Software Testing Estimation using Soft Computing Techniques

Amarpal Singh, Piyush Saxena, Abhishek Singhal

Abstract— Software development is an extremely composite plus brainstorming action. In previous days programmers wrote programs by means of machine language in which they exhausted their more time in thinking about an exacting machine's instructions rather than the solution of the problem in their hands. Progressively, program developers switched to advanced stage of programming languages (high-level languages). Software testing is an imperative attribute of software quality. However the prediction of this attribute is a cumbersome process. Therefore various methodologies are proposed so far to estimate the testing time of software. Among them Fuzzy Inference System (FIS) and Adaptive Neuro- Fuzzy Inference System (ANFIS) is one of the sophisticated methods which have immense prediction capability and this paper explores its application to evaluate testing time of the aspect-oriented system. Prediction of testing time is performed by FIS and ANFIS. The results obtained from the current study are compared with adaptive neuro- fuzzy inference system and it is revealed that which model is more useful.

Index Terms— Module oriented approach (MOA), Aspect oriented software approach (AOSA), Object Oriented Approach(OOA), Fuzzy Inference System (FIS), Adaptive Neuro-Fuzzy Inference System (ANFIS)

I. INTRODUCTION

The brisk enlargement of software products in dimension and complication has drawn the consideration of researchers to be more decisive on quality assessment. Several reliability calculation approaches such as classical statistical methods (Linear and Logistic Regression) and modern machine learning methods such as Artificial Neural Network (ANN), Support Vector Machine (SVM), Decision Tree (DT), Group Method of Data Handling (GMDH), Genetic Algorithms (GAs), Fuzzy Inference System (FIS), Adaptive Neuro-Fuzzy Inference System (ANFIS), Gene Expression Programming (GEP) proposed[1,2,3,4,5,6,7].

There are various ways to expand the software using the high level languages. The growth of software is also depends on type of application to be developed .

The Module-Oriented Approach (MOA) [13] is the leading approach for the software development. Modular programming is a process which is based upon the concept of procedure (subroutine, methods and functions). Any procedure can be executed during program's execution.

Modular programming is an improved option rather than simple, conventional, sequential or unstructured programming in several circumstances. MOA engage excessive complexity or which necessitate momentous straightforwardness towards maintainability.

Manuscript Received July, 2013.

Amarpal Singh, Masters of Technology, CS&E Amity school of Engineering Technology, Amity University Uttar Pradesh, Noida, India.

Piyush Saxena, Masters of Technology, CS&E Amity school of Engineering Technology, Amity University Uttar Pradesh, Noida, India .

Abhishek Singhal, Assistant Professor, CS&E Department, Amity school of Engineering Technology, Amity University Uttar Pradesh, Noida, India.

The Important Features and limitation of MOA are as follows:

1. The ability to recycle and reuse the code at different place of the program without bootlegging it.
2. It is an easier way to retain the path of code flow than 'go to' or 'jump' statement.
3. A method that is highly modular than configuration.
4. To enhance the features of any module where new code is obligatory to write.
5. It does not give a proper way to compile the data along with their operations.

Progressively, complexity grow we necessitate enhanced methods. Object-Oriented Approach (OOA) is way to remove the obstruction that arises in MOA.

Today, OOA attains its position and become the typical programming concept for real world applications. OOA helps to disintegrate the objects into its abstract behaviour. Object-Oriented Programming (OOP) consists of various fundamental concepts like modularity, encapsulation, inheritance and polymorphism.

Some Important features of OOA [13] are as following:

1. Pressure on data relative as compared to procedure.
2. It allows the system to interrelate with objects.
3. It allows reusable codes via inheritance.
4. It allows message passing b/w functions via objects.
5. It allows elasticity for adding new data and functions wherever required.
6. It allows the idea of polymorphism to employ universal performance of objects and boundary for related concepts.
7. Preservation of large side project, along with strewn various objects.

Aspect Oriented Software Development (AOSD) is a new methodology of splitting up of concerns in software development. The process of AOSD promises to modularize coding in a system. Similar objects in Object-Oriented Soft. Development may happen at any stage of software lifecycle, requirement specification, design and implementations etc.

Some examples of crosscutting concerns are synchronization, logging, exception handling and resource sharing. Programming codes cannot be efficiently encapsulated in terms of modules or objects but must be scattered throughout the code which are the restriction of OOA.

An "aspect weaver" takes the aspects and the core modules and composes the concluding system.

II. ASPECT ORIENTED PROGRAMMING LANGUAGE

The aspect-oriented approach to programming not only is applicable for object oriented systems but also for module-oriented systems.

1 AspectJ

Out of the existing AOP languages, AspectJ is the most admired and mainly used in research areas. AspectJ is a straightforward expansion of Java.

Some significant quality of AspectJ is as follow:

1. provides definition of new constructors,
2. Support for modular implementation of crosscutting concerns.
3. Enables plug-and-play implementations of crosscutting concerns such as synchronization, consistency checking, protocol management and others.

2 CaesarJ

CaesarJ is an aspect-oriented language which is known for reusability.

Some significant quality of CaesarJ is as follow:

1. It wires some more features like *virtual classes*, *mix-in composition*, *aspectual polymorphism*, and *bindings*.
2. It combines the aspect-oriented constructs, point cut.
3. It provides the mechanisms for advanced object-oriented modularization.

3 Hyper/J

Hyper/J urbanized by IBM, is also appropriate and admired as the part of AOP languages. While by means of Hyper/J, a program developer initiates with three inputs:

1. *hyperspace* file that describes the Java class files that can be manipulated by Hyper/J,
2. *concern mapping* file that describes which pieces of the Java source map to each dimension of concern, and
3. *hypermodule* file that describes which dimensions of concern should be incorporated (i.e., which *hyperslices*) and how that incorporation should proceed.

III. LITERATURE SURVEY

Numerous prediction methods such as LR, ANNs, SVM, DT, FIS and ANFIS, are used for predicting software reliability prediction effectively. Even though, these methods are infrequently applied for assessing testing time based on past failure behaviour. Khoshgoftar et al. [8] made an ample study in the area of connectionist models to provide the software reliability prediction.

They also introduced a loom for static reliability modelling. He founded that neural network models are capable for providing the better quality of fit and high accuracy. Several other empirical studies based on multivariate linear regression and neural network methods have been carried out for prediction of software reliability growth trends.

There are the following methods for predicting software testing time based on different severity level:

(i) Linear regression (LR):

It is a process that is used for analysing dependent variable from the set of self-governing variables. On using uni-variate linear regression process for showing relations b/w dependent variables and each self-governing variable.

(ii) *Radial basis function network (RBFN):*

Radial basis function neural network is a method for approximating software testing time using Gaussian activation function. The structural design of radial basis function neural network divides into three layers, i.e. Input layer, hidden layer and output layer.

(iii) Generalized regression neural network (GRNN):

Generalized Regression Neural Network (GRNN) is a method for assessing software testing time based on diverse severity stage of errors. The GRNN architecture has one radial basis layer and a special linear layer used for function approximation with sufficient number of hidden neurons.

(iv) Support Vector Machine (SVM):

It is a learning process that constructs an N-dimensional hyper plane that differentiates data in 2 categories. The basic use of SVM modelling is to separate groups of vectors in a way that case with one group of dependent variable on one part of the plane and the cases of the self-governed variables on the other side of the plane.

(v) Decision Tree (DT):

Decision trees (DTs) are used as a predictive model which maps observations of a variable to the target's value. DT's are used as a 'predictive machine – learning model' that is used to decide the average time to failure (MTTF). This is the final value as a dependent variable of a fresh sample on the attributes of independent variables [10].

(i) Fuzzy Inference System (FIS) :

The Fuzzy Inference System is used to deploy the proposed model. Fuzzy inference system consists of five components namely data base, rule base, fuzzification, defuzzification and decision making. Fuzzy works on both mamdani and sugeno type systems.

(ii) Adaptive Neuro-Fuzzy Inference System (ANFIS):

Employ of adaptive Neuro-FIS for assessing software testing time based on different severity stage of errors. In fuzzy inference system the partisanship function is accepted randomly or used as constant value. But, in case of ANFIS the partisanship function and connected attributes can be selected automatically that results in improved forecast of correctness of the process.

IV. METRICS SUITE FOR GENERIC ASPECT ORIENTED SYSTEM

This section of the paper describes the metrics complexity of generic AOS, it is essential to begin with a appropriate harmonized, clear and functioning terminology and formalism.

An AO system (S) consists of a set of three factors, (C(S)).they are namely set of characteristics (Att(c)), a set of operations (Op(c)) and a set of nested factors (Nested(c)).

The set of members of a factor *c* is defined by following equation [14].

M(*c*) = *Att(c)* ◻ *Op(c)* ◻ *Nested(c)*.

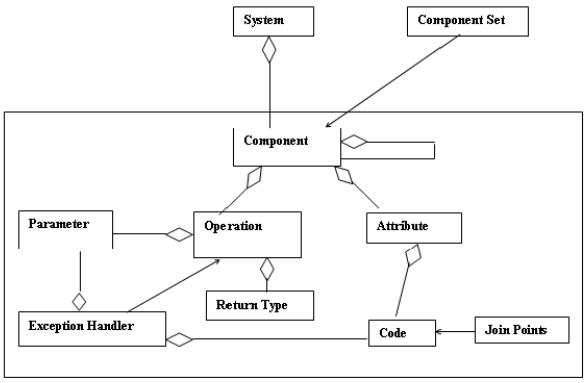

Fig 1: Factors of an Aspect-Oriented System

Definition: *Built-in types*, *user-defined types* and *factor types*

- BT(S): set of built-In types for structure S.
- UDT(S): set of user defined types of structure S.
- CT(S): set of factor types of the structure S.
- T(S): set of all available types
- T(S) = BT(S) * UDT(S) * CT(S)

Complexity of AO system falls on two major factors:

(i) code complexity of Component Set (C(S))

(ii) Complexity of interactions between the factors (IC(S)) of Component Set.

a. complexity of operation invocation (CMPXOI(c))

b. complexity of characteristic references (CMPXAR(c))

c. Complexity of statements (CMPXSTMT(c)).

1. Complexity of AO system [14] :

$$CMPXAOS = CMPXC(s) + CMPXIC(S)$$

Here,
CMPXAOS =complexity of AO system,
CMPXC(*s*) = the code complexity of Factor Set
CMPXIC(*S*) = interaction complexity of the Factor Set.

2. Code Complexity of Factor Set [14]**:**

$$CMPX_{AOS} = CMPX_{c(s)} + CMPX_{IC(S)}$$

Here,
X = sum of no. of factors set
CMPXM(*cx*)= code complexity of factor *cx* .

3. Code Complexity of a Factor [14]**:**

$$CMPX_{M(C)} = \alpha * CMPX_{Att(c)} + \beta * CMPX_{op(c)} + \gamma * CMPX_{Nested(c)}$$

Here,
α, β, γ = coefficients for *CMPXAtt*(*c*) ,*CMPXOp*(*c*) and *CMPXNested* (*c*).
CMPXAtt(*c*) = complexity of characteristics in factor *c*,
CMPXOp(*c*) = complexity of operations in factor *c*,
CMPXNested (*c*) = complexity of nested factors in factor *c*.

4. Complexity of Characteristics [14]**:**

$$CMPX_{Att\,(c)} = \sum_{i=1}^{L} w_i * Att_i(c)$$

Here,
L = sum of no. of characteristics in factor *c*
wl= equivalent weight assessment of characteristic (*Attl*(*c*)).

5. Complexity of Operations [14]**:**

$$CMPX_{Op\,(c)} = \sum_{m=1}^{M} w_m * Op_m\,(c)$$

Here,
M=total no. of operations in factor *c*
wm = equivalent weight assessment for operation *Opm*(*c*).

6. Complexity of Nested Factors [14]**:**

$$CMPX_{Nested\,(c)} = \sum_{n=1}^{N} w_n * Nested_n\,(c)$$

Here,
N = total no. of nested factors in *c*,
wn=equivalent wt. assessment of nested factor *Nestedn*(*c*).
finally, we calculated ***Code Complexity of a Factor*** as:

$$CMPX_{M(c)} = \alpha * \sum_{i=1}^{L} w_i * Att_i\,(c) + \beta * \sum_{m=1}^{M} w_m * Op_n\,(c) + \gamma * \sum_{n=1}^{N} w_n * Nested_n\,(c)$$

7. Code Complexity of Factor Set [14]:

$$CMPX_{M(c)} = \sum_{x=1}^{x} (\alpha * \sum_{i=1}^{L} w_i * Att_i\,(c) + \beta * \sum_{m=1}^{M} w_m * Op_n\,(c) + \gamma * \sum_{n=1}^{N} w_n * Nested_n\,(c))$$

Complexity of Interactions among Factors [14]***:***

CMPXIC(S)=CMPXOI(S)*CMPXAR(S)*CPMXSTMT(S)

Here,
CMPXIC(*S*) = complexity of interactions between the factors of system *S*,
CMPXOI (*S*) = complexity of operation creations in factors of system *S*,
CMPXAR(*S*) = complexity of characteristic references in factors of system *S* and
CPMXSTMT (*S*) = complexity of statements in workings of system *S*.

Complexity of Operation Invocation [14]***:***

$$CMPXOI(S) = \sum_{x=1}^{X}\Big(\sum_{d=1}^{D} w_d * IOI_{ST_{d(o)}}(c_x) + \sum_{e=1}^{E} w_c * IOI_{DYN_c}(c_x) + \sum_{f=1}^{F} w_f * EOI_{ST_{f(o,o')}(c_x)} + \sum_{g=1}^{G} w_g * EOI_{ID_g(o)}(c_x)\Big)$$

Here X = total no. of workings in system S ,
D = total no. of hidden operation creations. These operation can completely statically evaluated in factor cx ,
E = entire no. of hidden operation creations. These operation can only dynamically evaluated in factor cx ,
F = entire no. of explicit operation creations. These operation can completely statically evaluated in factor cx and
G = entire no. of explicit operation creations, i.e. for implementing family of the operations factor in factor cx .
wd = equivalent weight assessment of $IOISTd(o)(cx)$, we is the equivalent weight assessment of $IOIDYNe(o)(cx)$,
wf = equivalent weight assessment of $EOISTf\ (o, o')(cx)$ and
wg =equivalent weight assessment of $EOIIDg(cx)$.

Complexity of Characteristic References [14]***:***

$$CMPXAR(S) = \sum_{x=1}^{X}\Big(\sum_{H=1}^{H} w_k * AR_{ST_h}(c_x) + \sum_{i=1}^{I} w_i * AR_{ID_i}(c_x) + \sum_{j=1}^{J} w_j * AR_{FI_j(c_x)} + \sum_{k=1}^{K} w_k * AR_{ST_k}(c_x, c_x')\Big)$$

Here,
X = total no. of factors in system S,
H = total no. of characteristic references, accounting only for statically determined feature in factor cx ,
J = total no. of characteristic references, accounting only for the first statically determined implementation,
K = total no. of define sets with a meta variable representing the target factor in factor cx .
wh = equivalent weight assessment of $AR_{ST_h}(c_x)$,
wi = equivalent weight assessment of $AR_{ID_i}(cx)$,
wj = equivalent weight assessment of $AR_{FI_j(c_x)}$ and
wk = equivalent weight assessment of $AR_{ST_k}(c_x, c_x')$

Complexity of Statements [14]***:***

$$CMPX_{STMT(S)} = \sum_{x=1}^{X}\Big(\sum_{p=1}^{P} w_p * Ex_{C_p}(c_x) + \sum_{q=1}^{Q} w_q * ArrC_q(c_x) + \sum_{r=1}^{R} w_r * TypeC_r(C_x)\Big)$$

Here,
X = total no. of factors in system S ,
P = total no. of exception occurred in factor cx ,
Q=total no. of array creations in factor cx , and
R = total no. of type casting in factor cx .
wp = equivalent weight assessment of $Excp(cx)$,
wq = equivalent weight assessment of $ArrCq(cx)$, and
wr = equivalent weight assessment of $TypeCr(cx)$.

V. RESEARCH METHODOLOGY

The research methodology is adopted to estimate software testing time using soft computing method is as follows.

1. Fuzzy Inference System (FIS)

We analyse the performance of the proposed system by the use of fuzzy inference system. Fuzzy inference system is akin to a neural network type configuration that is able to map inputs via input membership functions and affiliated attributes.
As It is feasible to use Fuzzy Logic Toolbox [9] via functioning rigorously via the command line. It's easy to make the system with Graphics User Interface (GUI). There are 5 important tools for constructing, editing, and observing fuzzy inference systems in Fuzzy Logic Toolbox. These are:

- Fuzzy Inference System (FIS) Editor
- Membership Function Editor
- Rule Editor
- Rule Viewer
- Surface Editor

2. Adaptive Neuro Fuzzy Interference System (ANFIS)

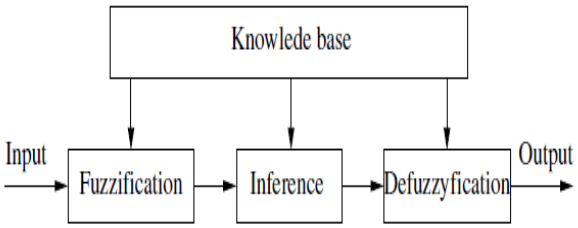

Fig 2: Flow chart for ANFIS

The FIS is a limitation that maps:

- Input characteristics to input membership functions,
- Input membership functions to rules,
- Rules to set of output characteristics,
- Output characteristics to output membership functions,
- Output membership functions to a solitary evaluation output
- Result concurrent with output

3. ANFIS Model Learning and Inference

- The Neuro-Adaptive learning process provides a technique for the fuzzy modelling exercise for study of data sets, in instruct to calculate the membership function attributes which permit the best fuzzy inference system to follow the provided input or output data [11].
- It permits the fuzzy system to adapt and study from the information and data it model.

4. FIS Configuration and Parameter Adjustment

- Network type config. analogous to neural network maps inputs via partisanship functions and linked attributes, & then through output partisanship functions and allied attributes, to be used to analyse input and output.

- The attributes connected with the partisanship functions are used to adapt by means of a training process/ method.
- The analysis and calculation of these attributes or their adjustments will be assisted by a ramp vector and that provides an analysis of how well the fuzzy inference system has modelled the input and output data for a provided set of attributes.
- Any of the various optimization techniques can be applied to adjust the parameter as to reduce the error measurement.
- Anfis uses backpropogation or least mean squares estimation and backpropogation.
- When the gradient vector is achieved, one of the optimization techniques is applied for adjusting the attributes so as to reduce error measurement.
- Anfis uses backpropogation method or least square estimation with back propagation for membership function estimation.

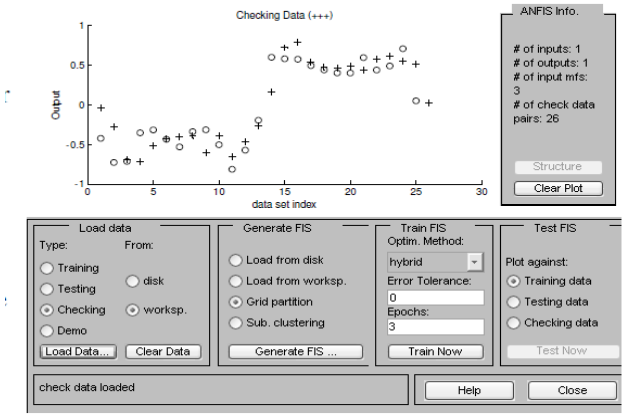

Fig 3: ANFIS editor

5. Automatic FIS Configuration Generation with ANFIS

• Partition method: grid partitioning (default) or subtractive clustering.

• For generating FIS choose first the no. of partisanship functions, MFs, then the type of input and output partisanship functions.

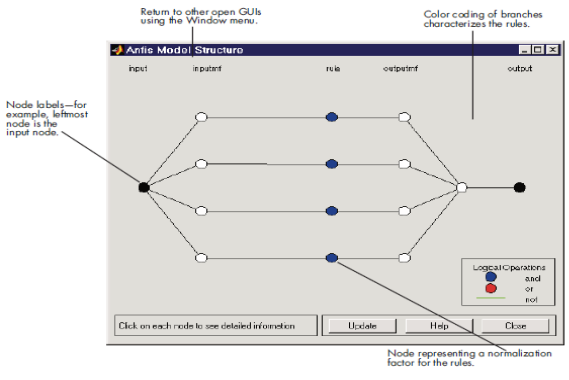

Fig 4: FIS configuration

We can inspect the construction of the resulting FIS:

6. ANFIS Training

- The two 'anfis' parameter optimization process options offered for FIS training are hybrid.
- Error Tolerance is used to generate a training stopping measure, i.e. related to the error size.
- Running training for 40 epochs gives the error results.
- Notice the checking error decreases up to a certain point in the training and then it increases.
- This increase represents the point of model over fitting.
- 'anfis' chooses the model parameters associated with the minimum checking error (just prior to this jump point).

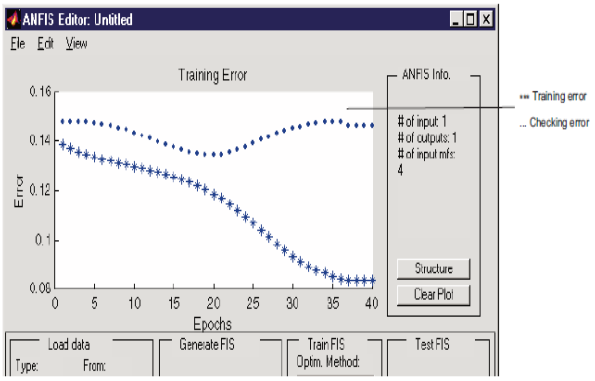

Fig 5: ANFIS Training

- An example for the assessment data opportunity of 'anfis' is helpful.

Testing the FIS in resistance to the checking data gives the following results.

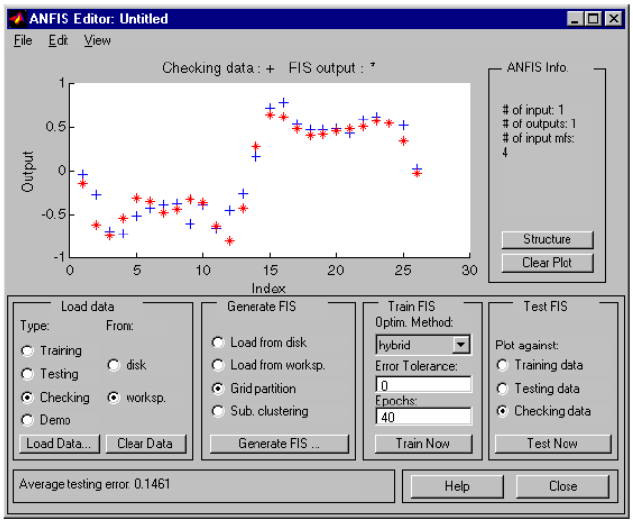

Fig 6: Checking Data.

7. Evaluation Criteria

To analyse the outcomes, we analyse the efficiency of proposition form with the use of different attributes as follows:

$$MARE = \left(\sum_{i=1}^{n} \left| \frac{estimate - actual}{actual} \right| \right) \% \, n$$

Mean absolute error is a amount use to measure how near prediction are to the actual outcomes. MAE guess the network output for each observation to estimate whether the projected method is subjective and tend to over/under estimate. P is the predicted assessment from the model and A is the actual observed assessment.

$$RMSE = \sqrt{\sum_{i=1}^{N}(P-A)^2 \ /n}$$

VI. A FUZZY LOGIC APPROACH TO COMPLEXITY METRICS

Our fuzzy model for integrating AO factor complexity CMPXM(C) accounts the effect of complexity of characteristics CMPXAtt(c), complexity of operation CMPXOp(c) and complexity of nested factors CMPXNested (c)[12]. The Block diag. for fuzzy representation id shown.

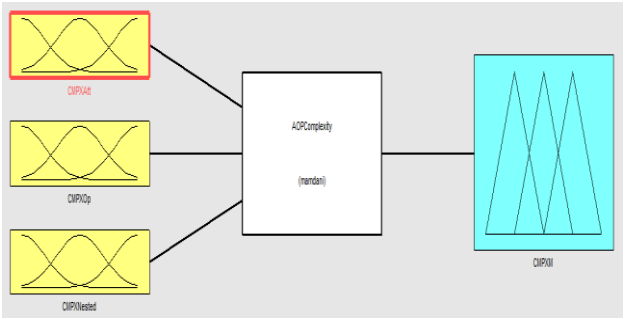

Fig 7: Fuzzy Model for Complexity Measurement of a Factor [12]**.**

The fuzzy model contains the 4 models:

1. The fuzzy model is the first step in execution of any fuzzy model that changes crisp values to fuzzy assessments.
2. In the next phase the assessments are analysed in fuzzy field by interface engine on the production rules i.e. knowledge base that is given by the domain experts. In this phase fuzzy operators are applied.
3. In the third phase insinuation process is applied and all the outputs are averaged.
4. In the final phase, the processed outputs are converted to crisp values via defuzzification method.

Membership Functions For Input Parameters

Complexity of factor *CMPXM* (*C*) have been taken in the scale of 0 to 1 and member functions as Null, Extremely Low, Low 'L', Medium 'M', High 'H', Extremely High. Because nested factor is also a factor, member function of nested factor will be same as of a factor *i.e.* NIL, VL, L, M, H and VH.

Complexity of nested factor *CMPXNested* (*c*) can be evaluated recursively with terminal condition as the factor is without nested factor *i.e. CMPXNested* (*c*) as NIL for that factor. For simplification, *CMPXAtt*(*c*) and *CMPXOp*(*c*) assessments have also been accepted in between 0 and 1. For *CMPXAtt*(*c*) and *CMPXOp*(*c*) , member functions have been considered as NIL, L, M and H.

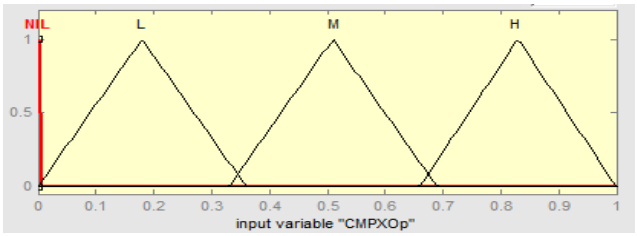

Fig 8: Member Functions of Input Variable CMPXOp

VII. FUZZY RULES FOR PROJECTED MODELS

We have used fuzzy logic and have designed 96 fuzzy rules (4 member functions of *CMPXAtt*(*c*) *4 member functions of *CMPXOp*(*c*) *6 member functions of *CMPXNested* (*c*)). Here, mamdani method for defining fuzzy rules issued, which is used for nonlinear equations.

These rules are designed on the basis of experience and expertise knowledge of the field that's why these are also known as knowledge base. For sample, some of the rules are listed in Table 5.4. First column labeled Rule# represent rule no., second column is for input linguistic variables, *CMPXAtt(c) ,CMPXOp(c) and CMPXNested (c) and third column is for output linguistic variable CMPXM (C)* .

Table 4: Some Sample Rules of the Complexity Fuzzy Model

Rule#	Input Variables			Output Variable
	$CMPX_{Att(c)}$	$CMPX_{Op(c)}$	$CMPX_{Nested(c)}$	$CMPX_{Op(c)}$
1	NIL	NIL	NIL	NIL
12	NIL	NIL	VH	VH
25	L	NIL	NIL	VL
37	L	M	NIL	M
45	L	H	L	H
57	M	L	L	M
65	M	M	H	VH
93	H	H	NIL	H

Using projected methodology and model, complexity of all the factors of the software system can be measured and can evaluate average of these complexity assessments. The average complexity assessment will be between 0 and 1 and will fall in any of the categories, VL, L, M, H and VH. With the help of this assessment, we can specify complexity level of AO system. Three dimensional surface view of this rule base is in Fig 9.

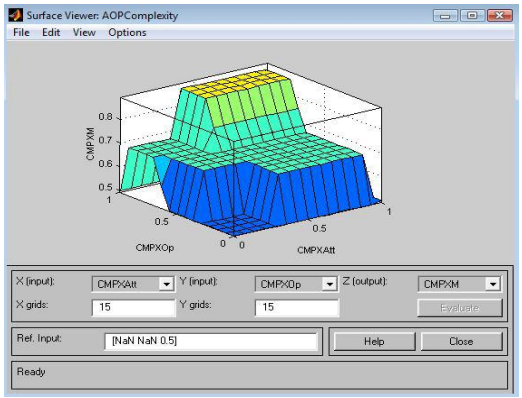

Fig 9: Surface Viewer of AOP Complexity

VIII. CALCULATIONS

Table 5: complexity of factors

$CMPX_{att(c)}$	**5.63**
$CMPX_{op(c)}$	**5.0**
CMPX(nested)	**5.25**
Code complexity of factor:	**10.4**

Table 6: Code complexity of factor set

Code complexity of factor set	
For **nil:**	**3.9**
For **Very Low:**	**3.9**
For **Low:**	**3.9**

For **Medium:**	**5.5**
For **High:**	**11.9**
For **Very High:**	**15.68**

Table 7: Error Observed

Error Observed	
Mean Absolute Error	**5.23**
Root Mean Square Error	**30.50**

IX. CONCLUSION AND FUTURE SCOPE

We have used fuzzy logic for defining software complexity metrics as linguistic variables and for the software testing time. Motivation of applying fuzzy logic is due to difficulties faced to get total complexity of a factor in generic aspect-oriented system because factors, which contribute in the complexity of a factor, are different in nature and have different type of complexity assessment. Using common terminology, formalism and generic/unified framework new complexity metrics have been defined. These metrics are defined for measuring code complexity and interaction complexity of AO system. Software testing time is only code complexity has been evaluated. A fuzzy model has been defined to measure code complexity of a factor. Mean Absolute Error (MARE), Root Mean Square Error (RMSE) and Average complexity of all the factors available in the AO software system will be indicator of the complexity level of the system. Using this model, complexity of software developed in most of the AO languages can be measured, which further may be used as an indicator to external software quality such as *maintainability*, *reusability*, *adaptability* and *understand ability*. In future work, by applying similar approach, interaction complexity may also be assessed for AOS.

REFERENCES

1. Aggarwal, K.K., Singh, Y., Kaur A, Malhotra R. 2009. Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: A replicated case study. *Software Process: Improvement and Practice 2009*; (14): pp. 39-62, DOI= http://onlinelibrary.wiley.com/doi/10.1002/spip.389.
2. Han, J., Kamber, M. 2001. *Data Mining: Concepts and Method.* Harchort India Private Limited, 2001.
3. Ho, S, Xie, M, Goh, T.N. 2003. A study of the connectionist models for software reliability prediction. *Computers and Mathematics with Applications 2003*; (46): pp. 1037-1045.
4. Hosmer, D., Lemeshow, S. 1989. *Applied Logistic regression*, John Wiley and Sons 1989.
5. Jun, Z. 2007. Predicting software reliability with neural network ensembles. *Expert Systems with Applications 2009*; 36(2): pp. 216-222. DOI= http://10.1016/j.eswa.2007.12.029.
6. Kai, Y.C., Lin, C., Wei, D.W., Zhou, Y.Y., and David, Z. 2001. On the neural network approach in software reliability modeling, *Journal of Systems and Software 2001*; (58): pp. 47-62.
7. Karunanithi, N., Whitley, D., and Malaiya, Y.K. 1992. Prediction of software reliability using connectionist models. *IEEE Transactions on Software Engineering, 1992*; 18(7): pp. 563-574.
8. Lyu, M.R. 1999. *Handbook of Software Reliability Engineering.* McGraw Hill, India, 1999; 131-151.
9. MATLAB TOOLBOX, http://www.mathworks.com "MatLab Toolbox for ANN, FIS, ANFIS".
10. Ping, F.P., and Wei, C.H. 2006. Software reliability forecasting by support vector machines with simulated annealing algorithms. *The Journal of Systems and Software* 2006; 79: pp. 747–755.
11. Ross, Q. C4.5: 1993. *Programs for Machine Learning*. Morgan Kaufman Publishers, San Mateo, CA 1993: 231-254.
12. Rajesh Kumar, P.S. Grover, and **Avadhesh Kumar** *"A Fuzzy Logic Approach to Measure Complexity of Generic Aspect-Oriented Systems"*, **Journal of Object Technology (JOT),** Volume 9, No. 3, pp: 43-57, May/June 2010.
13. **Avadhesh Kumar**, Rajesh Kumar, and P.S. Grover, *"A Comparative Study of Aspect-Oriented Methodology with Module-Oriented and Object-Oriented Methodologies"*, **ICFAI Journal of Information Technology**, Volume 2, No 4, pp: 7-15, December 2006.
14. **Avadhesh Kumar**, Rajesh Kumar, and P.S. Grover, *"Towards a Unified Framework for Complexity Measurement in Aspect-Oriented Systems"* , 2008 International Conference on Computer Science & Software Engineering (CSSE 2008), Wuhan, China, pp: 98-103, **IEEE Computer Society,** December 12-14, 2008.

AUTHORS PROFILE

Amarpal Singh Pursuing Master of Technology in Computer Science and Engineering from Amity School of Engineering and Technology, Amity University Uttar Pradesh, Noida, India, Area of Interest: Software Engineering and Soft Computing.

Piyush Saxena Pursuing Master of Technology in Computer Science and Engineering from Amity School of Engineering and Technology, Amity University Uttar Pradesh, Noida, India, Area of Interest: Data Mining and Warehousing and Soft Computing.

Abhishek Singhal is working as an assistant professor in Amity University Uttar Pradesh, Noida. He has obtained B.E. M.Tech. from MJP Rohilkhand University, Bareilly and U.P. Technical University, Lucknow respectively. Currently he is pursuing PhD from Amity University Uttar Pradesh, Noida, His research interest includes software testing, aspect-oriented systems, image processing and software engineering.