An Estimation of Software Testability using Fuzzy Logic

Umang Garg
Department of Computer Science & Engg.
Amity University, Uttar Pradesh
Noida, India
umanggarg76@gmail.com

Abhishek Singhal
Department of Computer Science & Engg.
Amity University, Uttar Pradesh
Noida, India
asinghal1@amity.edu

Abstract-Testability and its significance have been discussed in the past decade. Software testability insinuates to the ease of the code revealing its faults during testing. This paper proposes to take into account the developer and the development process in the software testability predictions along with other factors. Software testability plays an important role in identifying the acceptable risk and feasibility. Hence, testability analysis is very important today, which requires identifying all the possible factors and phases of software lifecycle that affect testability. This paper introduces 5 new factors, which affect testability of object oriented software and all of them take developer and development process into account. This paper also explains how these factors affect testability.

Keywords- Testability, Object oriented software, Developer, Design, Fuzzy model

I. INTRODUCTION

Testability is a quality measure that allows a component to be easily tested in isolation. As the system becomes larger, its complexity increases and it becomes more difficult to develop. Complex systems have begun to replace critical human decision in every facet and it has become increasingly essential to not only ensure its high reliability but also to find out it's feasibility from the developer's point of view.

Testability is the level of intricacy of testing a system. Assume our tests have 0.01 probability of revealing a bug and there are 10,000 errors in our framework. We have to keep running not less than 10,00,000 tests to discover these bugs. Testability decides the limit to which the risk of exorbitant or unsafe bugs can be reduced to a tolerable level. Poor testability implies one is probable to ship the software with bugs which are likely to reveal later at customer end [1]. Hence one can make an informed decision whether or not the software is feasible to develop or not here.

Testability is an obscure concept. Testability is not an intrinsic property as it cannot be calculated directly. Instead testability is an extrinsic attribute which depends on many factors as well as their interdependency. It is an arduous task to achieve a lucid understanding of all the possible factors which may affect testability and to what extent these factors affect testability.

The first research on software testability took place in the year 1975 [2]. It has been used in Boehm [3] and McCall [4] software quality model that became the base of ISO 9126 quality model [5].

As indicated by IEEE [6], software testability is the extent to which a software artifact (i.e. a software module, software system, design or requirements document) scaffolds testing in a given test context. In the event of high testability of the software artifact, finding bugs in the software (if it has any) by the method of testing will be easier. Testability depends on various factors like controllability, observability, isolateabilty, separation of concerns, understandability, automatability, heterogeneity. The purpose of this paper is to introduce some other factors that affect testability.

The remaining part of this paper is divided into 5 sections: the next 2 sections introduce fuzzy logic, fuzzy model and their types whereas the section following that gives an overview of previous work in same domain. The remaining sections describe the factors that affect testability of software and how they affect testability.

II. FUZZY LOGIC

Fuzzy Logic (FL) is a method of calculation constructed on the extent of truth or opinion than the typical 0 or 1, true or false, Boolean logic on which most computers are based. Fuzzy Logic (FL) is a technique of reasoning which bears resemblance to human reasoning. The method of Fuzzy Logic mimics the manner in which humans make decisions which includes all transitional options between a digital value 1 and 0.

The concept of Fuzzy Logic (FL) was first introduced by Dr. Lotfi Zadeh, University of California, Berkeley in 1960s. Dr. Zadeh was operating on the issue of the uncertainties of natural language. Just like most problems in life, Natural language cannot be easily transformed into the absolute values of 0 and 1. Fuzzy Logic (FL) handles the idea of partial truth that is the values which lie in between "completely true" and "completely false". It includes 0 and 1 as extreme cases

Lotfi Zadeh, perceived that unlike computers, the human's decision has a variety of possible answers between TRUE and FALSE, being –

DEFINITELY TRUE

MAY BE TRUE

CANNOT SAY

MAY BE FALSE

DEFINATELY FALSE

Fuzzy Logic (FL) is used as it may not provide us with an exact reasoning but provides with an acceptable reasoning.

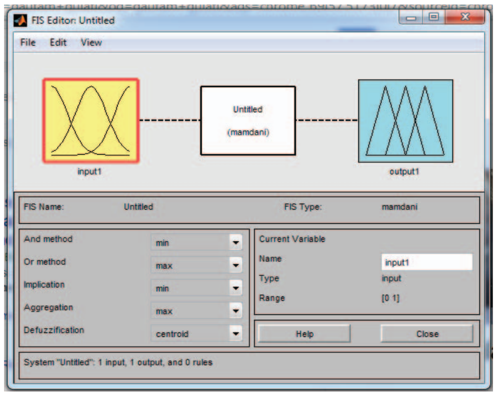
It can be implemented in a variety of systems ranging from large workstation based control systems to small micro-controllers.

III. FUZZY MODEL

A mathematical model which uses fuzzy sets is called fuzzy model. In them relationship between inputs and outputs are illustrated by if-then rules. Based on the type of structure of if-then rules, 2 main models can be classified:-

- Mamdani Model
- Takagi-Sugeno Model

MAMDANI MODEL

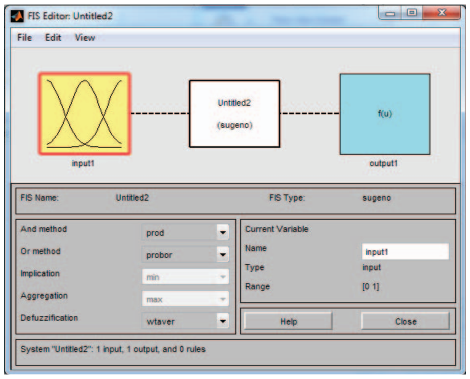


FIS Name: Untitled
FIS Type: mamdani
And method: min
Or method: max
Implication: min
Aggregation: max
Defuzzification: centroid
Current Variable
Name: input1
Type: input
Range: [0 1]
System "Untitled": 1 input, 1 output, and 0 rules

FIGURE 1. MAMDANI FUZZY MODEL

In Mamdani model both the If part and Then part are fuzzy proposition. It is advantageous in representing qualitative knowledge. It is mostly used in expert systems.

TAKAGI-SUGENO MODEL



FIS Name: Untitled2
FIS Type: sugeno
And method: prod
Or method: probor
Implication: min
Aggregation: max
Defuzzification: wtaver
Current Variable
Name: input1
Type: input
Range: [0 1]
System "Untitled2": 1 input, 1 output, and 0 rules

FIGURE 2. TAKAGI SUGENO MODEL

The if-part is defined in the same way as in Mamdani model that is proposition whereas the then- part is a function of the input variables.

IV. PREVIOUS WORK

Testability is an integral internal characteristic of software. A lot of studies investigated testability within different application domains to better understand the significance of testability. [7]

Voas and Miller [8, 9] in their research highlighted the concept of software testability. If a program comprises of faults the odds that a test case may fail was characterized as testability. They presented a metric for evaluating testability which depended on test case inputs and their resulting outputs. They likewise introduced a methodology Propagation, Infection, and Execution (called PIE) to assess the testability of software. However, this technique was highly complex and diverse.

Binder [1] stressed on the significance of enhancing testability during the software development life cycle. He put forward a few already existing metrics to estimate testability, including: LCOM (Lack of COhesion in Methods), DIT (Depth of Inheritance Tree), OVR (percentage of non-OVeRloaded calls) and DYN (percentage of DYNamic calls) [10]. He introduced a fishbone model exemplifying testability as a result of six factors being: (1) representation; (2) implementation; (3) the test support environment; (4) the software testing process; (5) built-in test capabilities; (6) the test suite. However, each of these factors assesses testability at a high abstraction level with a little or no accessible association with its implementation or its object oriented design [7].

McGregor et al. [11] presented a factor namely the VC (Visibility Component) to calculate the testability of object oriented software. VC is susceptible to object oriented features like exceptions, collaboration, encapsulation and inheritance. Availability of complete and accurate specification documents is an essential requirement of VC.

Bruce et al. [12] combined testability factors with the designing of object oriented software. They presented a fundamental system to find out the testability metric for software. They gave a few rules to follow while designing of OO Software to improve testability. But their model had a few drawbacks from implementation perspective.

Jungmayr [13] gave relationship between testability and static dependencies. He measured testability which was based on static dependencies in order to find out test-critical dependencies. He likewise characterized a new metric in the light of dependencies to calculate the effect of dependencies over testability. However, he did not consider the interactions between the dependencies, leaving behind the combined effect of the set of dependencies on testability.

Baudry et al. [14] enquired estimation of OO Design's testability by emphasizing on the patterns of design and how they enhance testability. They combined testing efforts with testability and focussed on class interactions in class diagram. However, this hypothesis wasn't empirically validated.

Briand et al. [15] introduced a new methodology, which makes use of class invariants, preconditions, and post conditions (instrumented contracts) in order to improve testability. These instrumented contracts were believed to increase the odds that a bug will be found when we execute the test cases. Depending on the accuracy of the contracts, the contract assertions diagnosed a high percentile of errors as revealed by their work. These assertions decreased the efforts required to detect bugs.

Mulo [16] highlighted the significance of testability measurement through every phase of software development. He organised metrics to quantify testability into 3 bifurcations namely data flow, behavioural and structural metrics. The research revealed that the metrics were ideal to direct developers and code reviewers than to measure the testability of software. He inferred that testability evaluation wasn't a simple task and needed great efforts and humongous budget.

Jianping Fu et al. [17] introduced a request-oriented approach for the measurement of software testability. Their self-contained method gathered a few inputs to finish all types of testability measurements. They tried their methods over a minor case study to prove its efficacy. However, their method wasn't tested on a variety of samples to reveal its actual effectiveness and hence, their methodology hasn't been broadly acknowledged.

Khatri et al. [18] introduced the use of reliability growth models to enhance testability. They recognised that testability of software is greatly affected by a fault's complexity. They put forward a few theoretical methods for enhancing and calculating the testability of software. However, the improvement in the value of testability was not evaluated and their model was completely hypothetical [7].

Nazir et al. [19] formed a testability metric that calculated testability on basis of 2 factors namely Complexity and Understandability. A lot of researchers concentrated upon usage of metrics for testability evaluation of object oriented software at various software development phases [20]–[25].

After this concise historical study of research works conducted on testability, we found that we can sum up all the factors that have been proposed earlier, which affect testability and their definition as given below-

i. Controllability: The extent to which it is viable to control the condition of the CUT (Component Under Test) as required for testing. [26]

ii. Observability: The extent to which it is viable to observe all (intermediate and final) test results.

iii. Isolateability: The extent to which the CUT (Component Under Test) can be tested in isolation. [28]

iv. Separation of concerns: The extent to which a single, well defined responsibility is performed by the CUT (Component Under Test).

v. Understandability: The extent to which the CUT (Component Under Test) is well documented or self-clarifying.

vi. Automatability: The extent to which the CUT (Component Under Test) can be tested using automated testing.

vii. Heterogeneity: The extent to which various test tools and methods are required to be used in parallel by various technologies.

V. PROPOSITION

In this section, we introduce some factors that affect testability of object oriented software. Testability refers to the simplicity with which a code will reveal its errors during random testing. The importance of testability is recognised in every previous work and how it needs to be paid attention at early stages of software development lifecycle. Software is developed by the immense hard work of the developers, then how can they not affect testability. We propose factors strictly related to software developers that may affect testability –

- Years of coding experience
- Time for development
- Previous development of similar projects
- Confidence in main language
- Efforts

i. YEARS OF CODING EXPERIENCE

Years of coding experience refers to the no. of years for which he has developed codes. Based on this, they may have low, medium or high coding experience.

A developer with low coding experience is generally not self-directed, he may need assistance in drawing solutions, he would require tasks rather than requirements or user stories.

A developer with medium coding experience is self-directed and can work with requirements. He would be comfortable in coding in his main language and have basic understanding of a few additional languages and platforms.

A developer with high coding experience is not only self-directed but can also develop from user stories, elucidate requirements and can derive solutions more easily. He is comfortable with more than one language and platforms and has moderate to high understanding of object oriented and functional programming.

Testability is affected by understandability. Understandability depends on two factors—

- availability of artifacts for users and developers
- well documented or self-explaining software

A developer with high coding experience can better understand user requirements and elucidate them in a well-documented form. He would build a functionally and structurally simple and self-explanatory code. Hence, they provide a higher testability when compared with a developer having low coding experience.

ii. TIME FOR DEVELOPMENT

Time for development is the time a developer is provided to develop the code and hand it over to the testing team. It does not include testing time.

When a developer has ample amount of time to develop a code he builds a less complex, more sophisticated and self-explanatory code. If compared with another developer or with the same developer, given less time to develop the same software, one could remarkably see the difference. When given more time, the outcome is more sophisticated and less complex

Hence, ample or more time for development leads to a more understandable and less complex code as compared to when given less amount of time for development. This increase in understandability and decrease in complexity gives higher testability.

iii. PREVIOUS DEVELOPMENT OF SIMILAR PROJECT

If a developer has developed similar projects previously it leads to a better understanding of the project for the developer. It not only decreases the time to develop the project but also increases the testability of the project due to the increased understandability and controllability.

Understandability suggests the developer's ability to understand the design of the software and correctly implement it. The developer having already developed a similar project would have a better understanding of the design and code of the project and can also provide better controllability of all the inputs. This better understanding and controllability of the project leads to a better testability.

Though the understandability of a developer with no experience on developing similar project may be high but it will never be as high as the one who has already developed it. Hence the testability will be lower for a developer with no or less experience on similar projects than for a developer with good experience on developing similar projects.

iv. CONFIDENCE IN LANGUAGE

Almost all developers have knowledge of more than one programming language. Even with knowledge of more than one language no developer can have equal knowledge in all of them. Hence, a developer can never feel equally confident in each programming language in which he develops a code. He would always encompass a little more confidence in one language than the others and may have a little less confidence in some languages than others. He would have more knowledge in one than the other and hence feel more comfortable and confident in developing codes in that programming language.

Developer's confidence in a particular language allows him to develop a less complex and self-explanatory code in that language than developing the same code in another language in which the developer is less confident.

It is very similar to having more coding experience than another developer. If a developer has more coding experience than another developer in a particular coding language, he will feel more confident than the other and hence develop a more understandable and testable code.

v. EFFORTS

The effort a developer puts in developing software is very essential for quality of end-product. If huge amount of efforts are put in developing the software, the end-product will be of better quality. Whereas if a developer puts in less effort to develop it, the end-product though may be acceptable but will be of poor or less quality.

Testability is a quality measure. If huge amount of efforts are put in developing software, the outcome is a more palpable one. But if fewer efforts are put in developing software, a less sophisticated code is developed leading to lower testability.

Though effort is an extrinsic measure, if compared two outputs of same type, one with more efforts put in development and the other with fewer efforts put in development, the first one would be more self-explanatory and hence, with higher testability.

VI. EXPERIMENTATION

We propose a model using fuzzy systems with inputs as Years of Coding Experience (YCE), Time for Development (TD), Previous Development of Similar projects (PDS), Confidence in Main Language (CML), Efforts (E) and output Testability (T).

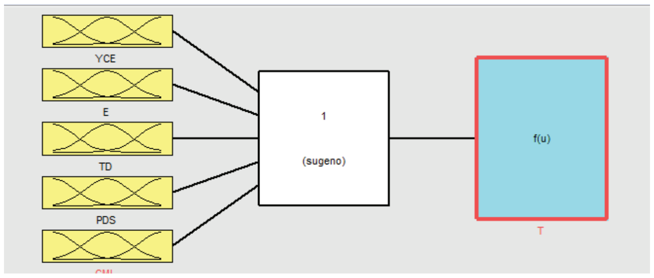


FIGURE 3. FUZZY MODEL WITH 5 INPUTS

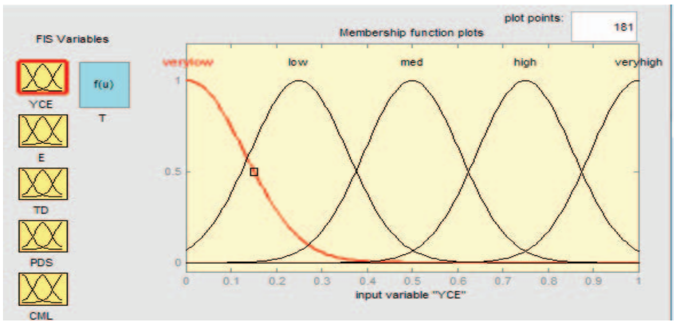
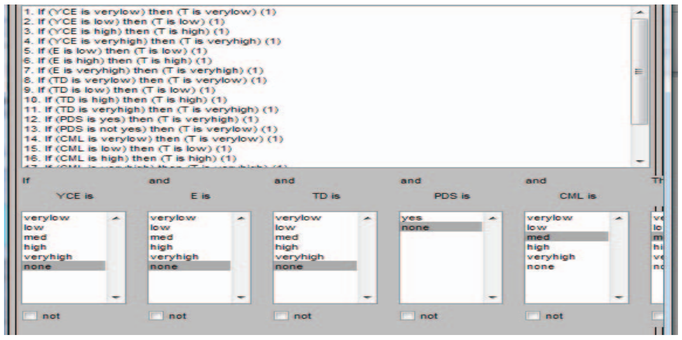


FIGURE 4. MEMBERSHIP FUNCTIONS



```
1. If (YCE is verylow) then (T is verylow) (1)
2. If (YCE is low) then (T is low) (1)
3. If (YCE is high) then (T is high) (1)
4. If (YCE is veryhigh) then (T is veryhigh) (1)
5. If (E is low) then (T is low) (1)
6. If (E is high) then (T is high) (1)
7. If (E is veryhigh) then (T is veryhigh) (1)
8. If (TD is verylow) then (T is verylow) (1)
9. If (TD is low) then (T is low) (1)
10. If (TD is high) then (T is high) (1)
11. If (TD is veryhigh) then (T is veryhigh) (1)
12. If (PDS is yes) then (T is veryhigh) (1)
13. If (PDS is not yes) then (T is verylow) (1)
14. If (CML is verylow) then (T is verylow) (1)
15. If (CML is low) then (T is low) (1)
16. If (CML is high) then (T is high) (1)
```

FIGURE 5. RULE BASE FOR FUZZY MODE

We used gaussian membership function and s-shaped membership function for inputs and fire the rule base to obtain outputs. We observed the value of testability (0.718) for various values of input [0.8,0.2,0.7,0.4,0.8]

We further observed the value of testability for variety of input values to evaluate the combined effect of all the factors over testability.

TABLE 1. TESTABILIY FOR DIFFERENT VALUE

INPUTS					OUTPUT
YCE	E	TD	PDS	CML	TESTABILITY
0.8	0.2	0.7	0.4	0.8	0.718
0.4	0.8	0.8	0.7	0.3	0.649
0.7	0.9	0.9	0.5	0.4	0.787
0.3	0.8	0.6	0.4	0.9	0.702
0.2	0.3	0.9	0.6	0.4	0.543
0.1	0.7	0.3	0	0.7	0.428

It is observed that even with low coding experience but high efforts, one can achieve high testability not as high as for a developer with high coding experience but relatively comparable. The effect of these factors as a combination varies for a variety of set of values.

VII. CONCLUSION

We studied the effect on testability for factors- years of coding experience, time for development, previous development of similar projects, confidence in main language and effort using fuzzy system. We observed the effect of these factors on testability, individually as well as their combined effect. Testability is an important software quality attribute and one needs to heed to it at all phases of software development lifecycle in order to achieve better testability.

REFERENCES

[1] Robert V Binder, "*Design for Testability in Object-Oriented Systems*", Communications of the ACM, 37(9), Sept. 1994, pp. 87-101.

[2] Liang Zhao, "A *New Approach for Software Testability Analysis*", in the Proceeding of the 28th international conference on software engineering - ICSE 06 ICSE 06, 2006.

[3] Boehm B.W. Brown J.R., Lipow H., MacLeod G.J and Merit M.J., "*Characteristics of Software Quality"*, In TRW, Amsterdam, North Holland,1978

[4] McCall J.A., Richards P.K. and Walters G.F., "*Factors in Software Quality"*, NIST, New York, 1977.

[5] ISO. *ISO/IEC IS 9126, "Software Engineering Product Quality*", ISO Press, Geneva, Switzerland, 1991

[6] *IEEE Standard Glossary of Software Engineering Terminology*. IEEE Computer Society Press, New York, 1990.

[7] Mamdouh Alenezi, "*Software Testability: Recovering From 2 Decades of Research*", International Journal of Computer Applications, 113(7), March 2015, pp. 1-5.

[8] Jeffrey M Voas and Keith W Miller, "*Semantic Metrics for Software Testability*", Journal of Systems and Software, 20(3), March 1993, pp. 207-216.

[9] Jeffrey M Voas and Keith W Miller, "*Software Testability: the New Verification",* IEEE software, 12(3), May 1995, pp. 17-28.

[10] Mouchawrab, S, "*A Measurement Framework for Object-Oriented Software Testability*", Journal of Information and Software Technology, 47(15), 2005, pp. 979-997.

[11] John D McGregor and Satyaprasad Srinivas, "*A Measure of Testing Effort*", In the Proceedings of the 2nd conference on USENIX Conference on Object-Oriented Technologies (COOTS)-Volume 2, USENIX Association, 1996, pp. 10-10.

[12] Bruce WN Lo and Haifeng Shi, "*A Preliminary Testability Model for Object-Oriented Software*", In the Proceedings. International Conference on Software Engineering: Education & Practice, 1998, IEEE, 1998, pp. 330-337

[13] Stefan Jungmayr, "*Testability Measurement and Software Dependencies*", In the Proceedings of the 12th International Work-shop on Software Measurement, October 2002, pp. 179–202.

[14] Benoit Baudry and Yves Le Traon, "*Measuring Design Testability of an UML Class Diagram*", Journal of Information and Software Technology, 47(13), Oct. 2005, pp.859–879.

[15] Lionel C Briand, Yvan Labiche, and Hong Sun, "*Investigating the Use of Analysis Contracts to Improve the Testability of Object-Oriented Code*", Journal of Software: Practice and Experience, 33(7), 2003, pp. 637–672

[16] Emmanuel Mulo, "*Design for Testability in Software Systems*", Master's thesis, Delft University of Technology, Delft, the Netherlands, 2007, pp.1-44.

[17] Jianping Fu and Minyan Lu, "*Request-Oriented Method of Software Testability Measurement*", In the Proceedings of International Conference on Information Technology and Computer Science, 2009, pp. 77–80.

[18] Sujata Khatri, RS Chhillar, and VB Singh, "*Improving the Testability of Object-Oriented Software during Testing and De-Bugging Processes*", International Journal of Computer Applications, 35(11), 2011, pp.24-35.

[19] M. Nazir, R. A. Khan, and K. Mustafa, "*A Metrics Based Model for Understandability Quantification*", Journal of Computing, 2(4), April 2010, pp. 90–94.

[20] B. Pettichord, "*Design for Testability*", In the Proceedings of *Pacific Northwest Software Quality Conference*, 2002, pp. 1–28.

[21] R. A. Khan and K. Mustafa, "*Metric Based Testability Model for Object Oriented Design (MTMOOD*)", ACM *SIGSOFT Software Engineering Notes*, 34(2), March 2009, pp. 1-6.

[22] D. Esposito, "*Design Your Classes for Testability*" 2008.

[23] J. E. Payne, R. T. Alexander, and C. D. Hutchinson, "*Design-for-Testability for Object Oriented Software*", *Object Magazine*, 7(5), 1997, pp. 34–43.

[24] Y. Wang, G. King, I. Court, M. Ross, and G. Staples, "*On Testable Object-Oriented Programming*", *ACM SIGSOFT Software Engineering Notes*, 22(4), July 1997, pp. 84–90.

[25] M. Harman, A. Baresel, D. Binkley and R. Hierons, "*Testability Transformation: Program Transformation to Improve Testability*", *Formal Method and Testing, LNCS*, 2011, pp. 320–344.

[26] Pradeep Kumar Singh, Om Prakash Sangwan, Amar Pal Singh, and Amrendra Pratap, "*An Assessment of Software Testability using Fuzzy Logic Technique for Aspect-Oriented Software*", International Journal of Information Technology and Computer Science, 7(3), 2015, pp. 18-26

[27] Charu Singh, Amrendra Pratap, Abhishek Singhal, "*An Estimation of Software Reusability using Fuzzy Logic Technique*", In the Proceedings of International Conference on Signal Propagation and Computer Technology, 2014, Ajmer, India

[28] Ming-Chih Shih, "*A Component Testability Model for Verification and Measurement*", In the Proceedings of 29th Annual International Computer Software and Applications Conference (COMPSAC 05), 2005, pp. 211-218.