I.J. Information Technology and Computer Science, **2015, 03, 18-26**
Published Online February 2015 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2015.03.03

An Assessment of Software Testability using Fuzzy Logic Technique for Aspect-Oriented Software

Pradeep Kumar Singh[1], Om Prakash Sangwan[2]
Amity University Uttar Pradesh, Noida, India[1] and Gautam Buddha University, Gr. Noida, India[2]
Email: pradeep_84cs@yahoo.com, sangwan_op@yahoo.co.in

Amar Pal Singh[3], Amrendra Pratap[3]
Amity University Uttar Pradesh, Department of CSE, Noida, India[3]
Email: singhamarpal48@gmail.com, amrendra.bt11@gmail.com

Abstract—Testability is a property of software which introduces with the purpose of forecasting efforts need to test the programs. Software quality is the most important factor in the development of software, which can be depend on many quality attributes. The absence of testability is responsible for higher maintenance and testing effort. In this paper Fuzzy Logic is used to ascertain the relationship between the factors that affects the software testability. This paper presents the application of fuzzy logic the assessment of software testability. A new model is proposed using fuzzy inference system for tuning the performance of software testability. Aspect-oriented metrics are taken i.e. Separation of Concern (SoC), cohesion, size and coupling. These metrics are closely related to the factors i.e. Controllability, Observability, Built in Test Capability, Understandability and Complexity. These factors are independent to each other and used for accessing software testability. A Triangular Membership Function (TriMF) is applied on these factors which defined in Mamdani Fuzzy Inference System in MATLAB. In this paper, we have defined and evaluated factors combination which is used for the assessment of software testability for as well as aspect oriented software.

Index Terms— Aspect Oriented Programming (AOP), Aspect Oriented Software Development (AOSD), Aspect Oriented Software (AOS), Fuzzy Logic, Aspect-Oriented Metrics, Separation of Concerns (SoC), Software Testability.

I. Introduction

Software Testability is one of the quality metric of Software and ISO has defined software testability as a functionality and it defines functionality as "the collection of characteristics of software that bear on the effort required to authenticate the software produced" [1, 2].

IEEE defines it as "An activity in which a component or a system is evaluated for some specific conditions, the results are examined and evaluation is based on some aspect of the component or the system" [3].

It is also well known reality that more than 50% of the total cost in the development of software is related to the software testing activities [4]. Hence, in software development life cycle, it is the most expensive phase in terms of efforts needed, money as well as time. So, it is very important to reduce the efforts and time required for testing the software. Many researchers have focused their study for the solutions to minimize the testing cost. If the testability of software can be improved, then it is possible to reduce the software cost along with achieving the higher easiness in writing test cases, test automation, fault detection.

The software testability is important during testing, coding, quality assurance, and designing [5]. Testable software attributes like low complexity, low coupling and good separation of concerns formulate an easier way for reviewers to realize the software artifacts [6].

"Software Testability" is a challenging issue to be investigated in the process of software development for improving the effectiveness of testing process. Software testability cannot be measured directly so it can be measured by the qualitative factors that affect testability. The testability of software components is determined by factors such as:

i) Controllability: The degree to which it is possible to control all values of its individual output domain.
ii) Observability: The degree to which it is possible to observe accurate output for a specified input.
iii) Built in Test Capability: It has the ability to test the software itself. It reduces the complexity as well as decreases the cost of software. It can improve controllability and observability.
iv) Understandability: The degree to which the component under test is documented or self-explaining.
v) Complexity: It is the quantitative measurement of the complexity of the program. Low complexity of any software system is an indication of high quality.

In this paper we have mainly focused on software testability in context of AO software. AOSD and its major features are discusses below.

A. Aspect-Oriented Software Development (AOSD)

Aspect oriented software development (AOSD) has widely used in industry as well as research environments. An AO system requires new measurement of frameworks

to evaluate the maintainability and testability degrees. Sant'Anna *et al.* [7] proposed a model for AO systems makes up of two elements: a quality model and a set of metrics. These elements are derived from existing metrics and familiar principles to keep away from the recreation of well tested results. Based on suggested model, a quantitative assessment, the drawbacks and advantages of the model were discussed for AO software.

Popular programming languages for aspect-oriented systems are AspectC (a C extension), AspectC++ (a C++ extension), AspectXML (a XML extension), AspectL (a LISP extension), AspectJ (a Java extension), CaesarJ and Hyper/J (being used by IBM). This work focused on those aspect-oriented programming languages that have new AO features and almost all the features of Java programming language. In this category, it is founded that AspectJ and AspectJ-like (Springs AOP framework, JBoss etc.), CaesarJ and Hyper/J are most famous programming languages. Aspect-oriented programming was developed to overcome the limitations of programming approaches such as OOP in handling crosscutting concerns. Aspect-orientation offers a new modularization way by separating cross-cutting concerns from non cross-cutting ones.

From the available AOP languages, AspectJ is the most popular and mostly used in research areas. AspectJ [8] is an easy extension to java which presents during the modular execution of crosscutting concerns and description of new constructors. It consists of mainly (i) Join points are code sections in the execution of a software where aspects are applied; (ii) Point-cuts are declarations responsible for selecting join points, that is, detecting which join points the aspect should intercept; (iii) Advices are code used to implement crosscutting concerns; (iv) Introductions (inter-type declarations) are code that structurally modifies a class, adding new members and relationships to it through a declare parents clause; (v) Aspects are entities that encapsulates point cuts, advices, and introductions in a modular code unit, defined similarly to classes [9]. This paper is divided in seven sections.

In first section, introduction of software testability in context of aspect oriented software is discussed, followed by literature review based on factors and metrics which affects software testability. In third part of this paper, aspect oriented design quality metrics are described. In fourth section, fuzzy logic approach is discussed. In fifth section, the simulation of fuzzy model is provided. Finally, results are discussed followed by future scope.

II. Related Works

A number of testability theories have been published till date and the testability concept has been grown with different research states. Some of the important theories given by researches in their paper's and discussions motivate us for the proposed work. However, till date less number of theories deal with AO programs quality evaluation.

Zhao [10] gives the earliest suggestion in the field of coupling measurement for aspect oriented systems. Zhao *et al.* [11] gives aspect cohesion assessment. It is derived from a dependency framework for AO software.

Ceccato *et al.* [12] extend the use of Chidamber and Kemerer's [13] metrics suite for measuring the software aspectization for AO systems.

Sant'Anna *et al.* [7] suggested a metric framework for AO programs. Zhang [14] proposed the size, complexity, coupling between objects, response time, no. of classes for their assessment.

Tsang *et al.* [15] used the CK metrics framework for their assessment for AO systems. The quality factors considered are maintainability, understandability, testability and reusability and the CK metrics is taken into account for evaluation.

Mulo [16] considers the two main factors of testability that are controllability and observability and strongly recommended that these can improve the ability of tester for good control of software.

Wang [17] identified the factors in order to improve testability that are controllability, observability, built in test capability, visibility, operability, simplicity, understandability, suitability.

Pan *et al.* [18] introduces a framework for an aspect oriented testability in which observability and controllability are taken as a factor. This paper mainly focuses on improving software observability.

Bach [19] describes practical testability is a function of five types of testability's i.e. project related testability, value-related testability, subjective testability, intrinsic testability and epistemic testability.

Basili *et al.* [20] gives a quality framework for metrics. According to authors viewpoint, testability degree of AO systems should be accessed through metrics and factors.

Khan *et al.* [21] proposed framework based on testability for object oriented design. In this framework inheritance, encapsulation, cohesion and coupling metrics are used.

Shaheen *et al.* [22] considered a survey on metrics for accessing testability of object oriented systems in which various metrics have been taken to identify the testability for object oriented systems.

Nazir *et al.* [23] proposed a list of commonly accepted factors in order to find out measurable characteristics of software testability.

Abdullah *et al.* [24] describes some observation. Authors mentioned firstly; In order to reducing effort in measuring testability of object oriented design we need to identify a minimal set of testability factors for object oriented development process, which have positive impact on testability measurement and secondly, testability metrics must be selected at the design phase because metric selection is an important step in testability estimation of objects oriented design.

Binder [25] described testability as the cost of disclosing software faults and the relative ease. Binder offers an analysis of the factors which are contributing to the software testability. Binder also listed some of the testability metrics from encapsulation metric, inheritance

metric and polymorphism metric. Encapsulation metric includes public access to data members, lack of cohesion in methods, while some of the inheritances metric are number of children, depth of inheritance tree and also polymorphism metric includes percentage of non-overloaded calls, percent of dynamic calls. He also said that controllability and observability will ultimately increase the software testability.

Bruce *et al.* [26] recognize the factors which affect testability in object oriented software and classified them in communication factors, inheritance factors and structure factors.

Bach [19] proposed testability factors as are operability, visibility, control, understandability, simplicity, stability, and suitability. Morris [27] tells that the observer pattern can also provides good support for separation of concerns.

Jungmayr [28] described factors that affect testability as separation of concerns, complexity, fault locality, coupling, observability, controllability, built-in-test capability, diagnostic capability, and automatability.

Gao *et al.* [29] identified a framework for the measurement of testability based on the factors i.e. observability, controllability, traceability, process capability and understandability.

Voas *et al.* [30] described testability as the possibility that a part of software fails in subsequently execution during testing. They have suggesting dynamic technique called software sensitive analysis for evaluating the software testability.

Bache *et al.* [31] define testability as a quality attribute to the testing effort needed and test criterion is an important factor of the testing effort.

Singh *et al.* [32] proposed metrics and model based on maintainability assessment for aspect oriented software.

The research by Bruntink *et al.* [33] is mainly concerned with identifying and evaluating the factors of testability in object oriented software and metrics related to the factors, which have been supported by the case studies.

The concept of testability of a software component was initiated by Freedman [34] considering controllability and observability.

Boxall *et al.* [35] determined the level of testability affected by understandability and it can be determined by interface properties.

Singh *et al.* [36] discussed the techniques for fault based mutation testing and tools for AspectJ programs and how these tools can be helpful in evaluating testability.

Piveta *et al.* [37] gives empirical data and analytical evaluation from ten projects which determines the six metrics for aspect oriented software i.e. weighted operations in module, lines of code, number of children, depth of inheritance tree, coupling on advice execution and crosscutting degree of an aspect and talk about how these metrics can be accustomed to recognize weakness in existing aspect oriented software.

Malla *et al.* [38] proposed software testability factors i.e. controllability, observability, understandability, complexity, process capability and related metrics i.e. size metrics, inheritance metrics and coupling metrics.

We have identified from above studies that testability of software can be measured using metrics for aspect-oriented system as well as for object-oriented system. Factors of testability i.e. controllability, observability, built in test capability, understandability and complexity can be measured using these design metrics. We have already conducted the detailed analysis on software testability for AO software in our previous work [53] too. A look on the factors that affect software testability is provided in Table 1.

Table 1. Factors of Software Testability

Factors Affecting Software Testability \ Authors	Binder 1994 [25]	*Jungmayr 2002 [28]*	*Wang 2003 [17]*	Gao 2005 [29]	Mulo 2007 [16]	Malla 2012 [38]	Bach 2013 [19]
Controllability	Y	Y	Y	Y	Y	Y	Y
Observability	Y	Y	Y	Y	Y	Y	Y
Built in test Capability	Y	Y	Y	Y	-	-	-
Traceability	-	-	-	Y	-	-	-
Automatability	-	Y	-	-	-	-	-
Understandability	-	-	Y	Y	-	Y	Y
Fault Locality	-	Y	-	-	-	-	-
Simplicity	-	-	Y	-	-	-	Y
Diagnostic Capability	-	Y	-	-	-	-	-
Complexity	-	Y	-	-	-	Y	-
Separation of Concerns	-	Y	-	-	-	-	-
Availability	-	-	-	-	-	-	Y
Decomposability	-	-	-	-	-	-	Y
Visibility	-	-	Y	-	-	-	Y
Suitability	-	-	Y	-	-	-	Y
Test suite	Y	-	-	-	-	-	-
Development process	Y	-	-	-	-	-	-
Coupling	-	Y	-	-	-	-	-
Stability	-	-	-	-	-	-	Y
Operability	-	-	Y	-	-	-	Y
Process Capability	-	-	-	Y	-	Y	-

In Table 1, the factors that are in Italic format also affects the software testability in Aspect Oriented environment as well as for object oriented software. However, remaining factors only affects in object oriented environment. Brief information about the software testability metrics is given in Table 2. In Table 2, the authors that are in Italic format have used aspect oriented environment for qualitative assessment in their research works. A look on software testability metrics and their related factors is reported in Table 3.

Table 2. Metrics of Software Testability

Authors	Metrics			
	SoC	Cohesion	Coupling	Size
Binder 1994 [25]	-	-	Y	Y
Bruce 1998 [26]	-	Y	Y	-
Jungmayr 2003 [39]	-	-	Y	-
Bruntink 2003, 2006 [40][33]	-	Y	Y	Y
Khan 2009 [21]	-	Y	Y	-
Shaheen 2010 [22]	-	Y	Y	-
Nazir 2010 [41]	-	-	-	Y
Eamar 2010 [42]	-	Y	Y	Y
Burrows 2010 [43]	Y	Y	Y	-
Malla 2012 [38]	-	-	Y	Y
Saraiva 2012 [44]	Y	Y	Y	Y
Piveta 2012 [37]	Y		Y	Y
Huang 2013 [45]	-	Y	Y	Y

Table 3. shows the factors that affects the software testability based the related work.

Table 3. Factors and related Metrics of Testability

Factors	Metrics
Controllability	Coupling metrics, Cohesion metrics, Size metrics
Observability	Coupling metrics
Built in test capability	Cohesion metrics, Coupling metrics
Understandability	Coupling metrics, Cohesion metrics
Complexity	Coupling metrics, Cohesion metrics, McCabe Complexity

III. Aspect-Oriented Design Quality Metrics

Our approach for evaluation and identification of testability of software is based on software quality factors and their related metrics. The aim of this work is to develop a model for the assessment of testability that can be measured using these metrics. AOSD has straight impact on the separation of the concerns and system size. Some predefined metrics cannot be used directly in aspect oriented software. Hence, AOSD identifies new cross cutting concern techniques and different type of abstractions. Some factors have been extensively recognized in both for OO and AO like coupling, size and cohesion. SoC in AO software differentiate the AOP from OO programming because of crosscutting concerns. We have identified four main metrics for the testability evaluation in AO software.

A. Separation of Concerns (SoC)

Separations of concern is the new metrics. Within [12] first time separation of concern metrics were anticipated. SoC is a well-established standard in software engineering. SoC is the ability to encapsulate, manipulate and identify those sections of software which are related to a particular concern. As yet, these metrics requires the identification of concerns manually. SoC metrics are as follows:

i) Concern Diffusion over Components (CDC): It contribute for execution of a concern and measures the total number of components that access the primary components using formal parameters, local variables, attribute declarations, throws declarations and return types.
ii) Concern Diffusion over Operations (CDO): It contribute for concern implementation and measures the total number of primary operations also counts advices and the number of methods which access return types, formal parameters, local variables, throws declarations and constructors.
iii) Number of Attributes per Concern (NOAconcern): It measures the total number of attributes for every aspect or class.
iv) Number of Operations per Concern (NOOconcern): It measures total no of operations in every concerns (inherited, public, private).

B. Coupling Metrics

Coupling between objects [46] can be defined as no. of coupled classes within all classes. Low coupling is desirable for better design. In [13], a metric suite for OO is provided and coupling is considered as one of the main metrics in framework. Coady *et al.* [47] provides separate set of metrics for couplings in aspect oriented. Coupling metrics are as follows: (i) Coupling between Objects (CBO) (ii) Depth of Inheritance Tree (DIT), (iii) Number of Children (NOC).

C. Cohesion Metrics

The measurement of cohesion for a component is the proximity of the relationship in internal components. The measurement is based on the methods of a class and how those methods are correlated to each other. Fenton *et al.* [48] defined cohesion in more illustrative way. Main cohesion metric is LCOM. If the value of LCOM becomes higher then cohesion is low.

D. Size Metrics

Size metrics are dependent on the length of program. We can measure it by LOC. As an aspect viewpoint it might be count a pointcut as a code line. In [13], LOC metric has been measured and avoids duplicacy in lines by writing pointcuts. For determining physical code and design the size metrics are as follows:

i) Number of system components (NOSC): It measures only the component names for the number of aspects and classes in the system. It is also known as Vocabulary Size (VS).
ii) Lines of Code (LOC): It measures the number of code lines, comment lines, blank lines and documentation lines are not counted as part of LOC. It is one of the traditional metric for measuring size of the project.
iii) Number of Attributes (NOA): It measures the number of attributes for every component. As per our generic framework, inherited attributes have also been included in counting.
iv) Number of Operations (NOO): It measures the number of operations for every component. It also includes inherited operations in our framework.
v) Number of Statements (NOS): It measures the number of statements in a method.
vi) Weighted Operation per Component (WOC): It counts the complexity of a component by counting the number of arguments of the operations.

As, testability of AO Software mainly depends on the metrics such as (i) Separation of Concerns (SoC), (ii) Cohesion, (iii) Coupling and (iv) Size. These metrics relates to the factors i.e. controllability, observability, built in test capability, understandability and complexity. In order to access testability of AOS, it is also very difficult to determine percentage of contribution of these metrics in testability. To overcome these difficulties, we adapted fuzzy logic technique considering the four metrics as input variable and testability as output variable.

IV. Fuzzy Logic

Fuzzy logic is a systematic technique to solve the problems that are very complex to understand quantitatively. It is a tool which deals with uncertainty and imprecision [49]. It is less dependent on historical data and fuzzy model can be built with less data [50, 51].

The fuzzy system accepts vague statements and imprecise data using the available membership functions and gives decisions as shown in Fig.1.

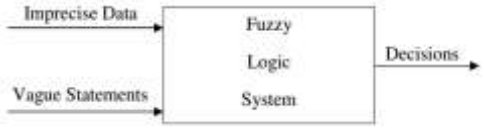

Fig. 1. Fuzzy Logic System

The fuzzy model gives mapping from input to output. Architecture includes four different modules. The fuzzification module converts the crisp input values into fuzzy values. Fuzzy values are forwarded by an interface engine derived from rule base in the knowledge base given by domain experts. Finally, defuzzification module converts fuzzy data to crisp values. The fuzzy model architecture is shown in Fig. 2.

In this proposed work, testability of AO system is a measure of different factors: Controllability, Observability, Built in Test Capability, Understandability and Complexity. These factors are independent to each other. These joined factors used in the measurement for the dependent variable i.e. testability because we can not directly measure the testability.

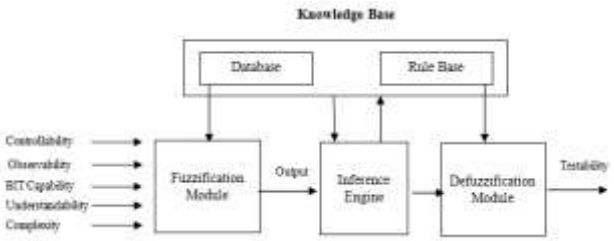

Fig. 2. Fuzzy Model

The proposed fuzzy logic considers all these factors as inputs and finally gives a crisp value of testability using rule base. All input values are categorized as low, medium and high. The output testability is categorized as very low, low, medium, high and very high. A rule base is created using all feasible arrangements of inputs. Fuzzy Inference System (FIS) includes the following module which is shown in Fig. 3.

i) FIS Editor: It shows information related to fuzzy inference system to handle the complex problems for the system.
ii) Membership Function Editor: It describes the shapes of all the membership functions related to every factor.
iii) Rule Editor: It is used for editing the rules which determines the behavior of the problem.
iv) Rule Viewer: It is used to see the rule base i.e. how an individual rule affects the results.
v) Surface Viewer: It is used to see the graph on the basis of given inputs and the output for the system.

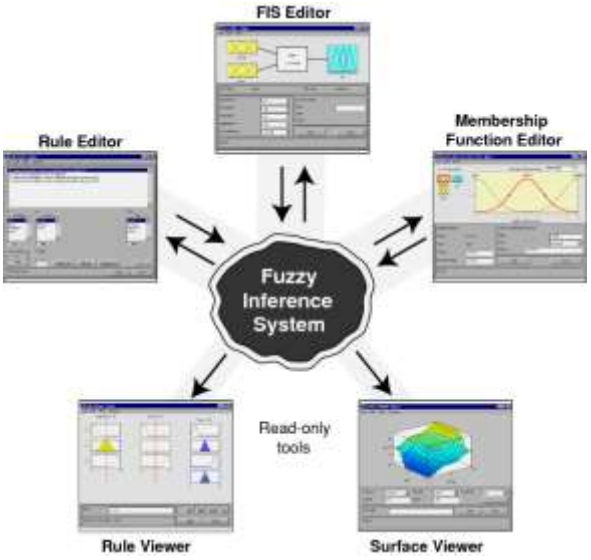

Fig 3. Fuzzy Inference System [52]

V. Simulation and Experimentation

In this section, the trained fuzzy inference system and proposed fuzzy model with inputs i.e. Controllability, Observability, Built in Test Capability, Understandability and Complexity to predict software testability as output given in Fig. 4.

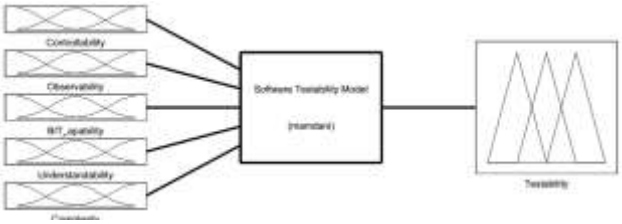

Fig. 4. Software Testability Model with 5 inputs, 1 output, 243 rules

All inputs are categorized into membership function i.e. low, medium and high and the software testability as output is categorized as very low, low, medium, high and very high. Triangular membership function is used to categorize the inputs and output scaled between [0 1] scale as follows:

For inputs, Low [0 0.185 0.37], Medium [0.31 0.495 0.68], High [0.63 0.815, 1] and For output, Very Low [0 0.1171 0.2341], Low [0.192 0.317 0.422], Medium [0.382 0.4985 0.6151], High [0.5732 0.6907 0.8082], Very High [0.7685 0.8842 1]

All 243 rules created and inserted in rule base, which represents all possible combinations of inputs i.e. 3^5 (243) sets. Some of the proposed rules are as follows:

Rule 1: If Controllability is high and Obsesrvability is high and BIT Capability is high and Understandability is high and Complexity is high then Testability is very low.

Rule 2: If Controllability is high and Obsesrvability is high and BIT Capability is low and Understandability is low and Complexity is low then Testability is low.

Rule 3: If Controllability is medium and Obsesrvability is medium and BIT Capability is medium and Understandability is medium and Complexity is medium then Testability is medium.

Rule 4: If Controllability is low and Obsesrvability is low and BIT Capability is low and Understandability is high and Complexity is low then Testability is high.

Rule 5: If Controllability is low and Obsesrvability is low and BIT Capability is low and Understandability is low and Complexity is high then Testability is very high.

High testability signify the high testing effort and cost which may lead to further higher maintainability and maintenance cost too. Details about the proposed fuzzy system are provided in Table 4.

Table 4. Details of the System used

Name of the system	'Testability'
Type of model	'Mamdani'
MATLAB version	7.8.0
No. of inputs	5
No. of outputs	1
No. of rules	243
No. of MF's used in inputs	3
No. of MF's used in output	5
Range	[0, 1]

Membership function for Controllability as input 1 in fuzzy logic tool is shown in Fig. 5.

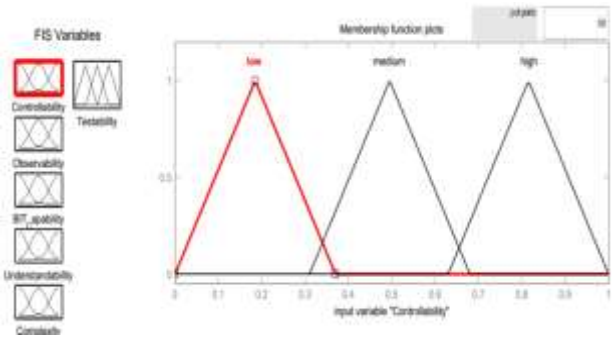

Fig. 5. Membership Function

All 243 rules are created a rule base is made which represents all possible combinations of inputs i.e. (243) sets as shown in Fig. 6.

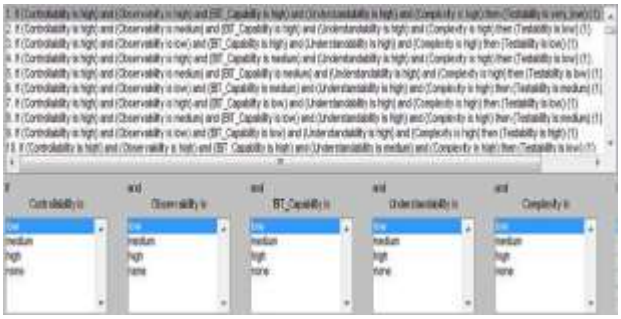

Fig. 6. Proposed Fuzzy Model Rule Base

Using a rule viewer shown in Fig. 7, testability is measured using five values of input factors.

Testability can be examined by some changes in the above rule viewer are shown in Fig. 8, which reflects the change in output. That is, for a set of inputs [0.8, 0.8, 0.8, 0.8, 0.2], the output is 0.117 which is the best result for testability.

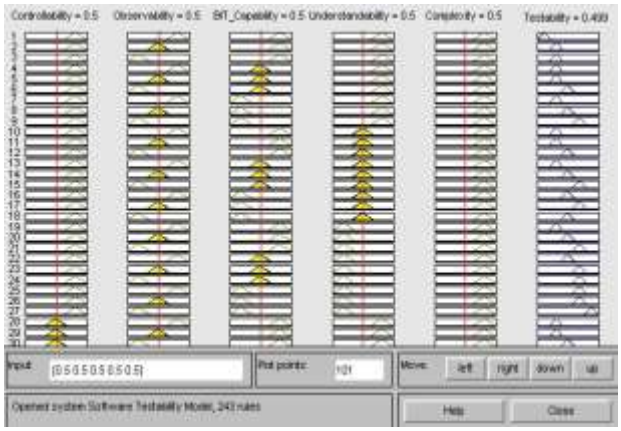

Fig. 7. Rule Viewer

Fig. 8. Rule Viewer with changed values

VI. Discussion of Results

This paper discusses testability in relation to AO Software. It identifies the factors which affecting testability and sets up a relationship on these factors for testability. Proposed model based on five factors: Controllability, Observability, Built in Test Capability, Understandability and Complexity for accessing Software Testability levels using AI techniques by Fuzzy Logic.

Some values assumed for Controllability, Observability, Built in Test Capability, Understandability and Complexity is taken as inputs and triangular membership function is used to defined in MATLAB in order to predict the Testability as output.

Proposed model categorized inputs as low, medium and high and testability as a output which is categorized as very low, low, medium, high, and very high. Total 243 rules are created based on expert advise and inserted in the rule base, which represents all the probable combinations of inputs i.e. (243) sets. Using rule viewer, testability can be determined by the proposed testability model using fuzzy logic.

For inputs [0.5, 0.5, 0.5, 0.5, 0.5]

- For Triangular Membership Functions (trimf), the testability is determined by taking all five values of input factors is 0.499 which is medium as shown in rule 3.

But for different inputs that is,

- For a set of inputs [0.8, 0.8, 0.8, 0.8, 0.2], the output is 0.117 i.e. Very Low which means best testability.

Rule-2 shows software testability value low & Rule-1 shows testability very low. This is shown in Fig. 9.

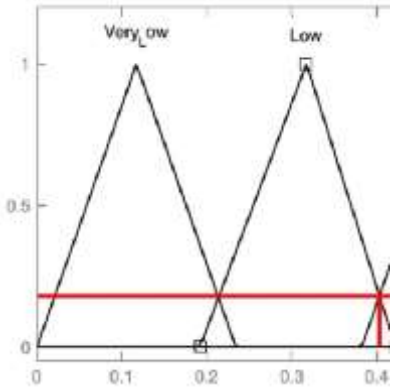

Fig. 9. Output for Software Testability

It has been identified based on expert advice and judgment that for best Testability of software, its Controllability, Observability, Built in Test Capability, and Understandability should be High whereas Complexity should be Low. Best testability is the ability of AO software to validate modified software where less testing efforts are required.

A three-dimensional plot is given in Fig. 10 that represents the surface view which mapped from controllability and observability factor on input axex (X and Y) to testability on output axes (Z). As similar to Fig. 10, more 3D figures can be drawn by considering another set of inputs and outputs (testability).

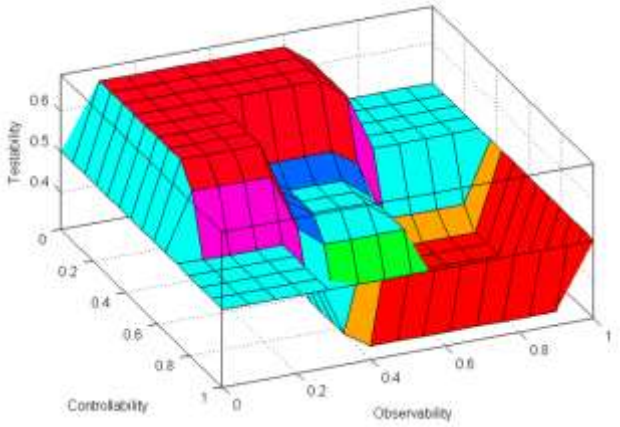

Fig. 10. Surface view w.r.t. inputs Controllability and Observability and output as Testability

A. Defuzzification

Defuzzification is a technique to produce a quantitative solution in fuzzy system. The fuzzy output needs to convert in a scalar quantity output. This process is called defuzzification.

One of the defuzzification technique is center of gravity also known as centroid method. In this implementation COG method is used for the aggregated output of 243 rules. COG method using (1), is applied on the output for testability as shown in Fig. 11.

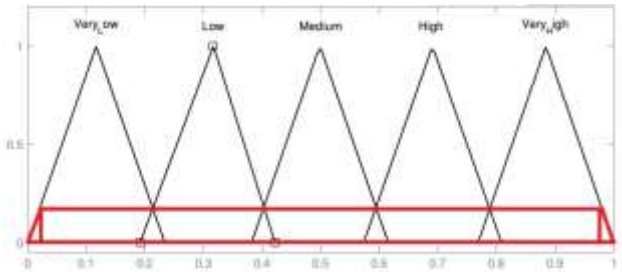

Fig. 11. Defuzzification of Software Testability

It is the most accurate technique for defuzzification and formula is

$$x^* = \frac{\sum A\bar{x}}{\sum A} \quad (1)$$

x*=0.0833 / 0.1666

x*=0.4997

Where x* is the defuzzified output, A shows the area of segment and $\bar{x}$ is the corresponding centroid. The testability for the inputs appeared above is 0.499 that is same as evaluated above. Hence our proposed fuzzy model is validated against the COG method by defuzzification and same result has been achieved for software testability.

VII. Conclusion and Future Prospects

Present work adopts fuzzy inference system for the assessment of testability of AO Software. It is less dependent on historical data and fuzzy model can be built little data which is the major advantage of this approach. Fuzzy helps in automate the process of identifying the testability using metrics. The result shows that suggested model can also be used to predict testability of aspect-oriented software system, which helps in reducing efforts needed in development and improve the quality of the system.

Software Testability is widely used now a day. It has great potential to improve and maintain software systems. In future prospects, Neural Network, Support Vector Machine may be used to predict testability. The main aim of this paper is to estimate testability for AO software because it can help in reducing the maintenance efforts and cost too. Software professionals may also use this approach to measure testability of AO software and forecast the testing and maintenance efforts, cost too.

Acknowledgment

We would like to thank the faculty of Amity University for helping us in refining the objective and Amity University for providing us research environment and facilities. We also like to extend our thanks to Dr. Arun Sharma, Professor, CSE, K.I.E.T. Ghaziabad for his valuable suggestions.

REFERENCES

[1] Bruntink Magiel, Deursen Arie Van, "Predicting Class Testability using Object-Oriented Metrics", Published in Proceedings 4th IEEE International Workshop on Source Code Analysis and Manipulation, pp. 136-145, Chicago, IL, US, September 2004.

[2] ISO 9126, International Organization for standardization.

[3] IEEE Standard Glossary of Software Engineering Terminology, IEEE, 1990.

[4] Mary Jean Harrold, "Testing: a Roadmap", Published in Proceeding's of the Conference on The Future of Software Engineering, pp. 61-72, Limerick, Ireland, 2000.

[5] Voas Jeffrey M., Miller Keith W., "Improving the Software Development Process using Testability Research", at NASA-Langley Research Center, pp. 114-121, IEEE Software, 1992.

[6] Jungmayr Stefen, "Reviewing Software Artifacts for Testability", at FernUniversität Hagen, Praktische Informatik III, Feithstrasse 142, D-58084 Hagen, EuroSTAR, Barcelona, Spain, 1999.

[7] SantAnna C., Garcia A., Chavez C., Lucena C., Staa A. von, "On the Reuse and Maintenance of Aspect Oriented Software: An Assessment Framework", Published in Proceedings of XVII Brazilian Symposium on Software Engineering, PUC-Rio, Computer Science Department, TecComm, 2003.

[8] Kiczales G., Hilsdale E., Hugunin J., Kersten M., Palm J., Griswold W.G., "An Overview of AspectJ". Published in Proceedings of the 15th European Conference on Object Oriented Programming, Springer, Heidelberg, Berlin, pp. 327–353, 2001.

[9] Santos R.P., Costa H.A.X., Parreira Júnior P.A., Amâncio A.F., Resende A.M.P., Werner C.M.L., "An Approach Based on Maintainability Criteria for Building Aspect-Oriented Software Implementation Model", INFOCOMP Journal of Computer Science, Special Edition, pp. 11-20, 2009.

[10] Zhao Jianjun, "Measuring Coupling in Aspect Oriented Systems", Published in Proceedings 10th International Software Metrics Symposium, Information Processing Society, Japan, 2004.

[11] Zhao Jianjun, Xu Baowen, "Measuring Aspect Cohesion", Published in Proceedings of International Conference on Fundamental Approaches to Software Engineering, pp.54-68, Springer-Verlag, Barcelona, Spain, 2004.

[12] Ceccato Mariano, Tonella Paolo, "Measuring the Effects of Software Aspectization", Published in WCRE: 1st Workshop on Aspect Reverse Engineering, 2004.

[13] Chidamber S.R., Kemerer C.F., "A Metrics Suite for Object Oriented Design", Software Engineering, IEEE Transactions, Vol. 20, No. 6, 476–493, 1994.

[14] Zhang Charles, Jacobsen Hans-Arno, "Quantifying Aspects in Middleware Platforms in AOSD", Published in Proceedings of the 2nd International Conference on Aspect Oriented Software Development, ACM Press, pp. 130–139, New York, NY, USA, 2003.

[15] Tsang Shiu Lun, Clarke Siobhán, Baniassad Elisa, "Object Metrics for Aspect Systems: Limiting Empirical Inference based on Modularity", Technical report, Distributed Systems Group, Dublin, Ireland, 2000.

[16] Mulo Emmanuel, "Design for Testability in Software Systems", Master's Thesis, submitted to Delft University of Technology, Netherland, 2007.

[17] Wang Yingxu, "Design for Test and Software Testability", University of Calgary, 2003.

[18] Pan Nankai, Song Eunjee, "An Aspect-oriented Testability Framework", ACM, RACS'12, San Antonio, TX, USA, 2012.

[19] Bach James, "Heuristics of Software Testability", Satisfice, Inc., Version 2.2, 2013.

[20] Basili V., Briand L., Melo W., "A Validation of Object-Oriented Design Metrics as Quality Indicators", IEEE Transactions on Software Engineering, Vol. 22 No. 10, pp. 751-761, 1996.

[21] Khan R.A., Mustafa K., "Metric based Testability Model for Object-Oriented Design (MTMOOD)", ACM SIGSOFT Software Engineering Notes, Vol. 34, No. 2, 2009.

[22] Shaheen Muhammad Rabee, Bousquet Lydie du, "Survey of Source Code Metrics for Evaluating Testability of Object-Oriented Systems", ACM Transactions on Computational Logic, 2010.

[23] Nazir Mohd, Khan Raees A., "Software Design Testability Factors: A New Perspective", Published in Proceedings of the 3rd National Conference on Computing for Nation Development, 2009.

[24] Abdullah, Srivastava Reena, Khan M.H., "Testability Estimation of Object Oriented Design: A Revisit", IJARCCE, Vol. 2, Issue 8, 2013.

[25] Binder Robert V., "Design for testability in object-oriented systems," Communications of the ACM, vol. 37, No. 9, pp. 87-101, 1994.

[26] Bruce W.N. Lo, Shi Haifeng, "A Preliminary Testability Model for Object-Oriented Software", published in Proceeding of International Conference on Software Engineering, IEEE, Education Practice, pp. 330-337, 1998.

[27] Morris Stephen B., "Design Patterns in Java: The Observer", Pearson Education, InformIT, 2007.

[28] Jungmayr Stefen, "Testability during Design", Published in Proceedings of the GI Working Group Test, Analysis and Verification of Software, Softwaretechnik-Trends, Potsdam, pp. 10-11, 2002.

[29] Gao Jerry, Shih Ming-Chih, "A Component Testability Model for Verification and Measurement", published in Proceedings of the 29th Annual International Computer Software and Applications Conference, IEEE Computer Society, pp. 211–218, 2005.

[30] Voas Jeffrey M., Miller Keith W., "Software Testability: The New Verification," IEEE Software, vol. 12, pp. 17-28, 1995.

[31] Bache Richard, Mullerburg Monika, "Measures of Testability as a basis for Quality Assurance", Software Engineering Journal, vol. 5, no.2, pp. 86-92, 1990.

[32] P.K. Singh, O.P. Sangwan, "Aspect Oriented Software Metrics Based Maintainability Assessment: Framework and Model", published in proceeding's of Confluence-2013, The Next Generation Information Technology Submit, 26th -27th September, Amity University, Noida, India 2013.

[33] Bruntink Magiel, Deursen Arie van, "An Empirical Study into Class Testability," Journal of Systems and Software, vol. 79, pp. 1219-32, 2006.

[34] Freedman Roy S., "Testability of Software Components", IEEE Transactions on Software Engineering, vol. 17, no.6, pp.553-564, 1991.

[35] Boxall M.A.S., Araban S., "Interface Metrics for Reusability Analysis of Components", Published in Proceedings of Australian Software Engineering Conference, pp. 40-46, Melbourne, Australia, 2004.

[36] P.K. Singh, O.P. Sangwan, Arun Sharma, "A Systematic Review on Fault Based Mutation Testing Techniques and Tools for Aspect-J Programs", published in proceeding's of 3rd IEEE International Advance Computing Conference,

IACC-2013 at AKGEC Ghaziabad, India, February 22-23, 2013.
[37] Piveta Eduardo Kessler, Moreira Ana, Pimenta Marcelo Soares, Araújo João, Guerreiro Pedro, Price R. Tom, "An empirical study of aspect-oriented metrics", Journal of ELSEVIER, Science of Computer Programming 78, pp. 117–144, 2012.
[38] Malla Prakash, Gurung Bhupendra, "Adaptation of Software Testability Concept for Test Suite Generation", Phd Thesis Submitted to School of Computing Blekinge Institute of Technology, SE-37179, Karlskrona, Sweden, 2012.
[39] Jungmayr Stefen, "Improving Testability of Object Oriented Software", Dissertation.de-Verlag im Internet GmbH, Berlin, 2004.
[40] Bruntink Magiel, "Testability of Object-Oriented Systems: a Metrics-based Approach", Master Thesis Submitted to University of Amsterdam, Software improvement group, 2003.
[41] Nazir Mohd, Khan Raees A., Mustafa Khurram, "A Metrics Based Model for Understandability Quantification", Journal of Computing, volume 2, issue 4, 2010.
[42] Kumar Avadhesh, "Analysis and Design of Metrics for Aspect-Oriented Systems", Phd Thesis Submitted to School of Mathematics and Computer Applications, Thapar University, Patiala, Punjab, India, 2010.
[43] Burrows Rachel, Ferrari Fabiano Cutigi, Garcia Alessandro, Taïani François, "An Empirical Evaluation of Coupling Metrics on Aspect-Oriented Programs", ACM, WETSoM, Cape Town, South Africa, 2010.
[44] Saraiva Juliana, Barreiros Emanoel, Almeida Adauto, Lima Flávio, Alencar Aline, Lima Gustavo, Soares Sergio, Castor Fernando, "Aspect-Oriented Software Maintenance Metrics: A Systematic Mapping Study", Published in Proceedings of the EASE - Published by the IET, 2012.
[45] Huang Rui, Li Mingyu, Li Zhang, "Research of Improving the Quality of the Object-Oriented System", International Journal of Information and Education Technology, Vol. 3, No. 4, 2013.
[46] Aopmetrics project. http://aopmetrics.tigris.org/
[47] Coady Y., Kiczales G., "Back to the Future: A Retroactive Study of Aspect Evolution in Operating System Code", Published in proceedings of the 2nd International Conference on Aspect Oriented Software Development, ACM Press, pp. 50–59, New York, NY,USA, 2003.
[48] Fenton N.E., Pfleeger S.L., "Software Metrics: A Rigorous and Practical Approach", Published by Course Technology, 1998.
[49] Sivanandam, S.N., Sumathi, S., Deepa, S.N., "Introduction to Fuzzy Logic using MATLAB", Springer, Heidel-berg, 2007.
[50] MacDonell S.G., Gray A.R., Calvert J.M., "Fuzzy Logic for Software Metric Practitioners and Researchers", Published in Proceedings of the 6th International Conference on Neural Information Processing ICONIP, Perth, pp. 308-313, 1999.
[51] Ryder J., "Fuzzy Modeling of Software Effort Prediction", Published in Proceedings of IEEE Information Technology Conference, pp. 53-56, Syracuse, New York, 1998.
[52] http://www.mathworks.in/help/fuzzy/building-systems-with-fuzzy-logic-toolbox-software.html (last accessed on 12/02/14).
[53] P.K. Singh, O. P. Sangwan, A. Pratap, A. P. Singh, An Analysis on Software Testability and Security in Context of Object and Aspect Oriented Software Development, International Journal of Security and Cybercrime, Romania, Vol. 3, Issue 1, pp. 17-28, 2014.

Authors' Profile

Pradeep Kumar Singh is M.Tech (CSE) from GGSIPU Delhi and pursuing his PhD. from Gautam Buddha University, Greater Noida, India. Currently, he is working as Assistant Professor in Amity University Uttar Pradesh, Noida, India. He is member of ACM, CSI and many professional bodies. He has published 10 papers in International Conferences and Journals of repute. His major area of Interest includes Software Engineering, Object Oriented Software Engineering, Aspect Oriented Software Engineering.

Dr. Om Prakash Sangwan is M.Tech (CSE) and PhD. Currently, he is associated with Gautam Buddha University, Greater Noida. Uttar Pradesh, India. He is Senior Member of ACM, CSI, IEEE and many professional bodies. He has filled two Patents and published 35 papers in International Journals. His major area of Interest includes Software Engineering, Aspect Oriented Software Engineering, Soft Computing.

Amar Pal Singh is a M.Tech.(CSE) student from Amity University Uttar Pradesh, Noida, India. His interests include Software Engineering, Soft Computing Techniques.

Amrendra Singh is a student of M.Tech. (CSE) from Amity University Uttar Pradesh, Noida, India. His interests include Software Engineering, Soft Computing Techniques.

How to cite this paper: Pradeep Kumar Singh, Om Prakash Sangwan, Amar Pal Singh, Amrendra Pratap,"An Assessment of Software Testability using Fuzzy Logic Technique for Aspect-Oriented Software", International Journal of Information Technology and Computer Science(IJITCS), vol.7, no.3, pp.18-26, 2015. DOI: 10.5815/ijitcs.2015.03.03