kalia, Computer Science Department, Himachal Pradesh University, Shimla, India.
Dr. Hardeep Singh, Computer Science & Engineering Department, Guru Nanak Dev University Amritsar India.

(iii). Metric Inheritance Factor

MIF is a direct measure of inheritance. Inheritance for a method is the sum of inherited methods in all classes of the software. The MIF is the ratio of method inheritance with the total number of available methods, i.e. locally defined plus inherited for all classes [5].

(iv). Attribute Inheritance Factor

AIF is the ratio of the sum of inherited attributes in all classes of the system to the total number of available attributes for all classes.

MIF and AIF directly measures inheritance levels for methods and attributes respectively. Higher values of MIF and AIF directly imply more inheritance. MIF measures complexity in association to message passing dependencies among various methods of different classes. AIF evaluates the attributes being accessed from different classes [9]. More inheritance implies higher levels of complexity and reduces understandability.

(v). Polymorphism Factor

PF is the number of methods that redefine inherited methods, divided by maximum number of possible distinct polymorphic situations [10]. The contribution of all overriding methods to the relative amount of polymorphism can be considered to be equivalent. With the metric using units appropriately and the metric suite dimensionally consistent, the PF metric is a valid metric to measure the potential of polymorphism [11]. PF decreases understanding.

(vi). Coupling Factor

The CF metric is a measure of coupling between classes excluding coupling arising due to inheritance. CF is computed by considering all possible pair wise sets of classes and validating whether the classes in the pair are related by message passing. CF is the direct measure of the size of a relationship between two classes. CF is evaluated as the ratio of the possible number of couplings in the software to the actual number of couplings not imputable to inheritance [11]. As high degree of interclass relationship will have a high CF value. On the contrary, CF does not provide valid results in evaluation of quality. It is due to the fact that high value of CF does not indicate high or low complexity. As it is feasible to construct a simple system which is highly coupled and also it is possible to have a complex system with negligible value of CF. Even for encapsulation measure CF value does not provide help. Similarly for understanding and maintainability quality factors, a class with high value of CF may exist, but having no degree of impact on understandability and maintainability. CF plays no impact on reusability too; a low degree of coupling may have high degree of reuse thru inheritance. Consequently, it is concluded that CF is not a valid measure of quality.

MOOD metrics suite is a very well defined validated through mathematical formulas, supported by a tool, provides thresholds to judge the metrics collected from a given design. The metrics of MOOD suite are at project level. Empirical study have concluded that the inheritance metrics have negative impact on encapsulation i.e. AIF and MHF are negatively related to each other, whereas PF and MHF are strongly positively related [8]. Regarding the metrics of MOOD it is validated that MIF, AIF, MHF, AHF and PF are valid measures of quality [12].

B. CK Suite

Shyam R. Chidamber and Chris F. Kemerer (CK) developed a metrics suite for OOD. CK metrics suite plays a significant role to know the design aspects of the software and to enhance the quality of software [13]. Most of the metrics suites are built upon the original CK metrics suite [14]. The CK metrics suite is designed to provide a summary of the overall quality of object oriented software and is available at the class level [15]. The metrics suite is associated to each small segment of the software providing the in depth information of the software and its quality. The CK metrics suite proposed class based six metrics, which assess different characteristics of OOD, having the following metrics:

(i). Weighted Methods per Class (WMC)
(ii). Response For a Class (RFC)
(iii). Lack of Cohesion of Methods (LCOM)
(iv). Depth of Inheritance Tree (DIT)
(v). Number Of Children (NOC)
(vi). Coupling Between Object classes (CBO).

The six metrics of CK Suite are described as follows:

(i). Weighted Methods per Class

WMC is used to measure the understandability, reusability, complexity and maintainability. WMC is the count of methods implemented within a class or the sum of the complexities of the methods. Children inherit all of the methods defined in a class thereby higher values of WMC imply more complexity and less understandability. Classes with large number of methods are likely to be more application specific, limiting the possibility of reuse [16]. WMC decreases understandability and reliability.

(ii). Response For a Class

RFC is the total number of all methods within a set that can be invoked in response to message sent to an object to perform an operation [16]. All methods accessible within the class hierarchy are included in the count. RFC measures complexity, if the number of invoked methods is high, for a message then complexity increases and maintainability decreases, thus the quality of the software decreases.

(iii). Lack of Cohesion of Methods

LCOM is the difference between the number of methods whose similarity is zero and the number of methods whose similarity is not zero. The similarity of two methods is the number of attributes used in common. However, Basili *et al.* [17], Briand *et al.* [18] and [19] noted problems in the LCOM metrics, a value of zero of LCOM is not an evidence of cohesiveness and also very high value of LCOM does not depict any inference. LCOM metric makes it difficult, if not impossible, to define a unit and to measure quality [20]. LCOM does not quantify quality accurately and is not a good measure.

(iv). Depth of Inheritance Tree

DIT assess how deep, in a class hierarchy, a class is. DIT measures maintainability, reusability. A class with small value of DIT, has much potential for reuse as deeper classes are difficult to maintain, due to increased efforts required to monitor its functionality.

(v). Number Of Children

NOC is a measure of the number of classes associated with a given class using an inheritance relationship. A class having many children is a bad class with a bad design [21]. Lower value of NOC helps in maintainability and complexity. Software with controlled values of NOC has good quality of the software.

(vi). Coupling Between Object classes

CBO of a class is defined as the number of other classes to which it is coupled. CBO determines whether a class is using an attribute in another class or not. CBO is beneficial in judging the complexity of testing and reusability [22].
Among the proposed CK metrics, the effective metrics are WMC, RFC, DIT, NOC and CBO.

C. Lorenz and Kidd Suite

Mark Lorenz and Jeff Kidd in 1994 introduced eleven metrics to quantify software quality evaluation which were applicable to class diagrams. The metrics were categorized into three groups namely

(i). Class Size Metrics
(ii). Class Inheritance Metrics
(iii). Class Internals Metrics.

Each of these groups further contained metrics, which are listed as below:

(i). Class Size Metrics

The Class size metrics dealt with quantifying a class by counting the Number of Public Methods (NPM), Number of Methods (NM), Number of Public Variables (NPV), Number of Variables per class (NV), Number of Class Variables (NCV) and Number of Class Methods (NCM).

(ii). Class Inheritance Metrics

Inheritance based quality measurement metrics become part of class inheritance metrics namely, Number of Methods Inherited (NMI), Number of Methods Overridden (NMO) and Number of New Methods (NNA).

(iii). Class Internals Metrics

General features of classes are evaluated using class internals metrics. Average Parameters per Method (APM) and Specialization IndeX (SIX) are computed in this category.

The Lorenz and Kidd Suite are criticized [23] for being mere counts of class properties. Counting the number of public methods and variables in different ways does not evaluate quality factors [12].

III. DESIRABLE PROPERTIES OF OOD QUALITY METRICS

Based on the review of the existing software metrics suite, a list of parameters is defined to accept or discard software metric. Lacking any of these properties will result in an inapplicable quality metrics.

A. Precisely defined metrics

Ambiguity in metrics definition allows many interpretations for the same metric. A mathematical formula or a clear explanation of the method of calculation of the metric should exist.

B. Empirically validated software quality metrics

Metrics suites without validation are always in doubt concerning their correctness. The MOOD metrics suite and CK metrics suite have been validated in several studies [5] [24] [25] and [17] [22] [26] respectively. Empirical validation for the Lorenz and Kidd metrics suite are lacking.

C. Interpretation of metrics

Lord Kevin quote “The degree to which you can express something in numbers is the degree to which you really understand it” is self explanatory to state that numbers do not have a meaning of their own, till the values are interpreted to make decisions.

D. Relationship between metrics and quality factors

The metrics value should address to the quality factors. An explicit relationship of increase or decrease in metrics value having implication on software quality factors be made.

E. Metrics computable at any stage

At any stage especially at the initial stage or before completion of the software values for metrics may be calculated. Mid way assessment of software metrics certainly helps in improvement of software quality.

The results of assessing the software metrics against the software quality desirable properties are summarized in TableI:

Table I: Assessment of metrics against the desirable properties

Property / Metrics	Precise Defined Metric	Empirical Validation	Inter-pretation ofMetric	Relation between Metrics & Factors	Computable at any stage
MHF	Yes	Yes	Yes	Yes	Yes
AHF	Yes	Yes	Yes	Yes	Yes
MIF	Yes	Yes	Yes	Yes	Yes
AIF	Yes	Yes	Yes	Yes	Yes
PF	Yes	Yes	Yes	Yes	Yes
CF	Yes	Yes	No	No	No
WMC	Yes	Yes	Yes	Yes	Yes
RFC	Yes	Yes	Yes	Yes	Yes
LCOM	No	No	Yes	No	No
DIT	Yes	Yes	Yes	Yes	Yes
NOC	Yes	Yes	Yes	Yes	Yes
CBO	Yes	Yes	Yes	Yes	Yes
Class Size Metrics	Yes	No	No	No	Yes
Class Inheritance Metrics	Yes	No	No	No	Yes
Class Internals Metrics	Yes	No	No	No	Yes

The first column has the set of all software metrics such as MHF, AHF MIF and so on. The desirable properties are mentioned in top row. The "Yes" in the Table I denotes that the metric satisfies the desirable property, on the other hand a "No" represents that the metric does not satisfy the desirable property. The Table I clearly depicts on the basis of desirable properties the Lorenz and Kidd metrics suite is unfit for evaluation of software quality. Also LCOM and CF have failed, in evaluation of the metric, for evaluation of quality as presented in Table I.

IV. CONCLUSION AND FUTURE SCOPE

In this paper a survey of three object oriented software metrics suite comprising of CK Suite, MOOD Suite and Lorenz and Kidd Suite was made. Keeping in view the significance of object-oriented the metrics suites evaluated in the study were from the object-oriented domain. The work of CK suite was seminal in defining metrics, binding scope of metrics, class level based and validating quality. Similarly MOOD suite is well defined, project level based, mathematically computable and provides thresholds that could be used to judge the metrics collected from a given design. However, on the contrary the Lorenz and Kidd suite is neither validated in the existing studies nor the metrics of Lorenz and Kidd suite are capable to measure software quality. The Lorenz and Kidd metrics are statistical measures for software in terms of counting: the number of methods under various categories, the number of variables, etc. The Lorenz and Kidd metrics seems to be ineffective for measuring software quality. From among the suites analyzed the study has recommended metrics which are useful in evaluation of software quality. The metrics namely, WMC, RFC, DIT, NOC and CBO are suitable for evaluation of software quality from the CK Suite and whereas from the MOOD suite the appropriate metrics are MHF, AHF, MIF, AIF and PF. These ten metrics calculate as per all object-oriented characteristics i.e. encapsulation, inheritance and polymorphism. Based on the comparison and analysis it is concluded that the mentioned list of metrics is the most complete, comprehensive and supportive. Further studies and empirical validations may be made to strengthen the inference made in this study.

REFERENCES

1. Roger S. Pressman, "Software Engineering: A Practitioner's Approach", 6th ed., McGraw Hill International, 2005.
2. N. Fenton and S. Lawrence Pfleeger, "Software Metrics: A Rigorous Approach", 2nd ed., International Thomson Press, London, 1996.
3. H. Zuse, "Software Complexity: Measures and Methods", Walter de Gruyer, New York, 1991.
4. F. Brito e Abreu, and R. Carapuca, "Candidate Metrics for Object-Oriented Software within a Taxonomy Framework", Proceedings of AQUIS'93 (Achieving Quality In Software), Italy, 1993.
5. Fernando Brito e Abreu, and Walcelio Melo, "Evaluating the Impact of Object-Oriented Design on Software Quality", Proceedings of the third international Software Metrics Symposium (Metrics'96), IEEE, Germany 1996.
6. F. B. Abreu, "The MOOD Metrics Set", In Proceedings of ECOOP'95, Workshop on Metrics, 1995.
7. Aman Kumar Sharma, Arvind Kalia, and Hardeep Singh, "An Analysis of Optimum Software Quality Factors", IOSR Journal of Engineering, vol. 2 issue 4, 2012.
8. Aman Kumar Sharma, Arvind Kalia, and Hardeep Singh, "Empirical Analysis of Object Oriented Quality Suites", International Journal of Engineering and Advanced Technology (IJEAT), vol. 1 issue 4, 2012.
9. B. A. Kitchenham, N. Fenton, and S. Lawrence Pfleeger, "Towards a Framework for Software Measurement Validation", IEEE Transaction on Software Engineering, vol. 21, no. 12, 1995.
10. F. Brito e Abreu, M. Goulao, and R. Estevers, "Towards the Design Quality Evaluation of OO Software Systems", Proceedings in Fifth International Conference on Software Quality, 1995.
11. Rachel Harrison, Steve J. Counsell, and Reuben V. Nithi, "An Evaluation of the MOOD Set of Object-Oriented Software Metrics", IEEE Transactions on Software Engineering, vol. 24, no. 6, 1998.
12. Aline Lucia Baroni, and Fernando Brito e Abreu, "A Formal Library for Aiding Metrics Extraction", 4th International Wokshop on OO Rengineering, 2003.
13. M. Subramanyam, and R. Krishnan, "Empirical Analysis of CK Metrics for OOD Complexity: Implication for Software defect", IEEE transactions on software engineering, 2003.
14. Jagdish Bansiya, and Carl G. Davis, 2002, "A Hierarchical Model for Object-Oriented Design Quality Assessment", IEEE Transactions on Software Engineering, vol. 28 no. 1, 2002.
15. Aman Kumar Sharma, Arvind Kalia, and Hardeep Singh, "Taxonomy of Metrics for Assessing Software Quality", International Journal of Engineering Research and Technology (IJERT), vol. 1 Issue 06, 2012.
16. Gurdev Singh, Dilbag Singh, and Vikram Singh, "A Study of Software Metrics", International Journal of Computational Engineering and Management (IJCEM), vol. 11, 2011.
17. V.L. Basili, L. Briand, and W.L. Melo, "A Validation of Object-Oriented Metrics as Quality Indicators," IEEE Transactions Software Engineering, vol. 22 no. 10, 1996.
18. L.C. Briand, J. Wust, J.W. Daly, and D.V. Porter, "Exploring the Relationship Between design Measures and Software Quality in Object-Oriented Systems", Journal Systems and Software, vol. 51 no. 3, 2000.
19. R. Shatnawi, "An Investigation of CK Metrics Thresholds", ISSRE Supplementary Conference Proceedings, 2006.
20. Wei Li, "Another Metric Suite For Object-Oriented Programming", Journal of Systems and Software, vol. 44, no. 2, 1998.
21. Alexander Chatzigeorgiou, "Mathematical Assessment of Object-Oriented Design Quality", IEEE Transactions on Software Engineering, vol. 29 no. 11, 2003.
22. Shyam R. Chidamber, and Chris F. Kemerer, "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, vol. 20, no. 6, 1994.
23. R. Harrison, S. Counsell, and R. Nithi, "An Overview of Object-Oriented Design Metrics", Proceedings of the 8th International Workshop on Software Technology and Engineering Practice (STEP'97), 1997.
24. Khaled EL Emam, Saida Beniarbi, Nishith Goel, and Shesh Rai, "A Validation of Object-Oriented Metrics", National Research Council Canada Internal Report No. 43607.
25. Fernando Brito e Abreu, and Rogeria Carapuca, "Object-Oriented Software Engineering: Measuring and Controlling the Development Process", 4th International Conference on Software Quality, USA, 1994.
26. L.H. Rosenberg, and L. Hyatt, "Applying ant Interpreting Object Oriented Metrics", Proceedings of Software Technology Conference, Utah 1998.