International Journal of Engineering & Technology, 7 (3.27) (2018) 142-146

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

New Software Testing Method Based on Evolutionary Algorithm

Samiyeh Khosravi

Faculty of Electrical and Computer Engineering, University of Birjand, IRAN
**Corresponding author E-mail: skhosravi@birjand.ac.ir*

Abstract

Software Testing is an important part of software development which is performed to support and enhance reliability and quality of software. It consists of estimating testing effort, selecting suitable test team, designing test cases, executing the software with those test cases and examining the results produced by these methods. This paper proposes Fuzzy Based approach for finding out the complexity weight based on requirement. Based on the weight, this paper computes the software complexity point. Fuzzy logic uses membership functions to include linguistic variables and quantifiers. Fuzzy Logic could be used in project estimation purposes efficiently by gathering size data on previously developed programs. The advantage of this model is that it is able to estimate the software complexity which in turn predicts the software requirement stability during the software development cycle.

Keywords: *Software Reliability, Evolutionary Algorithms, Fuzzy Logic Algorithm*.

1. Introduction

Many of the Software development phases are highly communication-intensive activity that involves, at minimum, analysts, architects, developers, testers, business stakeholders, and end users. Among all the development phases and products, requirements are key ingredient in the process of designing and realizing any systems. Without clearly understanding requirements and their proper management, large projects are likely to fail and could have very high maintenance cost. So, the focus of every development methodology is on requirement engineering phase. Software development involves roughly 50 percent computing and 50 percent communication. Whenever human communication involve, various linguistic variables come into picture e.g. consider two statements Trading Quantity is heavy and Room Temperature is high, here Trading Quantity and Room Temperature are linguistic variables. Linguistic variables can take both qualitative and quantitative values. Unfortunately, most teams are better at computing requirements, however, are almost entirely about communication.
Because there are many links in the requirements communication chain, a breakdown in any of these links leads to significant problems. This human communication might be the major factor in Cobb's Paradox. As A large number of linguistic quantifiers are used in human discourse. There are two classes of linguistic quantifiers: absolute and proportional. Absolute quantifiers such as 'about 10' and 'about 20' can be represented as a fuzzy set Q of the non negative real. A proportional linguistic quantifier indicates a proportional quantity such as 'most', 'many' and 'few'. These quantifiers are used in many stages of software development. So, some special method is required to deal with uncertainty and linguistic quantifiers.
We identify three aspects of test planning where uncertainty is present: the artifacts under test, the test activities planned, and the plans themselves. Software systems under test include, among others: Requirements specifications, produced by requirements elicitation and analysis. Design representations, produced by architectural and detailed design. Source code, produced by coding and debugging.

2. Literature Review

2.1 Background Work

Software development is very important and necessary in today's world in any field of applied science and technology. However, optimal software should be in such a way that it can best satisfy the constraints like cost, time, efforts and other resources. This can be achieved by formulating a systematic software development life cycle along with Software Testing Effort (STE) to ensure that above constraints can be satisfied fully before a software release.
Test engineering managers use many different methods to estimate and schedule their test engineering efforts. Different organizations use different methods depending on the type of projects, the inherent risks in the project and the technologies involved into them. There has been many research on Software Testing but research on Software Testing Effort estimation has not received due attention. Estimating the cost and duration of software testing effort is a major challenge.
Presently an early prediction of STE is based on the testing metrics which generally overestimate the efforts depending on the expertise of software testing team for developing the software. Halstead developed a set of metrics to measure a program module's complexity directly from source code, with emphasis on computational complexity. These measures were developed as a means of determining a quantitative indication of complexity directly from the operators and operands in the module. The study evaluates overall software testing effort to be allocated for a particular module. However, it has been observed that the study overestimates analysis of software testing efforts. The testing effort estimation using cognitive information complexity method has also been discussed by some of the researchers in which the time required to comprehend software is considered to be proportional to its cognitive complexity. In this approach, an existing simple

cognitive metric dealing with the important parameters of software is required for estimation of test effort which finally demonstrates how often used cyclometric number test effort increases with the increase in the software complexity. The number of test cases which can be determined by the function point's estimate for the corresponding effort. The actual effort in person-hours was then calculated with a conversion factor obtained from previous project data. The main disadvantage of using function points is that they require detailed requirements in advance.

2.2 Fuzzy Logic Scheme

Traditional requirement engineering techniques such as functional analysis have lots of draw back with them. Then, One of the most popular analysis technique known as Object Oriented Analysis came, Object-oriented analysis (OOA) is concerned with developing software engineering requirements and specifications that expressed as a system's object model (which is composed of a population of interacting objects), as opposed to the traditional data or functional views of systems. OOA can yield the following benefits: maintainability; reusability and high productivity. But still there are lots of drawbacks with OOA as: information loss at early stages, lack of design alternatives at later phases and context dependency. The main reason behind these limitations is by the fact that requirement engineering is a high communication intensive activity, whenever human communication involve there will be a fair chance of ambiguity and information loss. The cause for this information loss is use of classical two valued logic. For example in object oriented analysis either an entity in a problem domain is taken as class or not, there is no third possibility, that is only two quantization levels possible and it will increase information loss due to high quantization error.

Broadly, problems dealing with fuzzy logic concepts can divided into three steps:
(1) Fuzzification of data: In this step, the Fuzzification process is carried out by developing membership functions generated from different input sources.
(2) Development of Fuzzy Rule Base: The fuzzy rule base is usually constructed from the experience of the decision maker. This phase comprises of applying the fuzzy rule base over the fuzzy input and arriving at the fuzzy output.
(3) Defuzzification of Output Data: It converts the fuzzy output into crisp output. There are several ways of defuzzification. Some of them are mean-of-maxima method, center of gravity method, modified center of gravity method, height method etc. In this paper, center of gravity technique has been used for defuzzification.

3. Fuzzy Logic for Software Reliability Analysis

The uncertainty permeates these processes and products. Plans to test these artifacts, therefore, will carry their uncertainties forward. Software testing, like other development activities, is human intensive and thus introduces uncertainties. These uncertainties may affect the development effort and should therefore be accounted for in the test plan. In particular, many testing activities, such as test result checking, are highly routine and repetitious and thus are likely to be error-prone if done manually, which introduces additional uncertainty. Test planning activities are carried out by humans at an early stage of development, thereby introducing uncertainties into the resulting test plan. Also, test plans are likely to reflect uncertainties that are, as described above, inherent in software artifacts and activities. Test enactment includes test selection, test execution, and test result checking. Test enactment is inherently uncertain, since only exhaustive testing in an ideal environment guarantees absolute confidence in the testing process and its results. This ideal testing scenario is infeasible for all but the most trivial software systems. Instead, multiple factors exist, discussed next, that introduce uncertainties to test enactment activities.

3.1 Test Procedure

Test selection is the activity of choosing a finite set of elements (e.g., requirements, functions, paths, data) to be tested out of a typically infinite number of elements. Test selection is often based on an adequacy or coverage criterion that is met by the elements selected for testing. The fact that only a finite subset of elements is selected inevitably introduces a degree of uncertainty regarding whether all defects in the system can be detected. One can therefore associate a probability value with a testing criterion that represents one's belief in its ability to detect defects.

3.2 Test Process

Test execution involves actual execution of system code on some input data. Test execution may still include uncertainties, however, as follows: the system under test may be executing on a host environment that is different from the target execution environment, which in turn introduces uncertainty. In cases where the target environment is simulated on the host environment, testing accuracy can only be as good as simulation accuracy. Furthermore, observation may affect testing accuracy with respect to timing, synchronization, and other dynamic issues. Finally, test executions may not accurately reflect the operational profiles of real users or real usage scenarios. Test result checking is likely to be error-prone, inexact, and uncertain. Test result checking is afforded by means of a test oracle, which is used for validating results against stated specifications. Test oracles can be classified into five categories, offering different degrees of confidence. Specification-based oracles instill the highest confidence, but still include uncertainty stemming from discrepancies between the specification and customer's informal needs and expectations.
Here we can see how much uncertainty is involved in testing phase of software development. Though there are many approaches to deal with uncertainty these are: Fuzzy logic, Bayesian networks, Certainty Factor approaches, Dempster-Shafer approaches, and monotonic logic. But as we have seen in previous part of the paper uncertainty can be better modeled by Fuzzy logic. Other than modeling uncertainty fuzzy logic also incorporate notion of partial membership.

4. Software Testing rooted on Fuzzy Logic

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. 1.

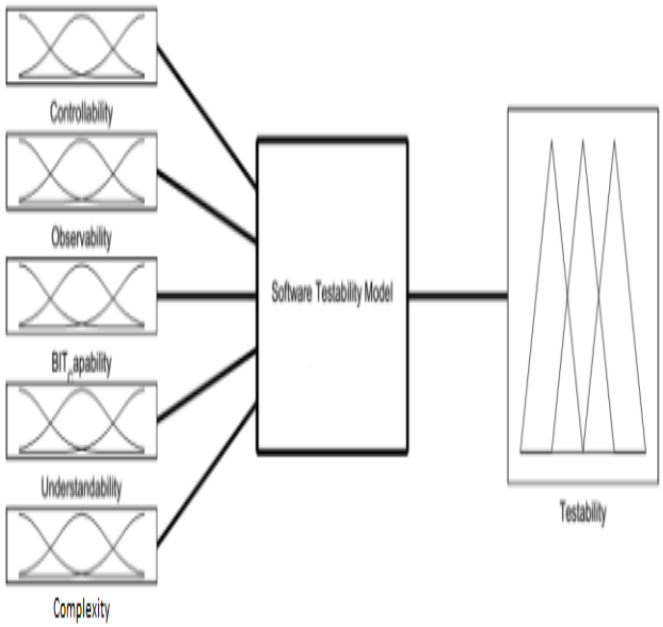

Fig 1: Software Testability scheme with 5 inputs, 1 output, 243 rules

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 Observability 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 Observability 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 Observability 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 Observability 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 Observability 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 1.

Table 1: Explanations of the System used

Name of the system	Testability
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. 2.

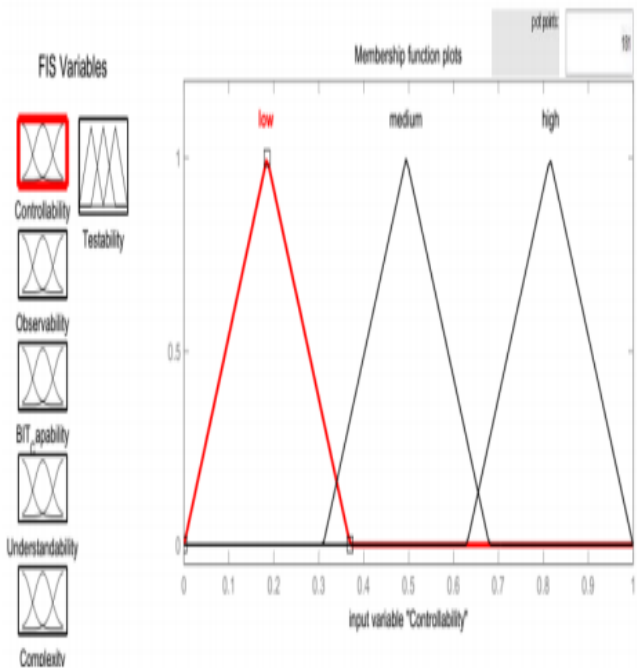

Fig 2: Relationship Function

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. 3.

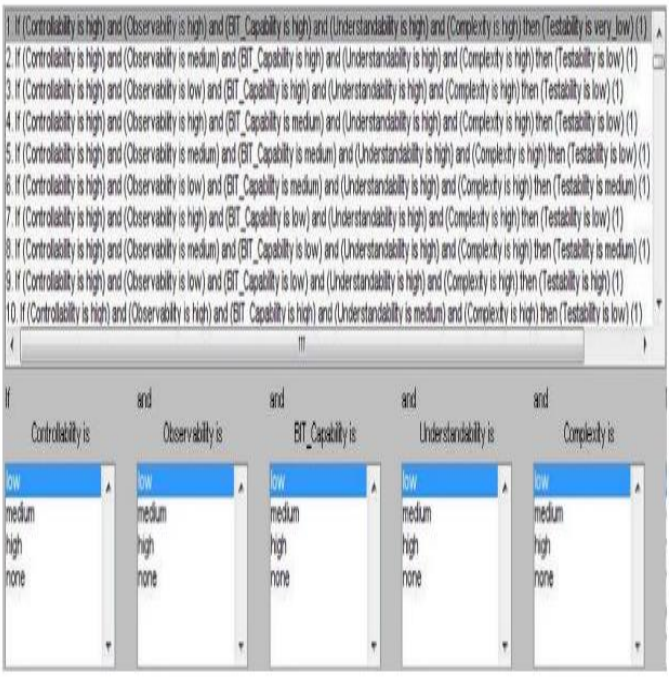

Fig 3: Planned fuzzy model rule base

Testability can be examined by some changes in the above rule viewer are shown in Fig. 4, 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

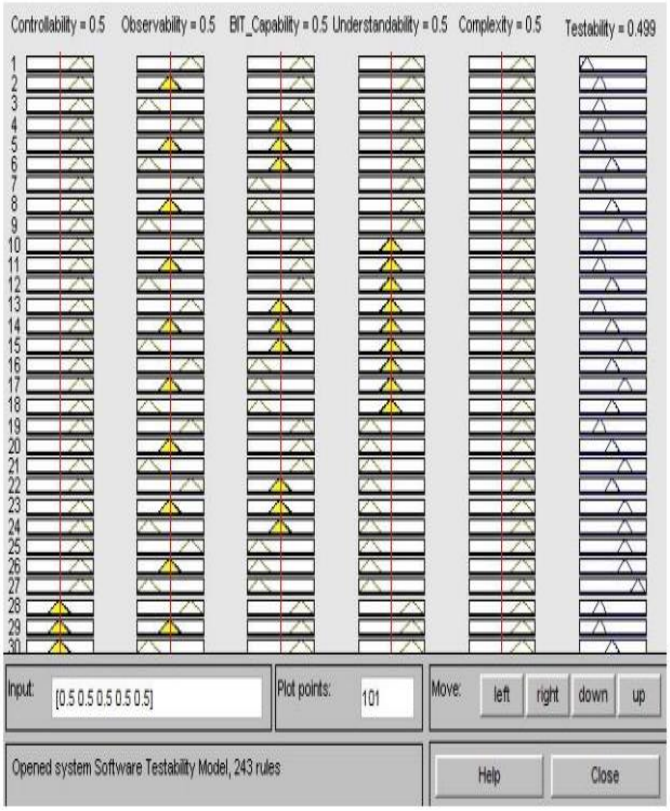

Fig 4: Schematic of Fuzzy rule viewer

5. Investigating the 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 an output which is categorized as very low, low, medium, high, and very high. Total 243 rules are created based on expert advice 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.

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. 5.

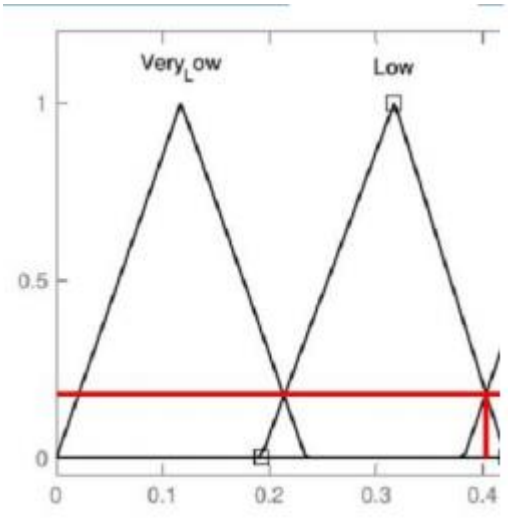

Fig 5: Output for Software Testability

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.

5.1 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 techniques 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 is applied on the output for testability as shown in Fig. 6.

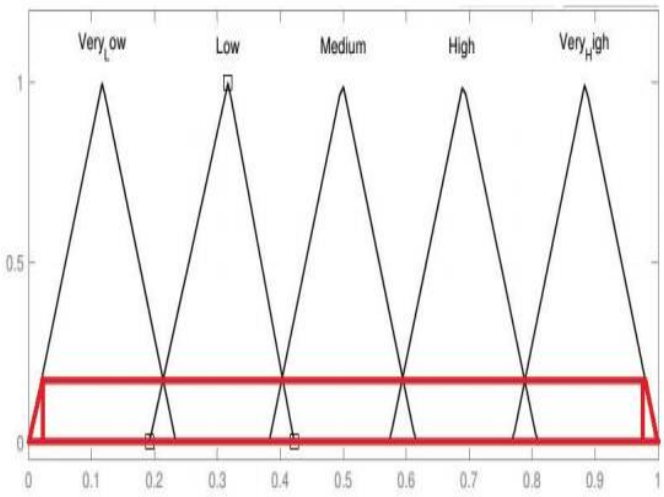

Fig 6: De-Fuzzification of Software Testability

It is the most accurate technique for defuzzification and formula is $x^* = 0.4997$

The testability for the inputs appeared above is 0.499 that is same as evaluated above. Hence our proposed fuzzy model is validated by defuzzification and same result has been achieved for software testability.

6. Conclusion

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 ad-

vantage 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

References

[1] Zhitao He, Haihua Yan, Chao Liu and Huacan He, "A formal definition of software testing based on fuzzy measure*," Fourth World Congress on Software Engineering*, 2013.

[2] Manoj Kumar, Arun Sharma and Rajesh Kumar, "Fuzzy entropy-based framework for multi-faceted test case classification and selection: an empirical study," IET software,vol. 8, Iss. 3, 2014, pp. 103-112.

[3] Xiaojing Wang, Angel F. Garcia and Martine Ceberio, "Interval-based Algorithms to extract Fuzzy Measures for Software Quality Assessment,",IEEE Trans., 2012.

[4] Aleksander Dimov and Sasikumar Punnekkat, "Fuzzy reliability model for component-based software system," EUROMICRO Conference on Software engineering and Advanced Applications, 2010.

[5] Mahdi Sharifi, Azizah Abdul Manaf, Ali Memariani, Homa Movahednejad and Amir Vahid Dastjerdi, "Consensus-based service selection using crowdsourcing under fuzzy preferences of users," *IEEE International Conference on Services Computing.* 2014

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

[7] 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.

[8] ISO 9126, International Organization for standardization