Journal of Computer Science 1 (2): 276-282, 2005
ISSN 1549-3636
© Science Publications, 2005

A Fuzzy Approach for Integrated Measure of Object-Oriented Software Testability

[1]Vandana Gupta, [2]K.K. Aggarwal, [3]Yogesh Singh
[1]Department of Computer Science, Kalindi College, Delhi University
Delhi 110008, Research Student GGS Indraprastha University, Delhi 110006, India
[2]GGS Indraprastha University, Delhi 110006, India
[3]School of Information Technology, GGS Indraprastha University, Delhi 110006, India

Abstract: For large software systems, testing phase seems to have profound effect on the overall acceptability and quality of the final product. The success of this activity can be judged by measuring the testability of the software. A good measure for testability can better manage the testing effort and time. Different Object Oriented Metrics are used in measurement of object-oriented testability but none of them is alone sufficient to give an overall reflection of software testability. Thus an integrated measure considering the effect of all these measures is required to well define the testability. The paper combines OO software metric values into a single overall value (called Testability Index) that can be used to calculate the testability of a class. The approach uses fuzzy techniques and concepts (fuzzification of crisp metric values, inference and aggregation, defuzzification of fuzzy output). We include empirical data of testing time of 25 different Java classes, which proves that individual metric values are not sufficient to arrive at the testability of a class and validates the testability index as a good integrated measure for arriving at the testability of the class.

Key words: Testing Effort, Testability Metric, Object Oriented Testability

INTRODUCTION

When dealing with software testability, some questions naturally come to one's mind. What is it that makes code hard to test? Why is one class easier to test than another? How can we tell that we are writing a class that will be hard to test? What contributes to a class' testability? There are many definitions of testability. The most common is the ease of performing testing [1]. This definition has its roots in hardware testing and is usually defined in terms of observability and controllability. Binder defines these two facets of testability succinctly [2]: "To test a component, you must be able to control its input (and internal state) and observe its output. If you cannot control the input, you cannot be sure what has caused a given output. If you cannot observe the output of a component under test, you cannot be sure how a given input has been processed." Testability holds a prominent place as part of the maintainability characteristics of the ISO 9126 quality model [3]; this also increases our understanding of Software in general. Having quantitative data on testability is of immediate use in the software development process. The software manager can use such data to plan and monitor testing activities. The tester can use testability information to determine on what code to focus during testing. And finally, the software developer can use testability metrics to review his code, trying to find refactorings that would improve the testability of the code.

Several measures for OO designs have been proposed (Li and Henry [23]; Chidamber and Kemerer [11]; Brito e Abreu [24]; Briand *et al.* [25]; Bieman and Kang [26]) and validated (Basili *et al.* [27]; Brito e Abreu and Melo [28]; Briand *et al.*, [29,30,33,34]; Harrison *et al.* [31, 32];). For historical reasons, the "CK metrics suite" proposed by Chidamber and Kemerer [11] are the most frequently referenced OO-design measures. Chidamber and Kemerer [11] have proposed the following 6 metrics - Weighted Methods per class (WMC), Response for a class (RFC), Lack of Cohesion of Methods (LCOM), Coupling Between Objects (CBO), Depth Inheritance Tree (DIT) and Number of Children (NOC), which measures the different software quality attributes like efficiency, complexity, understandability, reusability, maintainability and testability. Our approach is to evaluate a set of object-oriented metrics with respect to their capabilities to predict the effort needed for testing. We selected the subset of metrics consisting of WMC, RFC, CBO and DIT for measuring the testability of the software.

Factors Affecting Testability: Chidamber and Kemerer [11] have suggested that WMC, RFC, CBO and DIT metrics have bearing on test effort. The evaluation of metrics that are thought to have a bearing on the testing effort allows us, on the one hand to gain insight into the factors of testability [12] and to obtain re-fined metrics on the other. All of the above measurements assess different properties and

Table 1: Summary of Metrics Used in Our Study

Metric Acronym	Definition
Average Cyclomatic Complexity (ACC)	Cyclomatic complexity (McCabe) is used to evaluate the complexity of an algorithm in a method. It is a count of the number of test cases that are needed to test the method comprehensively. A method with a low cyclomatic complexity is generally better. This may imply decreased testing and increased understandability or those decisions are deferred through message passing, not that the method is not complex. [5-7]. For ease of calculation the Average Cyclomatic Complexity is used.
Depth of Inheritance Tree (DIT)	The depth of a class within the inheritance hierarchy is the maximum number of steps from the class node to the root of the tree and is measured by the number of ancestor classes. The deeper a class is within the hierarchy, the greater the number of methods it is likely to inherit making it more complex to predict its behavior [6-8].
Response for a Class (RFC)	The RFC is the count of the set of all methods that can be invoked in response to a message to an object of the class or by some method in the class. This metric looks at the combination of the complexity of a class through the number of methods and the amount of communication with other classes. If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes complicated since it requires a greater level of understanding on the part of the tester. A worst-case value for possible responses will assist in the appropriate allocation of testing time [6-9].
Coupling Between Object Classes (CBO)	Coupling Between Object Classes (CBO) is a count of the number of other classes to which a class is coupled. It is measured by counting the number of distinct non-inheritance related class hierarchies on which a class depends. Strong coupling complicates a system since a class is harder to understand change or correct by itself if it is interrelated with other classes. [5-10]

characteristics of software testability. None of these metrics can individually arrive at the testability of an Object Oriented class. In this paper we propose a metric for Object Oriented software called testability Index. Testability Index (TI), as an integrated measure of many characteristics of Object oriented software e.g. polymorphism, inheritance, encapsulation and abstraction. The WMC, RFC, CBO and DIT are integrated with the help of a fuzzy model, which can combine their individual contribution into a unified metric of object-oriented testability.

In this study, our approach is to select those OO metrics that affect testability. They are summarized in Table 1.

These metrics have been used in the evaluation of many NASA projects and empirically supported guidelines have been developed for their interpretation [35]. The threshold values for the individual metrics needs to be derived by discussing with project managers and programmers and studying the distributions of the metrics collected over a period of time.

Proposed Model: All of the above measurements assess different properties and characteristics of software testability. All these four parameters can be measured easily using some automated tools, but an overall indicator of software testability is desirable, which considers all four aspects and their relative impact on the testability of the software. So an integrated approach for measuring software testability should be used. All of these four measures are quite subjective in nature and thus some metric is needed, which not only integrates these four factors, but also is capable of handling their relative impact on testability. As a fuzzy model is the best choice for managing ambiguous, doubtful, contradicting and divergent opinions [4], a fuzzy model for an integrated software testability measure (Testability Index) is proposed here that takes into account the effect of ACC, DIT, RFC and CBO. Fuzzy logic is also another extension realized in Boolean logic that may be considered a generalization of multi-valued logic. By modeling the uncertainties of natural language through concepts of partial truth—truth-values falling somewhere between completely true and completely false– fuzzy logic deals with such values through fuzzy sets in the interval [0,1]. These characteristics allow fuzzy logic to manipulate real-world objects that possess imprecise limits. Utilizing fuzzy predicates (old, new, high etc.), fuzzy quantifiers (many, few, almost all etc.), fuzzy truth-values (completely true, more or less true) and generalizing the meaning of connectors and logical operators, fuzzy logic is seen as a means of approximate reasoning. A block diagram for the fuzzy model is shown in Fig.1.

The fuzzy model consists of 4 modules. The fuzzification module is the first step in working of any fuzzy model, which transforms the crisp input(s) into fuzzy values. In the next stage, these values are processed in fuzzy domain by inference engine based, on the knowledge base (rule base or production rules)

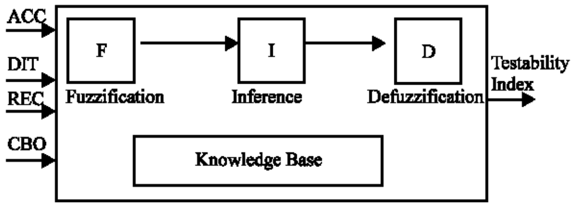

Fig. 1: A Fuzzy Model

supplied by domain expert(s). During this stage, the fuzzy operators are applied, the implication process is followed and then all outputs are aggregated. Finally the processed output is transformed from fuzzy domain to crisp domain by defuzzification module.

Working of Fuzzy Model: First of all crisp values of inputs are taken and then the degree to which the inputs belong to each of the appropriate fuzzy set is determined. Based on these fuzzy inputs, some rules get fired. For selection of rules all the input states are considered simultaneously i.e. these inputs are connected via AND operator. MIN fuzzy operator is applied to find the degree of membership of firing [17-19]. Out of the commonly used inference mechanisms–Mamdani style and Sugeono style, Mamdani style has been used in this model. MAX aggregation is used to integrate the effects of all rules fired. Centroid technique of defuzzification is used to get a crisp value of testability index on a scale of (0, 10). Lower value of testability index reflects less testing effort, while higher value indicates more testing effort.

Membership Functions for Input Parameters: In order to fuzzify the inputs the following membership functions for the ACC, DIT, RFC and CBO are chosen. [13-15]. Input variable ACC is divided into three states (linguistic variables) i.e. Low, Medium and High as shown in Fig. 2.

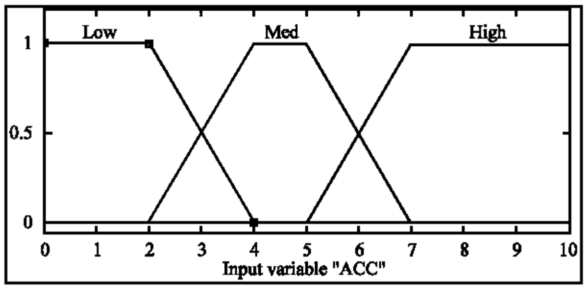

Fig. 2: Membership Function for ACC Metric

The base variable RFC has been divided into three states i.e. PR (Preferred), AC (Acceptable) and NA (Not Acceptable) as shown in Fig. 3.

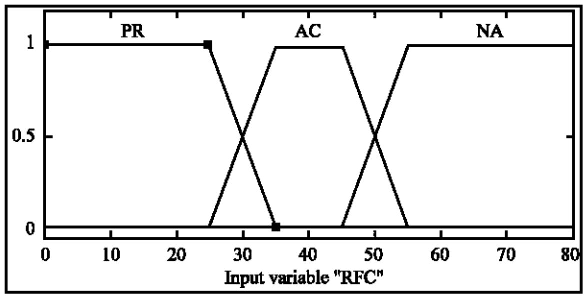

Fig. 3: Membership Function for RFC Metric

Input variable CBO is divided into three states (linguistic variables) i.e. PR (Preferred), AC (Acceptable) and NA (Not Acceptable) as shown in Fig. 4.

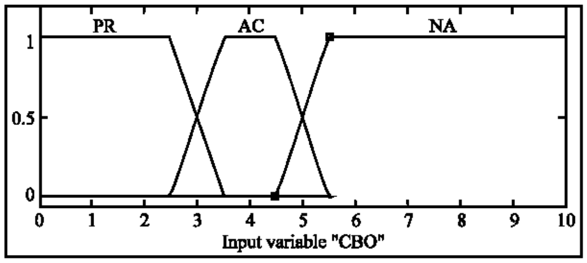

Fig. 4: Membership Function For CBO Metric

Input variable DIT is divided into three states (linguistic variables) i.e. PR (Preferred), AC (Acceptable) and NA (Not Acceptable) as shown in Fig. 5.

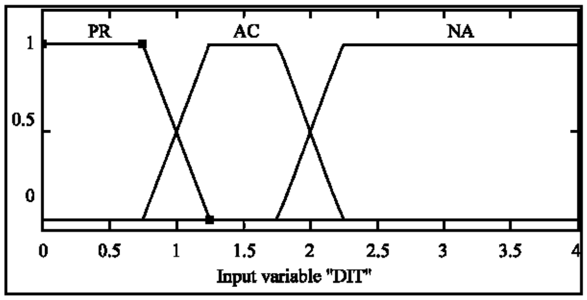

Fig. 5: Membership Function for DIT Metric

Output Variable/Parameter: Having defined the input metrics and threshold values we need interpretation guidelines to assist in identifying those areas of code deemed to be more testable. As suggested by Linda *et al* [35] a single metric should never be used alone to evaluate code risks, it takes at least two or three to give a clear indication of potential problems. For a class that has greater than two metrics that deviate the recommended limits the testability index is high. If the value of DIT, CBO & RFC is below NA range and the value for ACC is below the high limit then these metrics are within the recommended limits. The output variable testability index is defined to have 3-membership functions- LOW, MED and HIGH as shown in Fig. 6. The same are derived based on the rules defined in Table 2.

Table 2: Term Set for Output Variable Testability Index

# OO Metrics Deviating from Threshold Values	Testability Index
<= 1	Low (0-3)
2	Med (3-6)
> 2	High (6-10)

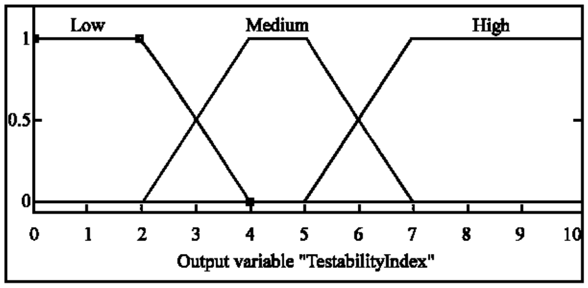

Fig. 6: Output Variable Testability Index Aggregation

In order to measure the software testability Index, the four input metrics are integrated with the help of fuzzy model, which consists of 81 rules.

Rule Base for the Proposed Model: In the proposed fuzzy model we are considering four inputs each consisting of three terms therefore our rule base consists of 81 rules after considering all the possible combinations of inputs. Suppose if a fuzzy model is having n inputs each consisting of m terms then the possible number of rules, say R, for this model can be calculated by considering the Cartesian product of all the input states.

$R = m*m*m*\ldots$upto n times
Or
$R = m^n$

For our model $R = 3^4 = 3*3*3*3 = 81$.
Our rule base can be represented in a tabular form consisting of 81 rows as below:

Table 3: Rule Base of the Fuzzy Model

Rule#	CBO	ACC	RFC	DIT	Testability Index
1	PR	LOW	PR	PR	LOW
2	AC	MED	PR	PR	LOW
3	NA	PR	NA	PR	MED
:					
:					
81	NA	NA	NA	NA	HIGH

Sample Output Computation for The Model: Suppose we have the following crisp inputs to the model: ACC = 2, DIT = 3, RFC = 50 and CBO = 1

These inputs are fed to the fuzzification module and after the fuzzification of the given values we find that ACC = 2 belongs to fuzzy set preferred (PR) with membership grade 1, DIT = 3 belongs to the fuzzy set not acceptable (NA) with membership grade 1, RFC = 50 belongs to fuzzy set acceptable (AC) with membership grade of 0.5 and to fuzzy set Not Acceptable (NA) with membership grade of 0.5 and CBO = 1 belongs to fuzzy set preferred (PR) with the possibility of a degree of 1. With these input values we find that following rules get fired:

Table 4: Testability Index Calculation for the Given Input

ACC (2)	RFC (50)	CBO (1)	DIT (3)	Testability Index	Membership Grade of Testability
PR	AC	PR	NA	LOW	Min (1,0.5,1,1) = 0.5
PR	NA	PR	NA	MED	Min (1,0.5,1,1) = 0.5

First rule gives the testability value Low to an extent of 0.5 and second rule gives testability index Med to an extent of 0.5. This is shown below in Fig. 7.

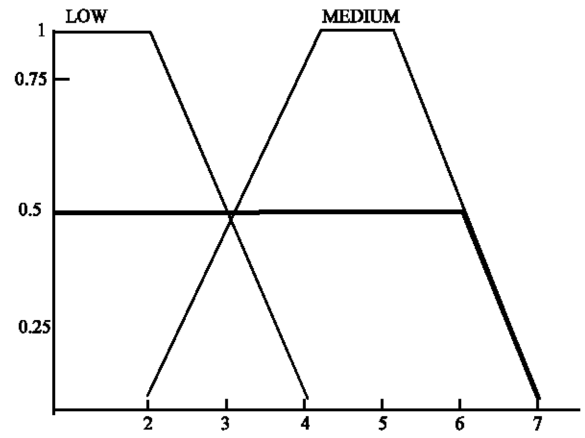

Fig. 7: Output Variable Testability Aggregation

Defuzzification: After getting the fuzzified outputs as shown above, we defuzzify them to get the crisp value of the output variable testability index [20, 21]. For this we are finding the Center Of Gravity (COG) of the above fuzzy output.

COG = x dx / dx where = degree of membership

$$\mathrm{COG} = \frac{{}_{0}^{6}\ 0.5\ xdx + {}_{6}^{7}\ yxdx}{{}_{0}^{6}\ 0.5\ dx + {}_{6}^{7}\ ydx}$$

$$\mathrm{COG} = \frac{\int_{0}^{6} 0.5\,x\,dx + \int_{6}^{7} (mx+c)\,x\,dx}{\int_{0}^{6} 0.5\,dx + \int_{6}^{7} (mx+c)\,dx}$$

$$\mathrm{COG} = \frac{0.5\,[x^2/2]_{0}^{6} + \int_{6}^{7} (7-x)\,x\,dx}{0.5\,[x]_{0}^{6} + \int_{6}^{7} (7-x)\,dx}$$

$$\mathrm{COG} = 2.7$$

The effect of these rules was observed with the help of MATLAB- Fuzzy Tool Box. The value of testability index for above-mentioned input comes out to be 2.85, which is very close to our calculated value of 2.7 above [22]. The integrated approach gives a true picture of the software testability. Crisp value of testability index can be helpful to software managers in judging the testability effort. Lower value of testability index indicates higher testing effort.

RESULTS

In order to validate the model, we considered 25 java classes from different J2EE based projects. The projects, which had the similar complexity in terms of business domain, were chosen. The projects used the similar SDLC and the same technical environment was used for developing them. The values of all four input metrics were computed for each of these 25 classes and listed in Table 5. The values of metrics for these classes are arrived at with the help of JHawk demo version. The output value of testability index was also calculated using proposed fuzzy model and the corresponding values for each of the classes are also listed on the table below. In the last column of this Table 5 the average unit testing time taken for each of these classes is mentioned.

Statistical Analysis: The average unit testing time of these 25 classes have been plotted against each of the four input metrics i.e. ACC, DIT, CBO and RFC in Fig. 8, 9, 10 and 11, respectively.

It can be easily observed that there is hardly any correlation existing between average unit testing time and any of the four input metrics. The correlation of testing time with ACC comes out to be approximately 0.4, while it is .012, 0.6, 0.6, for DIT, CBO and RFC respectively. These values of correlation clearly depict that none of these metrics individually is sufficient to predict the testability. On the other hand a plot of average unit testing time with computed values of testability index is drawn in Fig. 12, which clearly shows a strong correlation between the two curves.

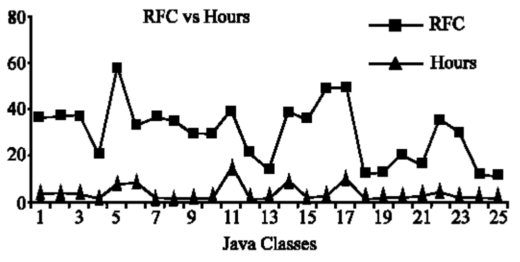

Fig. 8: Plot of RFC Versus Unit Testing Time

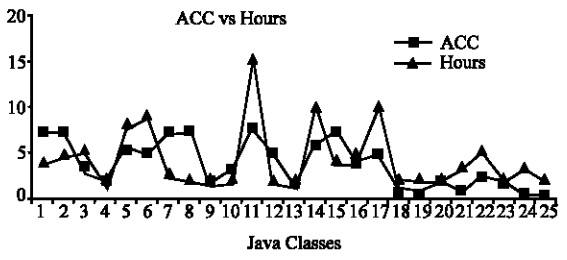

Fig. 9: Plot of ACC Versus Unit Testing Time

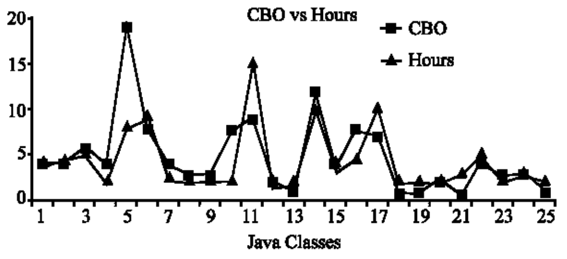

Fig. 10: Plot of CBO Versus Unit Testing Time

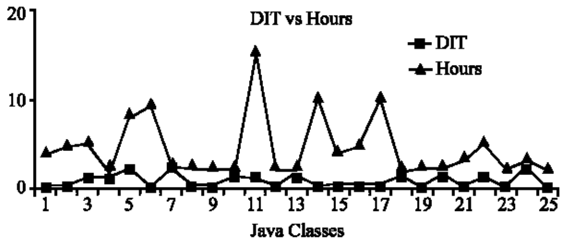

Fig. 11: Plot of DIT Versus Unit Testing Time

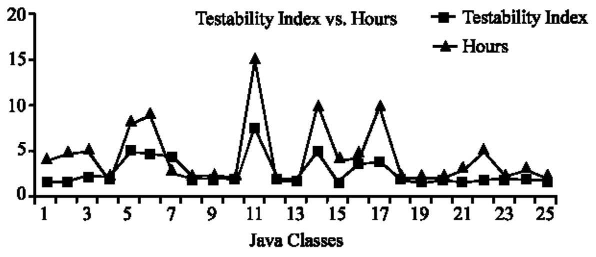

Fig. 12: Plot of Testability Index Versus Unit Testing Time

Table 5: Values of Testing Time and Input and Output Metrics

Class Name	ACC	RFC	CBO	DIT	Testability Index	Hours
AgingCashSchIIReportDAO	7.5	37	4	0	1.53	4
AuditCashSchIIReportDAO	7.5	38	4	0	1.53	4.5
BusinessMasterDAO	3.4	38	6	1	1.9	5
BusinessMasterReportDAO	2	22	4	1	1.79	2
BusinessMetricDataEntryDAO	6	60	19	2	5	8
BusinessReportDAO	4.8	35	8	0	4.5	9
CombinedSchIIReportDAO	7.5	37	4	2	4.41	2.5
CombineReportDAO	7.5	36	3	0	1.74	2
ExcelUploadDAO2	2	30	3	0	1.74	2
GeneralDataDAO	3.2	30	8	1	1.83	2
GroupMasterDAO	7.6	40	9	1	7.65	15
LoginDAO	5	22	2	0	1.63	2
MetricBusinessMasterDAO	1.5	14	1	1	1.74	2
MetricMasterDAO	5.71	40	12	0	5	10
PendingCashSchIIReportDAO	7.5	37	4	0	1.53	4
UserMasterDAO	4	49	8	0	3.59	4.5
ExportExcel2	4.86	50	7	0	3.83	10
AgingReportHandler	0.5	13	1	1	1.74	2
AuditReportHandler	0.33	13	1	0	1.53	2
BusinessMasterHandler	2	22	2	1	1.74	2
BusinessMasterReportHandler	1	17	1	0	1.53	3
BusinessMetricDataEntryHandler	2.67	37	4	1	1.85	5
BusinessReportHandler	1.67	32	3	0	1.79	2
CombinedSchIIReportHandler	0.33	13	3	2	1.79	3
CombineReportHandler	0.3	12	1	0	1.53	2

The correlation between the computed testability index values and observed testing time comes out to be 0.866, which indicates that testing time is strongly correlated with our computed value of testability index. This strengthens our belief that the proposed integrated measures can prove to be a good metric of software testability. As the integrated measure combines effects of four different metrics, each of which has got an effect on testability, this measure is expected to give better results than any of the individual metrics and the intuition has been verified with the help of empirical results shown above.

DISCUSSION

The proposed model measures the Software Testability based on four important measures of Object Oriented system- ACC, DIT, RFC and CBO. The fuzzy approach is used to integrate these four inputs and arrive at Testability Index for a class. The empirical data of unit testing time for Java classes has been collected and results have been validated. A strong correlation exists between unit testing time and output value of this model i.e. Testability Index. In most software projects managers feel challenge in estimating effort that should be spent on testing. Other side to look at the same problem is to find out if enough rigor has gone into the testing activity and hence what is the confidence on software quality? To be confident on software quality and at the same time to ensure software gets delivered within cost and schedule-Testability Index can be used to estimate testability of software unit (in OO world class). We for-see the following extension of our work. Our experimental basis should be extended. It is desirable to extend our findings to a large number of systems, developed by all sorts of teams using different development methodologies.

ACKNOWLEDGEMENTS

Thanks to Vikas Banga from Appulse Retail Software Solution (www.appulse.com), for his support through out the project and for providing the valuable data used for empirical validation and statistical analysis.

REFERENCES

1. IEEE Standard Glossary of Software Engineering Terminology, 1990. IEEE Computer Society.
2. Binder, R.V., 1994. Design for Testability with Object-Oriented Systems. Communications of the ACM, 37: 87-101.
3. ISO, 1991. International standard ISO/IEC 9126. Information Technology: Software product evaluation: Quality characteristics and guidelines for their use.
4. George, J. Klir and Bo Yuan, 2000. Fuzzy Sets and Fuzzy Logic: Theory and Applications. Prentice Hall of India, New Delhi.

5. Lee, Y., B. Liang and F. Wang, 1993. Some complexity metrics for object oriented programs based on information flow. Proceedings: CompEuro, pp: 302-310.
6. Lorenz, M. and J. Kidd, 1994. Object Oriented Software Metrics. Prentice Hall Publishing.
7. McCabe and Associates, 1994. McCabe Object Oriented Tool User's Instructions.
8. Rosenberg, L.H., 1997. Metrics for object oriented environments. EFAITP/AIE Third Annual Software Metrics Conference.
9. Sharble, R. and C. Samuel, 1993. The object oriented brewery: A comparison of two object oriented development methods. Software Engineering Notes, 18: 60 -73.
10. Tegarden, D., S. Sheetz and D. Monarchi, 1992. Effectiveness of traditional software metrics for object oriented systems. Proceedings: 25th Hawaii International Conference on System Sciences, pp: 359-368.
11. Chidamber, S.R. and C.F. Kemerer, 1994. A metrics suite for object-oriented design. IEEE Transactions on Software Engineering, 20: 476-493.
12. Voas, J. and K. Miller, 1993. Semantic metrics for software testability. J. Systems and Software, 20: 207-216.
13. Chidamber, S.R. and C.F. Kemerer, 1991. Towards a metrics suite for object oriented design. Proc. OOPSLA.
14. McCabe, T.J., 1976. A complexity measure. IEEE Transactions on Software Engineering SE-2, pp: 308-320.
15. Beizer, B., 1990. Software Testing Techniques. Van Nostrand Reinhold, New York, NY.
16. Aggarwal, K.K., Yogesh Singh and Jitender Kumar Chhabra, 2002. An integrated measure of software maintainability. Proceedings of Annual Reliability and Maintainability Symposium-International Symposium on Product Quality and Integrity RAMS- 2002, Seatle Westin U.S.A, Jan 28-31, pp: 235-241.
17. Marek, J. Patyra, L. Janos Grantner and Kirby Koster, 1996. Digital fuzzy logic controller: Design and implementation. IEEE Transactions on Fuzzy Systems, 4: 439-459.
18. Patyra, J. and D.M. Mlynek (Ed), 1996. Fuzzy Logic: Implementation and Applications. John Wiley and Sons Ltd. and B.G. Teubner.
19. George, J. Klir and Bo Yuan, 1995. Fuzzy Sets and Fuzzy Logic: Theory and Applications. Prentice Hall of India, New Delhi.
20. Ronald, R. Yager and D.P. Filev, 1995. Defuzzification with Constraints. Fuzzy Logic and Applications to Engineering, Information Sciences and Intelligent Systems. Edited by Z. Bien and K.K. Min. Kluwer Academic Publication, pp: 157-166.
21. Hellendorn, C. Thomas, 1995. On Quality Defuzzification: Theory and Application Example. Fuzzy Logic and Applications to Engineering, Information Sciences and Intelligent Systems. Edited by Z. Bien and K.K. Min. Kluwer Academic Publication, pp: 167-176.
22. Roger, J. and N. Gulley, 1995. Fuzzy Logic Toolbox for MATLAB, User's Guide. The Math Works Inc., USA.
23. Li,W. and S. Henry, 1993. Object-oriented metrics that predict maintainability. J. Systems and Software, 23: 111-122.
24. Brito e Abreu, F., 1995. The MOOD metrics set. Proc. ECOOP'95 Workshop on Metrics.
25. Briand L., P. Devanbu and W.L. Melo, 1997. An investigation into coupling measures for C++. 19th International Conference on Software Engineering (ICSE'97). Boston, pp: 412–421.
26. Bieman, J.M. and B.K. Kang, 1998. Measuring design-level cohesion. IEEE Transactions on Software Engineering, 24: 111-124.
27. Basili, V.R., L.C. Briand and W.L. Melo, 1996. A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering, 22: 751-761.
28. Brito e Abreu, F. and W.L. Melo, 1996. Evaluating the impact of object-oriented design on software quality. Proceedings of the Third International Software Metrics Symposium (METRICS'96). Berlin.
29. Briand L., J.W. Daly and J. Wust, 1998. A unified framework for cohesion measurement in object-oriented systems. Empirical Software Engineering, 3: 65-117.
30. Briand L.C., J.W. Daly, V. Porter and J. Wust, 1998. A comprehensive empirical validation of product measures for object-oriented systems. Technical Report ISERN-98-07.
31. Harrison, R., S.J. Counsell and R.V. Nithi, 1998. An investigation into the applicability and validity of object-oriented design metrics. Empirical Software Engineering, 3: 255-273.
32. Harrison, R., S. Counsell and V. Reuben, 1998. An evaluation of the MOOD set of object-oriented software metrics. IEEE Transactions on Software Engineering, 24: 491-496.
33. Briand L.C., J.W. Daly and J. Wust, 1999. A unified framework for coupling measurement in object-oriented systems. IEEE Transactions on Software Engineering, 25: 91-121.
34. Briand L.C., J. Wust, S.V. Ikonomovski and H. Lounis, 1999. Investigating quality factors in object-oriented designs: An industrial case study. 21st International Conference of Software Engineering (ICSE'99). Los Angeles, CA, pp: 345-354.
35. Linda, H. Rosenberg, Ruth Stapko and Albert Gallo. Risk-based Object Oriented Testing. NASA GSFC SATC NASA, Unisys SATC NASA, Unisys Code 302 Code 300.1