An Analytical Study of Object-Oriented Metrics (A Survey)

Mrs. Mansi Aggarwal , Dr. Vinit Kumar Verma , Mr. Harsh Vardhan Mishra

Abstract: **The OO metrics are plays an important role in the software development. In this paper we are mostly focus on a set of OO metrics that can be used to measure the quality and effectiveness of an OO design. The metrics for OO design focus on measurements that are applied to the class and design characteristics. These measurements of OO metrics also permit the designers to access the software early in process and these changes will reduce the complexity of OO software and improve the continuing capability of the design. This paper also summarizes the existing metrics, which will guide the designers to support their OO design.**

***Keywords*— Keywords: Object-Oriented (OO), NOC, DIT.**

I. INTRODUCTION

Several researchers have proposed a variety of criteria [1-11] for evaluation and validation to which a proposed software metric should adhere. Amongst them, we can mention validation through measurement theory [2, 4, 6], IEEE standards [5] Kaner's framework [3], and Weyuker's properties [11]. However, most of the existing evaluation and validation criteria were proposed when procedural languages were dominant. After the adaptation of OO languages by the software industry, not too much effort has been made to develop a model/ framework for evaluating software complexity measures in the OO domain. There are some proposals for OO languages [12,13,14]; however, most of them cover only specific features of evaluation. For example, Z use's properties [15] for OO metrics are mathematical in nature and based on principles of measurement theory. The lack of proper guidelines for evaluation and validation of OO metrics motivate us to develop a new evaluation criterion which includes all the features required for evaluation of the OO metrics. For achieving this goal, first we have analyzed the available validation and evaluation criteria, extracted their important features, suggested additions/modifications (if required), then presented them in a formal way. The validity of the proposed model is evaluated by applying eleven different well-known OO metrics.

OO metrics are measurement tools to achieve quality in software process and product. However, in general, software measurement has not yet achieved the needed degree of maturity [9] and it needs standardization [16]. Existing proposals, such as Weyuker's properties [11] and the application of measurement theory in software engineering [2, 4, 6, 17, 18], are a topic of discussion [19,20,21]. We have also worked in the related area of software measurement and presented several papers. We have presented a paper on the usefulness of Weyuker's properties for procedural languages [24]. In another work, we have analysed how Weyuker's properties are used by the developers of three OO metrics [25]. We have previously performed experimentations to analyze the current situation of standard measurement activities in small and medium software companies [26]. We have also performed a study on the situation of the empirical validation of software complexity measures in practice, and we accordingly proposed a model [27]. The applicability of measurement theory on software complexity measures is also investigated in one of our previous works [22]. In the present paper we analyze the present practices used for evaluation and validation of OO metrics, and we accordingly present a model for evaluating OO metrics. We also propose a framework for evaluating software complexity measures but, the present paper is specifically for OO metrics, since OO languages do not share the same features with procedural languages.

2. Literature survey

Most of the software maintainability assessment model have been proposed and compared with other molds. Zhuo F. et al. (1993) proposed maintainability index (MI) that determine the maintainability of software system based upon the status of the source code, which show high correlation between assessments automated model and some expert evaluation [28]. Binkley A. et al. (1998) collect the data of maintenance for the development of project written in any language like C, C++, COBOL etc and produce a level of interaction between modules, which show low coupling were subjected for fewer maintenance effort and fewer maintenance fault and failures [29].

Muthana S. et. al. (2000) proposed that the linear prediction model which is being evaluated by some industrial software system to estimate the maintainability of large system and to identified some fault prone models to define impact rate, effort and error rate [30]. Kiewkanya M. et al. (2004) prescribed that object-oriented (OO) is ease of maintenance to provide better understandability and modifiability. It describes three technique discriminant techniques (correlation between maintainability and structural complexity), weighted score level technique (combination of understanding and modifiability) and weighted predicate level technology (combination of predicate understandability and modifiability). Rizvi S.W.A. et al. (2010) propose a MEMOOD model, which provide an opportunity to improve the maintainability or understandability of class diagram and consequently the maintainability in final software [31]. Gautam C. et al. (2011), describe that the compound

MEMOOD model is better the MEMOOD model to determine the maintainability of class diagram in terms of their understandability, modifiability, scalability and level of complexity [32].

Abreu et al. [33] provides a new classification framework for the TAPROOT. This framework was defined with the other two independent vectors these are category and granularity. Six categories of Object-Oriented metrics were defined are design metrics, complexity metrics, size metrics, quality metrics, productivity metrics and reuse metrics and also proposed three Levels of granularity are software, class and methods but no empirical/theoretical base for the metrics was provided.

M.Alshayeb et al. [34] have given two iterative procedures for the pragmatic study of OO metrics. They include the short-cycled agile process and the long cycled framework evolution process. By bserving the results, it can be seen that the design efforts and source lines of code added, changed, and deleted were triumphantly predicted by object-oriented metrics in short-cycled agile process where as in the case long-cycled framework process the same features were not successfully predicted by it. This has shown that the design and implementation changes during development iterations can be predicted by object-oriented Metrics, but the same cannot be the case with long-term development of an established system.

3. Classification of Object-Oriented Metrics:

3.1 Metrics for Object-Oriented Software Engineering (MOOSE):

Chidamber and Kemerer (CK) et al. [35] proposed some metrics that have generated a significant amount of interest and are currently the most well known object-oriented suite of measurements for Object-Oriented software. The CK metrics suite consists of six metrics that assess different characteristics of the object-oriented design are-

(i)Weighted Methods per Class (WMC): This measures the sum of complexity of the methods in a class. A predictor of the time and effort required to develop and maintain a class we can use the number of methods and the complexity of each method. A large number of methods in a class may have a potentially larger impact on the children of a class since the methods in the parent will be inherited by the child. Also, the complexity of the class may be calculated by the cyclomatic complexity of the methods. The high value of WMC indicates that the class is more complex as compare to the low values.

(ii)Depth of Inheritance Tree (DIT): DIT metric is used to find the length of the maximum path from the root node to the end node of the tree. The following figure shows that the value of the DIT from a simple hierarchy. DIT represents the complexity and the behavior of a class, and the complexity of design of a class and potential reuse. Thus it can be hard to understand a system with many inheritance layers. On the other hand, a large DIT value indicates that many methods might be reused. A deeper class hierarchy indicates that the more methods was used or inherited through which this making more complex to predict the behavior of the class and the deeper tree indicates that there is high complexity in the design because all of the facts contained more methods and class are involved. A deep hierarchy of the class may indicates a possibility of the reusing an inherited methods.

(iii)Number of children (NOC): According to Chidamber and Kemerer, the Number of Children (NOC) metric may be defined for the immediate sub class coordinated by the class in the form of class hierarchy[38,39]. These points are come out as NOC is used to measure that "How many subclasses are going to inherit the methods of the parent class". The greater the number of children, the greater the potential for reuse, since inheritance is a form of reuse. The greater the number of children, the greater the likelihood of improper abstraction of the parent class. The number of children also gave an idea of the potential influence for the class which may be design.

(iv)Coupling between Objects (CBO): CBO is used to count the number of the class to which the specific class is coupled. The rich coupling decrease the modularity of the class making it less attractive for reusing the class and more high coupled class is more sensitive to change in other part of the design through which the maintenance is so much difficult in the coupling of classes. The coupling Between Object Classes (CBO) metric is defined as "CBO for a class is a count of the number of non-inheritance related couples with classes". It claimed that the unit of "class" used in this metric is difficult to justify, and suggested different forms of class coupling: inheritance, abstract data type and message passing which are available in object-oriented programming.

(v)Response for class (RFC): The response set of a class (RFC) is defined as set of methods that can be executed in response and messages received a message by the object of that class. Larger value also complicated the testing and debugging of the object through which, it requires the tester to have more knowledge of the functionality. The larger RFC value takes more complex is class is a worst case scenario-value for RFC also helps the estimating the time needed for time needed for testing the class.

(vi)Lack of Cohesion in Methods (LCOM): This metric is used to count the number of disjoints methods pairs minus the number of similar method pairs used. The disjoint methods have no common instance variables in the methods, while the similar methods have at least one common instance variable. It is used to measuring the pairs of methods within a class using the same instance variable. Since cohesiveness within a class increases encapsulation it is desirable and due to lack of cohesion may imply that the class is split in to more than two or more sub classes. Low cohesion in methods increase the

complexity, when it increases the error proneness during the development is so increasing.

3. 2 Extended Metrics For Object-Oriented Software Engineering (EMOOSE):

W.Li et al. [40] proposed this metrics of the MOOSE model. They may be described as-

(i)Message Pass Coupling (MPC): It means that the number of message that can be sent by the class operations.

(ii)Data Abstraction Coupling (DAC): It is used to count the number of classes which an aggregated to current class and also defined the data abstraction coupling.

(iii)Number of Methods (NOM): It is used to count the number of operations that are local to the class i.e. only those class operation which can give the number of methods to measure it.

(iv)Size1:- It is used to find the number of line of code.

(v)Size2:-It is used to count the number of local attributes & the number of operation defined in the class.

3.3 Metrics for Object-Oriented Design (MOOD):

F.B. Abreu et al. [41] defined MOOD (Metrics for Object-Oriented Design) metrics. MOOD refers a structural model of the OO paradigm like encapsulation as (MHF, AHF), inheritance (MIF, AIF), polymorphism (POF), and message passing (COF). Each of the metrics was expressed to measure where the numerator defines the actual use of any one of the feature for a particular design [42]. In MOOD metrics model, there are two main features are methods and attributes. Attributes are used to represent the status of object in the system and methods are used to maintained or modifying several kinds of status of the objects [43]. These metrics are defined as:

(i)Method Hiding Factor (MHF): MHF is defined as the ratio of the sum of the invisibilities of all methods defined in all classes to the total number of methods defined in the system under consideration. The invisibility of a method is the percentage of the total classes from which this method is not visible.

(ii)Attribute Hiding Factor (AHF): AHF is defined as the ratio of the sum of the invisibilities of all attributes defined in all classes to the total number of attributes defined in the system under consideration.

(iii)Method Inheritance Factor (MIF): MIF is defined as the ratio of the sum of the inherited methods in all classes of the system under consideration to the total number of available methods (locally defined plus inherited) for all classes.

(iv)Attribute Inheritance Factor (AIF): AIF is defined as the ratio of the sum of inherited attributes in all classes of the system under consideration to the total number of available attributes (locally defined plus inherited) for all classes.

(v)Polymorphism Factor (PF): PF is defined as the ratio of the actual number of possible different polymorphic situation. MIF & AIF are used to measure the inheritance of the class & also provide the similarity into the classes. CF is used to measure the coupling between the classes. the coupling are of two types static & dynamic coupling, due to which is increase the complexity of the class & reduce the encapsulation & potential reuse that provide better maintainability. Software developers for the object-oriented system always avoid the high coupling factor. Polymorphism potential of the class are used to measure the polymorphism in the particular class & also arise from inheritance.

3.4 Goal Question Metrics (GQM):

V. L. Basili [43] developed GQM approach. This approach was originally defined for evaluating defects for a set of projects in the NASA Goddard Space Flight Center environment. He has also provided the set of sequence which are helpful for the designers. The goal of GQM is to express the meaning of the templates which covers purpose, perspective and environment; a set of guidelines also proposed for driving question and metrics. It provides a framework involving three steps:

(i) List major goals of the development or maintenance project.
(ii) Derive from each goal the questions that must be answered to determine if the goals are being met.
(ii) Decide what must be measured in order to be able to answer the questions adequately.

Goal (Conceptual level): A goal is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view, relative to a particular environment. Objects of measurement are products, processes and resources.

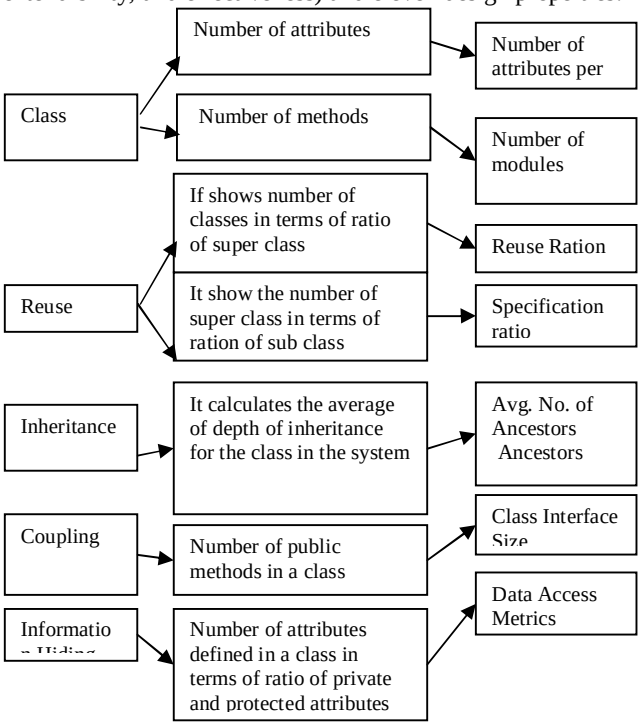
Question (Operational level): A set of questions is used to characterize the way the assessment/achievement of a specific goal is going to be performed based on some characterizing model.

Metric (Quantitative level): A set of data is associated with every question in order to answer it in a quantitative way. This data can be objectives and subjective, if they depend only on the objects that can be measured and not on the viewport from which they may be taken. For example, number of versions of a document, staff hours spent on a task, size of a program.

3.5 Quality Model for Object-Oriented Design (QMOOD):

The QMOOD [44] is a comprehensive quality model that establishes a clearly defined and empirically validated model to assess object-oriented design quality attributes such as understandability and reusability, and relates it through mathematical formulas, with structural object-oriented design properties such as encapsulation and coupling. The QMOOD

model consists of six equations that establish relationship between six object-oriented design quality attributes (reusability, flexibility, understandability, functionality, extendibility, and effectiveness) and eleven design properties.



QMOOD Metrics [25]

The whole description for QMOOD can be get from the Bansiya's thesis through which, The QMOOD metrics can further classified into two measures are:

System Measures: System measures describe such metrics are DSC (Design Size in Metrics), NOH (Number of Hierarchies), NIC (Number of Independent classes), NSI (Number of Single Inheritance), NMI (Number of multiple Inheritance), NNC (Number of Internal Classes), NAC (Number of Abstract Classes), NLC (Number of Leaf Classes), ADI (Average Depth of Inheritance), AWI (Average Width of Classes), ANA (Average Number of Ancestors).

Class Measures: Class measure metrics are those metrics which can define some metrics are MFM (Measure of Functional Modularity), MFA (Measure of Functional Abstraction), MAA (Measure of Attribute Abstraction), MAT (Measure of Abstraction), MOA (Measure of Aggregation), MOS (Measure of Association), MRM (Modeled Relationship Measure), DAM (Data Access Metrics), OAM (Operation Access Metrics), MAM (Member Access Metrics), DOI (Depth of Inheritance), NOC (Number of Children), NOA (Number of Ancestor), NOM (Number of Methods), CIS (Class Interface Size), NOI (Number of Inline Method), NOP (Number of Polymorphic Method), NOO (Number of Overloaded Operators), NPT (Number of Unique Parameter Types), NPM (Number of Parameter per Method), NOA (Number of Attributes), NAD (Number of Abstract Data Types), NRA (Number of Reference Attributes), NPA (Number of Public Attributes), CSB (Class Size in Bytes), CSM (Class Size in Metrics), CAM (Cohesion Among Methods of class), DCC (Direct Class Coupling), MCC (Maximum Class Coupling), DAC (Direct Attribute based Coupling), MAC (Maximum Attribute based Coupling), DPC (Directed Parameter based Coupling), MPC (Maximum Parameter based Coupling), VOM (Virtual ability Of Method), CEC (Class Entropy Complexity), CCN (Class Complexity based on Data), CCP (Class Complexity based on method Parameter), CCM (Class Complexity based on Members).

3.5 LI W. METRICS

Li et al. [45] proposed six metrics are Number of Ancestor Classes (NAC), Number of Local Methods (NLM), Class Method Complexity (CMC), Number of Descendent Classes (NDC), Coupling Through Abstract data type (CTA), and Coupling through Message Passing (CTM).

(i)Number of Ancestor Classes (NAC): The Number of Ancestor classes (NAC) metric proposed as an alternative to the DIT metric measures the total number of ancestor classes from which a class inherits in the class inheritance hierarchy. The theoretical basis and viewpoints both are same as the DIT metric. In this the unit for the NAC metric is "class", justified that because the attribute that the NAC metric captures is the number of other classes' environments from which the class inherits.

(ii)Number of Local Methods (NLM): The Number of Local Methods metric (NLM) is defined as the number of the local methods defined in a class which are accessible outside the class. It measures the attributes of a class that WMC metric intends to capture. The theoretical basis and viewpoints are different from the WMC metric. The theoretical basis describes the attribute of a class that the NLM metric captures. This attribute is for the usage of the class in an object-oriented design because it indicates the size of a class's local interface through which other classes can use the class. They stated three viewpoints for NLM metric as following:

1) The NLM metric is directly linked to a programmer's effort when a class is reused in an Object-Oriented design. More the local methods in a class, the more effort is required to comprehend the class behavior.
2) The larger the local interface of a class, the more effort is needed to design, implement, test, and maintain the class.
3) The larger the local interface of a class, the more influence the class has on its descendent classes.

(iii)Class Method Complexity (CMC): The Class Method Complexity metric is defined as the summation of the internal structural complexity of all local methods. The CMC metric's theoretical basis and viewpoints are significantly different from WMC metric. The NLM and CMC metrics are fundamentally different as they capture two independent attributes of a class. These two metrics affect the effort required to design, implement, test and maintain a class.

(iv)Number of Descendent Classes (NDC): The Number of Descendent Classes (NDC) metric as an alternative to NOC is defined as the total number of descendent classes (subclass) of a class. The stated theoretical basis and viewpoints indicate that NOC metric measures the scope of influence of the class on its sub classes because of inheritance. Li claimed that the NDC metric captures the classes attribute better than NOC.

(v)Coupling through Abstract Data Type (CTA): The Coupling through Abstract Data Type (CTA) is defined as the total number of classes that are used as abstract data types in the data-attribute declaration of a class. Two classes are coupled when one class uses the other class as an abstract data type [45]. The theoretical view was that the CTA metric relates to the notion of class coupling through the use of abstract data types. This metric gives the scope of how many other classes' services a class needs in order to provide its own service to others.

(vi)Coupling through Message Passing (CTM): The Coupling through Message Passing (CTM) defined as the number of different messages sent out from a class to other classes excluding the messages sent to the objects created as local objects in the local methods of the class. Two classes can be coupled because one class sends a message to an object of another class, without involving the two classes through inheritance or abstract data type [Li., 98]. Theoretical view given was that the CTM metric relates to the notion of message passing in object-oriented programming. The metric gives an indication of how many methods of other classes are needed to fulfill the class' own functionality.

3.6 SATC's Metrics

Rosenberg Linda [46] proposed to select OO metrics that supports the goal of measuring the code, quality, result and they proposed many object-oriented metrics due to lack of theoretical basis and that can be validated. These metrics may be used to evaluate the OO concepts like methods, coupling and inheritance and mostly focus on both of the internal and external efficiency measures of the psychological complexity factors that affect the ability of the programmer. It proposed three traditional metrics and six new metrics for the object-oriented system metrics-

Traditional Metrics

(i)Cyclomatic Complexity (CC): Cyclomatic Complexity is used to measure the complexity of an algorithm in a method of class. Cyclomatic Complexity of methods can be combined with other methods to measure the complexity of the class. Generally, this is only used for the evaluation of quality attribute complexity.

(ii)Line of Code: It is a method used to evaluate the ease of understandability of the code by the developer and the maintainer. It can easily be counted by the counting the number of lines for the code and so on. Generally, used to measure the reusability and maintainability.

New OO Metrics

The six new OO metrics are may be discussed as:

(i)Weight Method per Class (WMC): It is used to count the methods implemented within a class. The number of methods and complexities involved as predictors, how many time and effort is required to develop and maintain the class.

(ii)Response for a Class (RFC): It is used to the combination of the complexity of a class through the number of methods and the communication of methods with other classes. This is used to evaluate the understandability and testability.

(iii)Lack of Cohesion of Method (LCOM): Cohesion is a degree of methods through which all the methods of the class are inter-related with one another and provide a well bounded behavior. It also measures the degree of similarity of methods by data inputs variables and attributes. Generally, ii is used to evaluate the efficiency and reusability.

(iv)Depth of Inheritance Tree (DIT): Inheritance is a relationship between the class that enables the programmer to use previously defined object including the operators and variables. It also helps to find out the inheritance depth of the tree from current node to the ancestor node. It is used to evaluate the reusability, efficiency, understandability and testability.

(v)Number of Children (NOC): This is used to measure the subclass subordinate to a class in the hierarchy. Greater the number of children means greater reusability and inheritance i.e. in the form of reuse. Generally, it is used to measure efficiency, testability and reusability.

SATC focused on some selected criteria for the OO metrics as:

(i) Efficiency of constructor design to decrease architecture complexity.
(ii) Specification of design and enhancement in testing structure
(iii) Increase capacity of psychological complexity.

4. Conclusion and future aspect:

This paper introduces the basic metric suite for object-oriented design. The need for such metrics is particularly acute when an organization is adopting a new technology for which established practices have yet to be developed. It is unlikely that universally valid object-oriented quality measures and models could be devised, so that they would suit for all languages in all development environments and for different kind of application domains. Therefore measures and models should be investigated and validated locally in each studied environment. It should be also kept in mind that metrics are

only guidelines and not rules. They are guidelines that give an indication of the progress that a project has made and the quality of design.

References

[1] Fenton N. (1993) New Software Quality Metrics Methodology Standards Fills Measurement Needs', IEEE Computer, April, pp. 105-106
[2] Briand L. C., Morasca S., Basili V. R. (1996) Property-based Software Engineering Measurement, IEEE Transactions on Software Engineering, 22(1), pp. 68-86
[3] Kaner C. (2004) Software Engineering Metrics: What do They Measure and How Do We Know?' In Proc. 10th Int. Software Metrics Symposium, Metrics, pp. 1-10
[4] Fenton N. (1994) Software Measurement: A Necessary Scientific Basis', IEEE Transactions on Software Engineering, 20(3), pp. 199-206
[5] IEEE Computer Society (1998) Standard for Software Quality Metrics Methodology. Revision IEEE Standard, pp. 1061-1998
[6] Kitchenham B., Pfleeger S. L., Fenton N. (1995) Towards a Framework for Software Measurement Validation. IEEE Transactions on Software Engineering, 21(12), pp. 929-943
[7] Morasca S. (2003) Foundations of a Weak Measurement-Theoretic Approach to Software Measurement. Lecturer Notes in Computer Science LNCS 2621, pp. 200-215
[8] Wang Y. (2003) The Measurement Theory for Software Engineering. In Proc. Canadian Conference on Electrical and Computer Engineering, pp. 1321-1324
[9] Zuse H. (1991): Software Complexity Measures and Methods, Walter de Gruyter, Berline
[10] Zuse, H. (1992) Properties of Software Measures. Software Quality Journal, 1, pp. 225- 260
[11] Weyuker, E. J. (1988) Evaluating software complexity measure. IEEE Transaction on Software Complexity Measure, 14(9) pp. 1357-1365
[12] Marinescu, R. (2005) Measurement and Quality in Object –oriented design, In Proceedings 21st IEEE International Conference on Software Maintenance, pp. 701-704
[13] Reißing R. (2001) Towards a Model for Object-oriented Design Measurement, Proceedings of International ECOOP Workshop on Quantitative Approaches in Object-oriented Software Engineering, pp. 71-84
[14] Rosenberg L. H. (1995) Software Quality Metrics for OO System environment. Technical report, SATC-TR-1001, NASA
[15] Zuse, H (1996) Foundations of Object-oriented Software Measures. In Proceedings of the 3rd International Symposium on Software Metrics: From Measurement to Empirical Results (METRICS '96) IEEE Computer Society, Washington, DC, USA, pp. 75-84
[16] Misra S. (2010) An Analysis of Weyuker's Properties and Measurement Theory, Proc. Indian National Science Academy, 76(2), pp. 55-66
[17] Zuse, H. (1998) A Framework of Software Measurement, Walter de Gruyter, Berline[18] Morasca S (2001) Software Measurement, Handbook of Software Engineering and Knowledge Engineering, 2001, World Scientific Pub. Co. pp. 239-276
[19] Gursaran, Ray G. (2001) On the Applicability of Weyuker Property Nine to OO Structural Inheritance Complexity Metrics, IEEE Trans. Software Eng., 27(4) pp. 361-364
[20] Sharma N., Joshi P., Joshi R. K. (2006) Applicability of Weyuker's Property 9 to OO Metrics" IEEE Transactions on Software Engineering, 32(3) pp. 209-211
[21] Zhang L., Xie, D. (2002) Comments on 'On the Applicability of Weyuker Property Nine to OO Structural Inheritance Complexity Metrics. IEEE Trans. Software Eng., 28(5) pp. 526-527
[22] Misra S., Kilic, H. (2006) Measurement Theory and validation Criteria for Software Complexity Measure, ACM SIGSOFT Software Engineering Notes, 31(6), pp. 1-3
[23] Poels G., Dedene G. (1997) Comments on Property-based Software Engineering Measurement: Refining the Additivity Properties, IEEE Trans. Softw. Eng. 23(3) pp. 190-195
[24] Misra. S. (2006) Modified Weyuker's Properties, In Proceedings of IEEE ICCI 2006, Bejing, China, pp. 242-247
[25] Misra S., Akman I. (2008) Applicability of Weyuker's Properties on OO Metrics: Some Misunderstandings, Journal of Computer and Information Sciences, 5(1) pp. 17-24
[26] Tolga O. P., Misra S. (2011) Software Measurement Activities in Small and Medium Enterprises: An Empirical Assessment', In press, Acta Polytechnica, Hungarica, 4
[27] Misra S. (2011) An Approach for Empirical Validation Process of Software Complexity Measures, In press, Acta Polytechnica, Hungarica. Issue 4
[28] Zhuo F., Lowther B., Oman P. and Hagemeister Jack., "Constructing and testing software maintainability assesement models", IEEE Computer Society, 1993, pp 61-70.
[29] Binkley A. and Schach S., "Validation of the coupling dependency metrics as a predictor of run time failures and maintainability measures", Proc. 20th International conference of software engineering, pp. 452-455, 1998.
[30] Muthanna S., Kontigiannis K., Ponnambalam K. and Stacey B.,"A Maintainability Model for industrial Software System Using Design Level Metrics", IEEE Computer Society, 2000, pp 248-256.
[31] Rizvi S.W.A. and Khan R.A., "Maintainability Estimation Model for Object-Oriented Software in Design Phase (MEMOOD)",Journal of Computing, Volume 2, Issue 4, April 2010,
[32] Gautam C., kang S.S., "Comparison and Implementation of Compound MEMOOD MODEL and MEMOOD MODEL", International journal of computer science and information technologies, pp 2394-2398, 2011.
[33]. B. F. Abreu: "Design metrics for OO software system", ECOOP 95, Quantitative Methods Workshop, 1995.
[34]. M. Alshayeb and Li.W.,"An empirical validation of object-oriented metrics in two different iterationsoftware processes", IEEE transactionod Software Engineering, Vol-29, no-.11, Nov 2003.

[35] K. Morris, "Metrics for Object-oriented Software Development Environments," Masters Thesis, MIT, 1989.

[36]M. Lorenz, J. Kidd, "OO Software Metrics", Prentice Hall, NJ, (1994).

[37] C. Shyam, Kemerer, F. Chris, "A Metrics Suite for Object- Oriented Design" M.I.T. Sloan School of Management, pp. 53-315, 1993.

[38] C. Shyam and C. F. Kemerer, "Towards a Metrics Suite for OO Design", Proceeding on OO Programming Systems, Languages and Applications Conference (OOPSLA'91), ACM, Vol. 26, Issue 11, Nov 1991, pp. 197-211.

[39] C. Shyam and C. F. Kemerer, "A Metrics Suite for OO Design", IEEE Transactions on Software Engineering, Vol. 20, No. 6, June 1994, pp. 476-493.

[40] W. Li, Sallie, Henry "Metrics for Object-Oriented system", Transactions on Software Engineering, 1995.

[41] B. F. Abreu: "Design metrics for OO software system", ECOOP'95, Quantitative Methods Workshop, 1995.

[42] N. Fenton et al, "Software metrices: a rigorous and practical approach", International Thomson computer press 1996.

[43] V.L.Basili, L. Briand and W. L. Melo, "Avalidation of object-oriented Metrics as Quality Indicators", IEEE Transaction Software Engineering. Vol. 22, No. 10, 1996, pp. 751-761.

[44] J. Bansiya, C. G. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment", IEEE Transactions on Software Engineering, 28, (1), (2002), 4–17.

[45] Li W., "Another Metric Suite for Object-oriented Programming", The Journal of System and Software, Vol. 44, Issue 2, December 1998, pp. 155-162.

[46] Rosenberg Linda, "Software Quality Metrics for OO System Environments", A report of SATC's research on OO metrics.