Metric Based Testability Model for Object Oriented Design (MTMOOD)

R. A. Khan
Department of Information Technology
B. B A. University (A Central University)
Lucknow-UP
khanraees@yahoo.com

K. Mustafa
Department of Computer Science
J. M. I. (A Central University)
N. Delhi
<kmfarooki@yahoo.com>

Abstract

This paper does an extensive review on testability of object oriented software, and put forth some relevant information about class-level testability. Testability has been identified as a key factor to software quality, and emphasis is being drawn to predict class testability early in the software development life cycle. A Metrics Based Model for Object Oriented Design Testability (MTMOOD) has been proposed. The relationship from design properties to testability is weighted in accordance with its anticipated influence and importance. A suit of adequate object-oriented metrics useful in determining testability of a system has been proposed, which may be used to locate parts of design that could be error prone. Identification of changes in theses parts early could significantly improve the quality of the final product and hence decrease the testing effort. The proposed model has been further empirically validated and contextual interpretation has been drawn using industrial software projects.

Keywords: Software Testability, Design, Object Oriented, Metrics, Model, Software Characteristics

1. Introduction

Object orientation has rapidly become accepted as the preferred paradigm in industrial software development environments for large-scale system design. This technology offers support to provide software product with higher quality and lower maintenance costs [11]. Classes in object oriented system provide an excellent structuring mechanism that allows a system to be divided into well designed units which may then be implemented separately. One of the major advantage of having object orientation is its support for software reuse, which may be achieved either through the simple reuse of a class in a library or via inheritance. If we are to capitalize on the potential advantages of object orientation then it is important that the object-oriented approach is adopted and supported throughout the software development life cycle.

Unfortunately, the very advantages of object oriented cited above become potential disadvantage as for as the testing is concerned. Object oriented paradigm has created new challenges to testing, which has to deal with new problems introduced by the powerful object oriented features such as encapsulation, inheritance, polymorphism, and dynamic binding. Especially dealing with instantiations of classes and their collaboration may be very difficult when testing is performed. These new challenges to testing of object-oriented application have been met by extensive research that still produces new results [13]. When software testing begins, an initial survey has obvious advantages over testing blind. Testability suggests the testing intensity, and provides the degree of difficulty which will be incurred during testing of a particular location to detect a fault[1][2]. It is an inevitable fact that testability information is useful strategy complementary to testing. Higher test coverage may be achieved by making a system more testable for the same amount of effort, which as a result increases the confidence to the system [3]. Achieving testability is mostly a matter of separation of concerns, coupling between classes and subsystems, and cohesion [18]. In order to minimize the testing effort, an attempt can be made to predict which class is more testable, by looking at two classes [4].

Much of the research work reveals that maximum efforts have been dedicated with the source code. The determination of testability for an already written code may be too costly because the latter the changes are introduced the more expensive they are [16]. Thus, there appeared to be need for evaluating the testability of software in early stage of development life cycle without the availability of code. Availability of a suitable and adequate measuring tool at the early stage of development enables early prediction of system testability thus reducing the cost of making necessary changes.

In order to provide the significant and improved measurement of object oriented software development product, a concerted effort to find quantifiable way to relate measurable object oriented characteristics to the high level desirable software quality attributes is required [12]. It is evident from literature survey that there is no known comprehensive and complete model or framework for evaluating the testability of designs developed using an object oriented approach based on its internal design property [11]. The model and metrics proposed in this paper address many of the issues raised by various researcher and practitioners. This model has the low-level design metrics well defined in terms of design characteristics. The set of empirically identified and weighted object oriented design properties are used to assess the testability.

Rest of the paper is organized as follows: Section 2 describes briefly software testability at design phase. In section 3, a Metrics Based Testability Model for Object Oriented Design (MTMOOD) has been proposed, design properties have been defined and a brief description of proposed metrics has been included. The effectiveness of this model in predicting design testability has been validated against several real world projects in section 4. It has been concluded that testability predicted by this model shows high correlation with evaluators' assessments.

2. Software Testability at Design Phase

It is an inevitable fact that software must be verified. It is true not only for critical systems where a failure might lead to loss of lives or great economical values, but also for non-critical systems. A system that deviates from the expected behavior is often worse than no system at all. Testability is one of the most important quality indicators since its measurement leads to the prospect of facilitating and improving a test process. The process of software engineering creates a unique problem for testability [8]. Software testability analysis has been an important research direction since 1990s and becomes more pervasive when entering 21st century [9].

http://doi.acm.org/10.1145/1507195.1507204 DOI: 10.1145/1507195.1507204

Testability, comprising of certain characteristics of a software system that makes it easier or harder to test and to analyze the test results, is an important factor to achieve an efficient and effective test process. Designing, verifying and measuring highly testable software becomes an important and challenging task for software developers.

According to IEEE standard, the term of testability refers to the degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met; and the degree to which a requirement is stated in terms that permit the establishment of test criteria and performance of tests to determine whether those criteria have been met. Testability is a software quality characteristic that is of major relevance for test costs and software dependability. Still, testability is not an explicit focus in today's industrial software development projects. An extensive survey of literature reveals that processes, guidelines, and tools related to software testability are missing [14].

Improving software testability is clearly an important objective in order to reduce the number of defects that result from poorly designed software[17]. Testable design is more specific then good design because it is explicitly intended to match a particular test context. One pro-active strategy that organizations can adopt is to design their software products with testability as one of the key design criteria. Aspects of testability like observability and reproducible behavior are not the primary focus of good design and require special treatment. A design is a process that starts from a study of a domain problem leading to some formal documentation. Software design, in some ways, is a strange art [7]. At the first instance it may result in a model of the domain problem by formally capturing and representing the user's requirements and hence paving the way for a conceptual relation. It may serve well as a communication medium between the designer and the user on the one end, and act as a basis for implementation on the other end [5]. No doubt, it is key to the successful development of quality software. It is also the step that will determine the overall structure, nature, and approach of the resulting software. This is evident that the software design is an important stage spanning the whole software lifecycle, not only for software development but also for re-developing legacy systems [6].

Conventional software testing focuses on the generation of tests for existing classes and systems. The technologies of design for test draw attention to building testability into software during design so that the succeeding processes in test generation and implementation can be greatly simplified [15]. There are different ways to improve the testability of a software design. A talk delivered by Y. Wang demonstrates that software testability at class and system levels can be quantitatively modeled and analyzed [15]. An early estimation of software testability, exclusively at design phase, will reduce testing effort and duration and will increase likelihood of test automation and maintainability. A lack of testability, like other design faults, is expensive to repair when detected late during software development. Therefore, testability should be addressed already during reviews of early development artifacts. Design-for-testability is a very important issue in software engineering. It becomes crucial in the case of object oriented software because of the fact that control flows are generally not hierarchical but are diffuse and distributed over the whole architecture [10].

3. Model Development

Dromey's generic quality model[11][19] has been considered as a basis to develop the Metrics Based Testability Model for Object Oriented Design (MTMOOD) shown in Figure 1, which involves the following steps:

1. Identification of product properties (Object Oriented Software) that influences testability of software.
2. Identification of Object Oriented Design Metrics
3. A means of linking of them

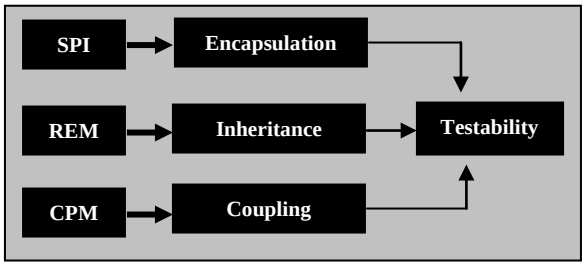

Figure 1: Metrics Based Model for Object

3.1 Identification of Product Property

It is the abstraction and real world modeling concepts that differentiates object oriented software to structural software. Three fundamental characteristics required for an object oriented approach: Encapsulation, Polymorphism and Inheritance. Polymorphism and Inheritance are two aspects unique to object oriented approach, while encapsulation is not [20].

Information hiding is a way of designing such that only subsets of the module's properties, its public interfaces, are known to users of the module. It gives rise to encapsulation in object oriented language. Encapsulation means that all that is seen of an object is its necessary interface, namely the operations we can perform on the object. Information hiding is a theoretical technique that indisputably proven its value in practice. Large programs that use information hiding have been found to be easier to modify by a factor of 4 than programs that don't adhere to this technique [21][22].

Inheritance is a form of reuse that allows a programmer to define objects incrementally by reusing previously defined objects as the basis for new objects. Inheritance decreases complexity by reducing the number of operations and operators, but this abstraction of objects can made maintenance and design difficult [21][23].

Design Property	Definition
Encapsulation	Defined as a kind of abstraction that enforces a clean separation between the external interface of an object and its internal implementation
Inheritance	A measure of the 'is-a' relationship between classes.
Coupling	Defined as the interdependency of an object on other objects in a design.
Cohesion	Defines as the internal consistency within the parts of design.

Table 1: Design Property Definitions

Polymorphism is an important concept that allows building a flexible system. This concept of polymorphism allows developer to specify what shall occur and how shall occur. Polymorphism means having the ability to take several forms. For object oriented system, polymorphism allows the implementation of a given operations to

http://doi.acm.org/10.1145/1507195.1507204 DOI: 10.1145/1507195.1507204

be dependent on the object that contains the operations; and operations can be implemented in different ways in different classes. There is no doubt polymorphism assists programmer to reduce complexity [24] and improve on many desirable attributes including reusability [21][22]. Table 1 summarizes the definitions of the design properties included in MTMOOD model.

3.2 Identification of Object Oriented Design Metrics

Several research works in the object oriented metrics arena were produced in recent years [25-34]. However, widespread adaptation of object oriented metrics in numerous application domains should only take place if the metrics may be shown to be theoretically valid, in the sense that they accurately measure the attributes of software for which they were designed to measure and have also been validated empirically [35]. But there is a general agreement among experts that metrics have not been even validated theoretically. Most of the metric is accepted by practitioners on 'heavy usages and popularity' and by academic experts on empirical (post development) validation. In such a scenario many of the available metrics may not be used properly and have been discarded. Reasons may be that these could not find a place among practitioners or not found to be valid by experts on empirical findings. Now the question is 'Why such metrics which could not become acceptable were developed?'- Leading to all efforts in development going vain.

Since the traditional software metrics aims at the procedure-oriented software development and it cannot fulfill the requirement of the object-oriented software, a set of new software metrics adapted to the characteristics of object technology is highly desirable. Accordingly object-oriented metrics tools become a popular category of software measurement tool [11]. This led to the definition of four new metrics, Encapsulation Metrics (ENM), Reuse Metrics (REM), Coupling Metrics (CPM) and Cohesion Metrics (CHM), which could be calculated from design information only. A suite of object-oriented metrics, covering all the design attributes, has been listed and described in table 2.

Metrics	Name	Description
ENM	Encapsulation Metric	This metric count of all the methods defined in a class
REM	Reuse Metric	This metric count of the number of class hierarchies in the design.
CPM	Coupling Metric	This metric count of the different number of classes that a class is directly related to.
COM	Cohesion Metric	This metric computes the relatedness among methods of a class based upon the parameter list of the methods [computed as LCOM, 1993 Li and Henry version]

Table 2: Proposed Metrics

3.4 Building Linkages

In order to establish a relationship between design constructs and testability, the influence of design constructs quality attributes are being examined with respect to SATC's attributes [20]. It was observed that each design constructs affects certain quality attributes [20-22][24]. The extensive review of object oriented development books and publications [19][22][36-38] indicate that the encapsulation is viewed to promote efficiency and complexity. Inheritance design property has a significant influence on the efficiency, complexity, reusability and testability/maintainability. While low coupling is considered good for understandability, complexity, reusability and testability/maintainability. Higher measures of coupling are viewed to adversely influence these quality attributes. Cohesion is viewed to have a significant effect on a design's understandability and reusability.

Based upon the design property to testability relationship, the relative significance of individual design properties that influence software testability is weighted proportionally. A multiple linear regression (MR) has been used to get the coefficients. The multiple linear regression establishes a relationship between dependent variables and multiple independent variables. The MR equation takes the form:

$$y= \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots\ldots\ldots\ldots \beta_k x_k \quad (1)$$

Where

- The xs are the independent (predictor or regressor) variables.
- Y is the dependent (response) variable.
- β (Regression coefficient) is the average amount of dependent increase when the independents are held constant.

Testability of a class depend upon one or more number of design attributes , component-wise effect may be speculated and respective component weightage (CW) may be fixed using regression equation. Thereby, the CWs of individual design attributes have been calculated in terms of regression coefficient β. Three medium size C++ projects were used to fit the regression line by acquiring real data from commercial projects of Software Company based in Delhi, India. Name of the Projects and actual source data is being concealed, as per wishes of company's management. We have assured authenticity of source data, to the best possible extent. These projects include the number of classes (10-20). The projects were independently completed over a period four months. Using these data, the CW coefficient calculated for encapsulation, inheritance and coupling to show the complexity relationship with design properties (encapsulation, inheritance and coupling). Equation 2 summarizes the computational formula for Testability with the component weightage.

Testability= -0.08 * Encapsulation + 1.12 * Inheritance + 0.97 * Coupling (2)

Class Name	ENM	REM	COP	COM
Company	3	0	2	3
Employee	3	0	3	1
Department	2	0	2	1
Manufacturing	3	1	1	1
Development	4	1	1	2
Product	2	0	1	2
Sample	3	1	0	2
Commercial	2	1	0	1
Project	3	0	2	1
Salaried	3	1	0	1
Contractual	3	1	0	1

Table 3.0: Metrics Calculation

4. Calculation of Metrics Suit on Class Diagram

The following four metrics proposed in this paper may be easily calculated from the class diagram. Figure 2.0 shows a simple class diagram in UML notation. Encapsulation Metrics (ENM) can be

http://doi.acm.org/10.1145/1507195.1507204 DOI: 10.1145/1507195.1507204

calculated as soon as class operations are identified. Reuse Metrics (REM) can be calculated by counting of the number of class hierarchies in the design shown in figure 1.0. Here, to calculate the proposed Coupling Metrics (CPM) and Cohesion Metrics (COM), the source code of methods are not required.
The detail calculated values of all these four metric suit is shown in table 3.0 for the figure2.0.

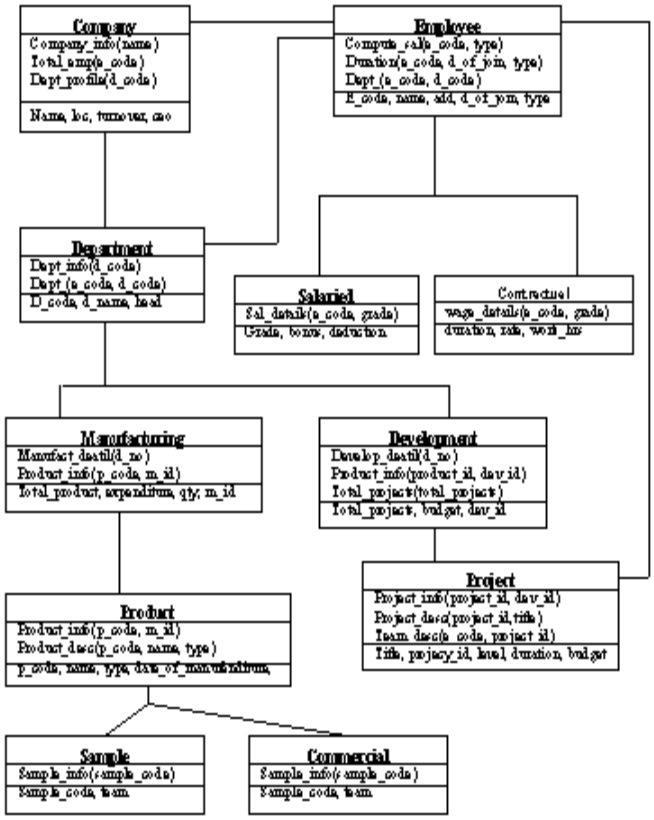

Figure 2.0: Class Diagram

5. An Experimental Validation

This section assesses how well the proposed metrics suit is able to predict the 'Class Testability' of object-oriented software. The internal characteristic of a design varies significantly with the objective and domain of the design. These characteristics influence the testability and other quality attributes and, therefore, the overall quality. So validation of the predictability of MTMOOD requires the set of object oriented designs with same set of requirements for the evaluation and validation, as a limitation of this model.
A medium size C++ project was used in the validation of proposed model. This project was selected because several designs of the project that addresses the same set of requirements were easily available. The size of these projects, reflected by the number of classes (10-20), made them an ideal candidate. The projects were independently completed over a period of months. A suite of six projects was developed by different individuals for the study. The six projects of the validation suite were also evaluated using the proposed metric suite. The metrics values calculated are shown in table 4. The values for testability are computed using the equations 2.0 and shown in table 5. These testability scores are used to assign the Testability Benchmark (BC) to all the six projects.

Metrics	Projects					
	1	2	3	4	5	6
Encapsulation	3.3.6	2.88	3.91	4.10	4.22	2.86
Inheritance	.36	.33	.58	.42	.55	.26
Coupling	1.81	1.88	2.08	2.52	2.72	2.26
Cohesion	1.27	1.33	1.58	1.89	1.33	1.26

Table 4: Design Property Metrics Value

	Projects					
	1	2	3	4	5	6
Testability	20.92	17.77	28.33	49.28	52.65	34.02
BC	2	1	3	5	6	4

Table 5: Design Testability Indices

Project No.	BC	Evaluators Ranking									
		1	2	3	4	5	6	7	8	9	10
1	2	3	3	3	3	2	2	1	3	1	3
2	1	2	2	1	2	2	1	2	2	2	3
3	3	4	4	3	2	2	4	3	3	2	4
4	5	5	4	4	6	6	5	4	6	5	5
5	6	5	5	5	6	5	6	6	6	6	6
6	4	5	5	4	3	4	4	4	4	4	4

Table 6 Evaluators Model and Testability Ranking

A group of ten independent evaluators was assigned to study the quality of the six projects in the validation suite. All the evaluators had 8 to 12 years of experience in commercial software development, had knowledge of the object oriented paradigm, and had developed software using C++. The study was done over a period of month. All the participants analyzed each project's design and assigned the scores on an ordinal scale to the Testability by using their own traditional tool. These scores for all attributes were summed separately to give the evaluator's Testability Benchmark (BC) for the project. The projects were ranked 1 through 6 based on decreasing testability scores by each evaluator as shown in Table 6.

5.1 Statistical Analysis

The Spearman's rank correlation coefficient, r_s was used to test the significance of correlation between design time metric based Testability assessment and the evaluator's implementation based assessment of the project. It provides a nonparametric significance test that works well with ranked data without precise proportional scaling and can be used to detect relationships other than linear one. For the level of significance of α=95%, the threshold value calculated for n=6 projects was±. 817.
For two independent evaluations X_1 and X_2 of n items that are to be correlated, the values of X_1 and X_2 are ranked from 1 to 'n' according to their relative size within the evaluations. For each X_1 and X_2 pair in relative rank, differences in ranks'd' is computed. The sum of all d^2 s denoted by $\sum d^2$ is used to compute r_s using formula:

$$r_s = 1 - \frac{6\sum d^2}{n(n^2-1)} \qquad -1.0 \le r_s \le +1.0 \qquad (3)$$

r_s= Coefficient of Rank Correlation
n= No. of Paired Observation

∑= Notations meaning 'the sum'
d= difference between the ranks for each pair of observations

5.2 Contextual Interpretation

The correlation values between MTMOOD determined Testability rankings of the six projects and ten evaluators assessments of overall quality of these projects based on design and implementation shown in table 6, has been listed in table 7. Pairs of MTMOOD – Evaluator with Correlation values r_s above (±. 817) are checked in table 6. It is evident from the correlation values shown in table 6 that the Testability indicators evaluated using proposed MTMOOD are highly correlated with the measurements done by human evaluators.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
$\sum d^2$	5	6	3	5	4	1	3	3	3	3
r_s	.857	.826	.914	.857	.886	.971	.914	.914	.914	.914
r_s> **.817**	√	√	√	√	√	√	√	√	√	√

Table 7: Metric Suits and Evaluators Ranking Correlation

6.0 Conclusion

The proposed model for the assessment of testability in object-oriented design has been validated using structural and functional information from object oriented software. The models' ability to estimate overall testability from design information has also been demonstrated using several functionally equivalent projects where the overall testability estimate computed by model had statistically significant correlation with the assessment of overall project characteristics determined by independent evaluators. The proposed model is more practical in nature having quantitative data on testability is of immediate use in the software development process. The software developer can use such data to plan and monitor testing activities. The tester can use testability information to determine on what module to focus during testing. And finally, the software developer can use testability metrics to review code, trying to find refactorings that would improve the testability of the code also.

References

[1] Jeffrey M. Voas & Keith W. Miller, Software Testability: The New Verification, IEEE Software, Volume 12, Issue 3, pp.17 – 28, May 1995.

[2] Richard G. Hamlet, Probable Correctness Theory, Information processing Letters, 25(1):17-25, April, 1987.

[3] Birgitta Lindstrom, Methods for Increasing Software Testability, as a dissertation towards the degree of M. Sc. Thesis submitted to University of Skovde, Department of Computer Science, (HS-IDA-MD-00-017) December 2000.

[4] Artem, Testability, 2007.
Available at: http://agilesoftwaredevelopment.com/tags/testability

[5] Symons D., "Software Sizing and Estimating MK2 FPA (Function Point Analysis)", Wiley, 1991.

[6] Khan R A, Quality Estimation of Object Oriented Code-A Design Metrics Perspective, Ph.D. thesis submitted to the Department of Computer Science, Jamia Millia Islamia New Delhi, 2005.

[7] John Hunt, Designing Software for Testability, 29 Oct 2007.
Available at:
http://www.regdeveloper.co.uk/2007/10/29/design_for_testability/page2.html

[8] Troy Lamoreaux, Mark Ofori-Kyei Mark, Pinone, A Process for Improving Software Testability, Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM'04), IEEE, 2004.

[9] Liang Zhao, A new approach for software testability analysis, International Conference on Software Engineering**,** Proceeding of the 28th international conference on Software engineering, Shanghai, China pp. 985–988, 2006

[10] Benoit Baudry, Yves Le Traon, and Gerson Sunyé, Testability Analysis of a UML Class Diagram, Proceedings of the Eighth IEEE Symposium on Software Metrics (METRICS.02), IEEE 2002.

[11] Khan R A & Mustafa K, A Model for Object Oriented Design Quality Assessment Proceedings, Integrated Design And Process Technology Symposium, , Kusadasi, Izmir, Turkey, June 28-July 2, 2004.

[12] J. Bansiya, C. G. davis, A Hierarchical Model for Object Oriented Design Quality Assessment, IEEE Transaction on Software Engineering, Vol. 28, No. 1, pp. 4-17, Jan 2002.

[13] Raine kauppinen, "Testing framework- based software product lines", Master thesis, University of Helsinki, Department of Computer Science, April 17, 2003.

[14] Stefan Jungmayr, Design for Testability, CONQUEST 2002 – 57.

[15] Y. Wang, Design for Test and Software Testability, University of Calgary, 2003.
Available at: http://www.ucalgary.ca/~ageras/wshop/abstracts/2003/design-for-testability.htm

[16] Llona Nbluemke, "Object oriented metrics useful in the prediction of class testing complexity", IEEE 2001.

[17] Design for Testability An e-newsletter published by Software Quality Consulting, Inc. Vol. 3 No. 3, March 2006**.**
Available at: http://www.swqual.com/newsletter/vol3/no3/vol3no3.html

[18] Jeremy D. Miller, Designing for Testability, The Shade Tree Developer, Jun 29 2007.
Available at:
http://codebetter.com/blogs/jeremy.miller/archive/2007/06/29/designing-for-testability.aspx

[19] G. R. Dromey, A Model for Software Product Quality, IEEE Transaction on Software Engineering, vol. 21, no.2, pp.146-162, Feb.1995.

[20] R. A. Khan & K. Mustafa, A review of SATC research on OO Metrics, proceedings, National Conference on Software Engineering Principles and Practices, SEPP-04, March 5-6, 2004.

[21] R. A. Khan & K. Mustafa, High Level Design Quality Assessment of Object Oriented Codes, accepted for publication in the proceedings in 2nd International Workshop on Verification and Validation of Enterprise Information System VVEIS Porto, Portugal, April 13, 2004.

[22] R. A. Khan, K. Mustafa, & S. Yadava, Quality Assessment of Object Oriented Code in Design Phase, Proceedings, QAI 4th Annual International Software Testing Conference, Pune, India Feb. 20-21, 2004.

[23] http://colaboration.csc.ncsu.edu/CSC325_Fall 2002 /lectures/ Object_oriented_Metrics

[24] Linda Rosenberg, Software Quality Metrics for Object Oriented System Environments, A report of SATC's research on OO metrics http://ourworld.compuserve.com/homepages/qualazur/$swmesu2.htm

[25] Bansiya Jagdish & Devis Carl, "Automated Metrics and Object Oriented Development", Dr. Dobb's Journal December 1997.
http://www.ddj.com/documents/s=934/ddj9712d/9712d.htm

[26] Dumke, Reiner R., "A Measurements framework for Object- Oriented Software Development", submitted to Annals of Software Engineering , Vol.1, 1995.

[27] Henderson Sellers, B., "Identifying internal and external characteristics of classes likely to be useful as structural complexity metrics", Proceedings of 1994 intern. Conf. On Object Oriented Information Systems OOIS'94, London, December 1995, Springer-Verlag, pp. 227-230, London, 1995.

[28] Campanai M. and Nesi P., "Supporting Object Oriented Design with Metrics", Proceedings of TOOLS Europe' 94, France, 1994.

[29] Cant S. N., B. Henderson Sellers, and Jeffery D. R., "Application of cognitive complexity metrics to object- oriented program", Journal of Object- Oriented Programming, pp. 52-63, July- August 1994.

[30] Hopkins, Trevor P., "Complexity Metrics for Quality Assessment of Object-Oriented Design", SQM'94, Edinburgh, July 1994, Proceedings published as Software Quality Management II, vol. 2: Building Quality into Software, pp. 467-481, Computational Mechanics Press, 1994.

[31] Abreu, Brito F. and Carapuca, Rogerio, "Candidate Metrics for Object-Oriented Software within a Taxonomy Framework", Proceedings of AQUIS'93 (Achieving Quality In Software), Venice, Italy, October 1993: selected for reprint in the Journals of Systems and Software, Vol. 23 (I), pp. 87-96, July 1994.

[32] Chidamber, S. R. and C. F. Kemerer, “Towards a metrics suite for Object-Oriented Design”, Proceeding: OOPSLA ’91, July 1991, pp. 197-211.
[33] Li., W. and Henry, “Maintenance metrics for the object-oriented paradigm”, Proceeding Of The First International Software Metrics Symposium, May 1993, pp.-52-60.
[34] Hitz, M. and Montazeria. B., “Chidamber and Kemerer’s Metrics Suite: a measurement theory perspective”, IEEE Transactions on software Engineering, 22(4), April 1996, pp. 267-271.
[35] Schneidewind N. F., “Methodology for validating software metrics”, IEEE Software Engineering, vol. 18, no. 5, pp. 410-422, 1992.
[36] B. Kitchenham, & Pfleeger, S. L., Software Quality: The Elusive Target, IEEE Software, 1996, 13(1): 12-21.
[37] S.R. Chidamber, C.F. Kemerer, A metrics Suites for Object Oriented Design, IEEE Transaction on Software Engineering, Vol.20, No. 6, pp. 476-493, Jan 1994.
[38] B. Basili, L. Briand, and W. L. Melo, A validation of Object Oriented Metrics as Quality Indicators, IEEE Trans. Softwaree Engineering, Vol.22, No. 10 pp. 751-761, Oct-1996.

http://doi.acm.org/10.1145/1507195.1507204 DOI: 10.1145/1507195.1507204