International Journal of Information Technology and Knowledge Management (ISSN: 0973-4414) July-December 2012, Volume 5, No. 2, pp. 484-489

A Fuzzy Model for Object Oriented Testability & its Performance

Surender S Dahiya[1], Smriti Bhutani[2], Ashish Oberoi[2], Manjeet Singh[3]
[1]Shivalik Institute of Engineering & Technology, Aliyaspur, Ambala
[2]Department of CSE, Maharishi Markandeswar Engineering College, Mullana, Ambala
[3]Department of CSE, YMCA University of Science & Technology, Faridabad
surendahiya@gmail.com, smritimtech@gmail.com

Abstract: For large software systems, the testing phase tends to have comparatively much higher cost than all the previous life-cycle phases taken together, obviously resulting in much more efforts. A good measure of software testability can help better manage the testing phase effort. This paper proposes a fuzzy model for measurement of software testability based on five-object oriented metrics. Empirical data of testing time of five software projects has been collected and used to evaluate the performance of the model.
Keywords: Testability, Fuzzy system modeling, CK metrics.

I. INTRODUCTION

Comprehensive testing is not realistic, as it is computationally infeasible. So, emphasis is on creating optimum test suits and designing software in such a way, so that faults reveal themselves during testing, if they present in software. This approach is also called testability of software.As nearly 40% of development efforts are spent on testing [1], a good measure of testability at designing phase can help in testable software and better management of testing resources.

Software testability is an external software attribute that evaluates the complexity and the effort required for software testing. Software testability has been defined and describe in literature from different point of views. The IEEE standard glossary defines testability as "the degree to which a system or component facilitates the establishment of test criteria and performance of test to determine where those criteria have been met". ISO defines it in a similar way: "attributes of software that bear on effort needed to validate the software product". Binder [7] relates software testability to two properties of the software under test: Controlability and Observability. Voas et al. [6] defines software testability basis on the software sensitivity to faults.

II. RELATED WORKS

Several models have been put forward to quantify the testability of conventional software [9] [10] [11], but these cannot be applied to Object Oriented (OO) programs as these fail to address the characteristics of OO paradigm, such as inheritance, coupling, polymorphism etc. Binder [7] listed a number of simple metrics to assess testability, including: lack of cohesion in methods, percentage of non-overloaded calls, percentage of dynamic calls, and depth of inheritance tree. He also mentioned that using stubs, and drivers, as well as assertions, can improve testability as they increase controllability and observability, respectively. However, it did not provide any empirical evidence that there is a correlation between the suggested metrics and testability and factors are also described at a high level of abstraction leading to

no-clear relationship with the metrics. Baudary etal. [4] also related testability to the testing effort and focus on class interactions in class diagrams: The authors identified patterns of class interactions that potentially lead to increased testing effort. Although it is shown on two examples that the metric is adequate to measure the testability of class interactions, the cause effect relationship between that metric and testability remains unclear.

Bruntinket al. [3] identified several qualitative and quantitative factors affecting testability of software. Qualitative factors include testing criteria, implementation, documentation, test suits and test tools. They related the testability of a system to the number of test cases required to test it and to the effort required to develop each individual test case. They found a correlation between class level metrics such as number of methods and test level metrics such as the number of test cases and the number of lines of code per test class, however, no correlation between inheritances related metrics (e.g. depth of inheritance tree) and the proposed testability metrics. Freedman [9] followed the notions of observability and controllability used in hardware testing to test a measure of testability. Observability captures the degree to which a component can be observed to generate the correct output for a given input. The notion of 'controllability' relates to the possibility of a component generating all values of its specified output domain. Briand et al. [5] illustrated an approach where instrumented contracts are used to increase observability, diagnosability and testability of a program. McGregor et al. [2] attributed testability to component visibility of the program.

As we have seen in the above discussion, authors have addressed only the partial concept of object orientation. Binder [7] and McGregor took inheritance related metrics to propose testability metrics but failed to include coupling concept. On the other side Baudry and Bruntink could not use inheritance measure to suggest a testability measure and some authors Briand and McGregor did not explore the code structure to propose a testability measure. Objects, classes, inheritance, encapsulation and polymorphism characterize object orientation. So to develop a unified measure for OO software testability, we need to address all these characteristics.

III. OBJECT ORIENTED METRICS FOR TESTABILITY

Bruntinket al. [3] identified several qualitative and quantitative factors affecting testability of software. Qualitative factors include testing criteria, implementation, documentation, test suits and test tools. They also listed out certain factors, which are analogous to OO metrics suggested by Kermer and Chidamber (CK) in such as **Weighted Method Count(WMC)**, **Lack of Cohesion among Methods (LCOM)**, **Depth in the Inheritance Tree (DIT)**, **Number of Children (NOC)**, **and Coupling between Objects (CBO)**etc. A great detail of these measures are given in [12]. Since, software is characterized by these measures; these should have some effect on testability of software and can be measured effectively.

A. Weighted Method Count (WMC): As per the fault/failure model, testability of a method may be expressed as a product of the execution rate and the fault propagation rate associated with that method. This is same as procedure oriented software testability. To find out the execution rate of a statement in method, method can be tested against a number of test suits generated randomly and then we can calculate the average execution rate of statements in a method. Cyclomatic complexity can be used an alternative to measure execution rate. To calculate propagation rate, Domain to Range Ratio (DRR) measure can be used. DRR is a simple metric that measures the ratio of the information that is flowing out of procedure and flowing in procedure [6]. Lo and Shi [8] have taken propagation rate as inversely proportional to DRR. So method testability can be taken as $\mathbf{T(m_i) = \frac{E(m_i)}{DRR}}$ where m_i is the ith method and $E(m_i)$ is the execution rate of method.

Lo and Shi [8] have argued that every method in a class contributes equally to the execution rate of a class, so if there are M methods in a class then there is a 1/M chance of a test case passing through a specific method. Therefore, the number of methods in a class is inversely proportional to the testability of a class. The larger the number of methods in a class, the lower is the testability of the class. But we do not endorse this argument. As all methods cannot be equally important to class, we propose to take a WMC like measure in place of NOM because. As testability is inversely proportionally to number of method, weight of each method can be defined as .

B. Lack of Cohesion among methods (LCOM): The cohesion measures the degree of similarity of methods within a class. High cohesion promotes encapsulation and it usually means decreasing the number of methods, the size of the class and hence the testing efforts. On the other side low cohesion usually denotes poor design or poor organization of a class and thereby increasing the complexity of a class and the likelihood of errors. Therefore the more cohesive are the methods within a class, the higher the testability of that class. The lack of cohesion among methods results in low testability of the class.

C. Depth in the inheritance tree (DIT):DIT is defined as the maximum length from the node to the root of the tree. The deeper the inheritance tree for a class, more and more features it inherits from other classes making it more complex to test and maintain. The nearer the class to the root of an inheritance, probability that it will be tested is greater since every time a test case goes through its offspring, it will also go through the class itself and thus increasing its testability. However, the deeper a particular class is in the hierarchy, the greater the potential for reuse of inherited methods.

D. Coupling between Objects (CBO): The communication between components within an OO program is actually the interaction or coupling between classes. Coupling between classes through message passing is detrimental to modular design and prevents reuse, since this kind of coupling tends to complicate the relationship between classes, making the refinement and maintenance of the coupled classes difficult. Probability of introduction of faults in software is more with increased interaction between classes and thus more efforts are required in software testing. The larger number of classes a given class is coupled to, the greater the effort is needed to find software faults among the entangled classes if faults do exist. As a result the testability of the class is lower.

E. Number of children (NOC):A Class with a large number of children has higher execution rate since according to the fault-failure model, all test cases going through the children of a class will also go through this class. Therefore the greater the number of children a class possesses, the higher the percentage of test cases will go through this class resulting in higher testability. But high values of NOC may indicate an inappropriate abstraction in the design [12]. A moderate value for NOC is desirable.

IV. FUZZY MODEL FOR PRORAM TESTABILITY

Lo and Shi [8] have tried to find out a unified measure for testability. But their approach is erroneous due to several accounts. First, they have multiplied the testability derived from above factors. As these factors are parallel dependent or independent they cannot be multiplied to arrive on a single number for system testability. Shortcoming at one place cannot diminish the testability of the whole system drastically. Moreover all the factors have different unit of measurement so even multiplication cannot be used without normalization or without suggesting some weightage to each other. To handle all these shortcomings Fuzzy Logic is an ideal choice. Second they have used coupling to measure class testability, while it is a measure of interaction between classes not of class internals. Third, they have not suggested any measure of testability for the whole system.

Five input variables WMC, LCOM, DIT, CBO and NOC are taken for testability fuzzy system. Assume normalized range from 0 to 1 for every input variable we have partitioned into three fuzzy sets namely LOW, MED and HIGH.

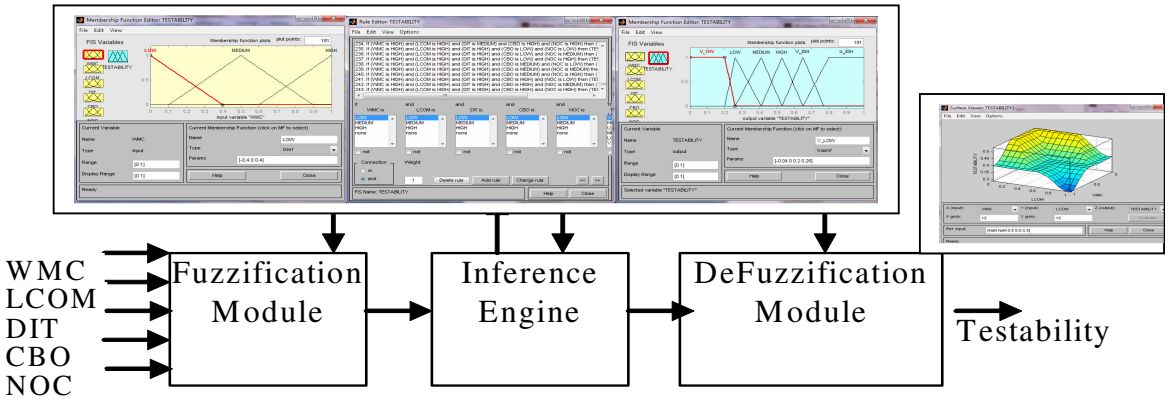

Figure 1. Fuzzy based testability Model

The output of the system Testability is specified by six Membership functions viz. V_LOW, LOW, MED, HIGH, V_HIGH and U_HIGH as shown in figure 1. After output specification, the next step is rulebase generation. In order to measure the software testability using the five input metrics having three MFs each, two hundred forty three rules have been defined. Now the degree to which the inputs belong to each of the appropriate fuzzy sets is determined. Based onthese fuzzy inputs, some rules get fired. The outputs of all fired rules are aggregated and are then defuzzified using centroid technique to get a crisp value of testability on scale of 0 to 1. Lower value of testability reflects bad testability, while higher value indicates good testability.

v. EXPERIMENTAL RESULTS AND DISCUSSIONS

In order to validate the model, we considered five object-oriented software projects of real world. First Program P1 is a quality-designed code, the collections API. This source is found in J2SDK 1.4.2 in java.util. Second program P2 is acceptance-testing framework program called Fitness which is an open-source. The source code for 'Fitnesse' program is available at: https://sourceforge.net/projects/fitnesse/. Third program in the list P3 is JDOM which is an API that provides a way to represent an XML document for easy and efficient reading, manipulation, and writing. The source code for JDOM is available at: http://www.jdom.org/Second last in the list P4 is BonForum which is an open-source chat application. It is described fully in the book, "XML, XSLT, Java and JSP" written by Westy Rockwell. The source code can be downloaded at: http://sourceforge.net/projects/bonforum/. The last one P5 is a student project which consists of a user interface that could be applied to a chat program.

CK metrics of these projects have been measured by "Metric 1.3.4" which is a plug-in for Eclipse; to collect the metrics the considered projects do not have to be compiled since the tool measures on java files. The tool can be downloaded at http://metrics.sourceforge.net/. Now as we need to prove only relationship between actual testing effort and testability evaluated by Fuzzy model rather than measuring an absolute value. The absolute values have been changed to normalized values by dividing each value in the row by highest value in row making each row value range from 0 to 1 as shown in table 1.

Table 1. Normalized values of CK metrics of five programs

Metric /System	P1	P2	P3	P4	P5
WMC	0.43	0.12	0.74	1.00	0.12
LCOM	0.55	0.48	0.59	1.00	0.16
DIT	1.00	0.47	0.17	0.00	0.00
NOC	0.78	1.00	1.00	0.98	0.12
CBO	0.75	0.53	0.92	1.00	0.58
Time	0.84	0.75	.67	1.00	.084
Testability	0.398	0.395	0.363	0.29	0.607

For validation purpose, we have recorded the testing effort in the form of time taken to test these systems. Testing time includes test case generation which can execute at least 95% of LOC. The output value of testability was also calculated using the proposed fuzzy model and the corresponding normalized values with range from 0 to 1 for each of the projects is also listed in this Table 1. In the second last row of this Table, the normalized testing time in the range of 0 to 1 for each of these five projects is mentioned.The values of manual testing time of these five projects have been plotted against each of the four input metrics i.e. WMC, LCOM, DIT, CBO and NOC in Figures 2.

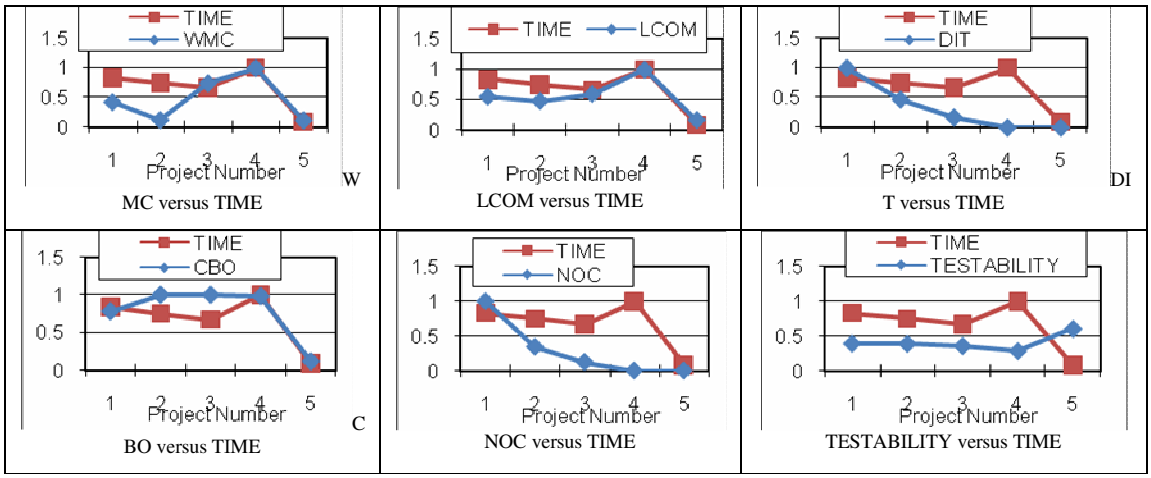

Figure 2: Plot of various attributes versus TIME

It can be easily observed that there is hardly any correlation existing between time and any of the three input metrics WMC, DIT and NOC. It is just around 25% in case of DIT and NOC and nearly 68% in case of WMC. In case of LCOM and CBO correlation is more than 90%. These values of correlation clearly depict that none of these metrics except LCOM and CBO individually is sufficient to predict testability. In case of LCOM and CBO more experimentation is needed. On the other hand a plot of testing time with computed values of testability clearly shows a strong negative correlation between the two curves. The correlation between computed testability values and observed testing time comes out to be 0.98, which indicates that the testing time is strongly correlated with our computed value of testability. This strengthens our belief that the proposed measure can prove to be a good metric of software testability. As the proposed measure combines effects of five 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 this intuition has been verified with help of the empirical results shown above.

VI. CONCLUSIONS

This paper has proposed a fuzzy model based on five-parameter for measurement of software testability. The empirical data of testing time of different software projects has been collected and the results have been validated. A strong correlation has been observed between testing time and the output value of this model i.e. testability. Thus output of this model can advise the software project managers in judging the testing efforts of the software. The empirical validation also suggests that two metrics also individually strong correlation with testing effort. This has to be verified further with more experimentation. Our next effort will toward finding more factors responsible for testability in object oriented system.

REFERENCES

[1] R. S. Pressman, "Software Engineering", McGraw- Hill, NewYork, 1992.

[2] McGregor and S. Srinivas, "A measure of testing effort", proceedings of the USENIX 1996.

[3] M. Bruntink and A. V. Deursen, "Predicting Class Testability using Object-Oriented Metrics," Proc. IEEE International Workshop on Source Code Analysis and Manipulation, pp. 136-145, 2004.

[4] B. Baudry, Y. Le Traon and G. Sunyé, "Testability analysis of a UML class diagram," Proc. IEEE Symposium on Software Metrics, pp. 54-63, 2002.

[5] L. C. Briand, Y. Labiche and H. Sun, "Investigating the Use of Analysis Contracts to Improve the Testability of Object-Oriented Code," Software - Practice and Experience, vol. 33 (7), pp. 637-672, 2003.

[6] J. M. Voas and K.W. Miller, "Software testability: The new verification", IEEE Software, pp. 17-27, May 1995.

[7] R. V. Binder, "Design for Testability in Object-Oriented Systems," Communication of the ACM, vol. 37 (9), pp. 87-101,1994.

[8] B.W.N. Lo and H. Shi. "A Preliminary Testability Model for Object-Oriented Software," International Conference on Software Engineering: Education & Practice, p. 330, 1998.

[9] R.S. Freedman, "Testability of software components", IEEE Transactions on Software Engineering, Vol. 17, No. 6, pp. 553-564, June 1991.

[10] W. Howden and H. Yudong, "Software testability analysis", ACM Transactions on Software Engineering and Methodology, Vol. 4,pp. 36-64, January 1995.

[11] J.M. Voas and K.W. Miller, "Improving the software development process using testability research", IEEE Software, pp. 114-121, 1992.

[12] S.R. Chidamber& C.F. Kemerer, "A metrics suite for object oriented design", IEEE Transactions on Software Engineering, Vol. 20, No. 6, pp. 476-493, June 1994.