Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 7, July 2014, pg.283 – 290

REVIEW ARTICLE

Enhancing Maintainability of Software Using Fuzzy Approach: A Review

Taruna, M.Tech Student
Department Of Computer Science and Applications, M.D. University, Rohtak, Haryana, India
Email id: tarunadeswal@gmail.com

Dr. Bal Kishan, Assistant Professor
Department Of Computer Science and Applications, M.D. University, Rohtak, Haryana, India

Abstract

Maintainability is considered an important factor for development of efficient software system. Software maintenance is a process of changing the existing software but leaving the primary functions intact. Maintenance of software includes a wide range of activities like correcting errors, enhancing the capability of software, deleting the capability of software and optimization .This paper is based on maintainability of software modules. It describes that how maintenance will affect the quality of software products. This paper discusses various issues related with the maintainability of software. Here, we will focus on how to improve maintainability of software module by using fuzzy based model approach.

Keywords: Maintenance, Fuzzy logic, Fuzzyfication, Defuzzyfication

Introduction

Software Maintainability [1, 2]

The term most frequently associated with more flexible software and significantly reduced long-term costs is maintainability. Frequently found definition of maintainability such as:

"The effort needed to make specified modification to a software system or component after delivery to correct faults, improve performances or other attributes, or adapt to a Changed environment."

and

"A system is maintainable if the correction of minor bugs only requires minor efforts."

are obviously overly simplified. The latter is particularly confusing as is in fact not a definition but a tautology. If one asked what a minor bug is, the answer would most certainly be "its correction requires minor efforts". Besides these rather native definitions, various metrics-based approaches try to define maintainability as compliances to a set of rules that correspond to measurable properties of the code, such as strong cohesion, limited coupling, etc. The general problem with this approach is the lack of a sound rationale for the selected criteria which in turn sometimes has a tendency of discussing some kind of technical beauty instead of, effectively improving software maintenance. In 2003, we conducted a study on software maintenance practices in German software organization. While 60 % out of the 47 respondents said that they would consider software maintenance [3] as a "significant problem", only 20% performed specific checking for maintainability during quality assurance. The criteria used by these 20% to check for maintainability differed significantly and ranged from object orientation, cyclomatic complexity limited numbers of lines per method, descriptive identifier naming, down to service oriented architecture or OMG's model driven architecture.

Types of Maintenance

Software under maintenance consists of finite number of states .The states have a specific operating efficiency. The maintenance process can bring the software from one state to another within a specific time slot allotted to the software maintenance engineers. The software fails or reaches its maximum efficiency depends upon the nature of maintenance problems. It is generally seen that maintainability very much depends on the type of Programs. As the number of object-oriented software systems increases, it becomes more important for organizations to maintain those systems effectively. However, currently only a small number of maintainability prediction models are available for object oriented systems. Software maintainability depends both on qualitative and quantitative data. Existing maintainability models [4, 5, 6, 7] aggregate data into hierarchies of characteristics with given dependencies. However, data used to score the characteristics can be uncertain or even completely unknown.

Therefore, it would be meaningful to evaluate sensitivity of the aggregated result, i.e. the maintainability, with respect to the uncertainty and incompleteness of data. In addition, real cases require an aggregation model able to evaluate the impact of changing the dependencies among the characteristics in the hierarchy on the maintainability. Maintenance has often taken a back seat where software development is concerned. However, once software is delivered it gets maintained for the rest of its useful life. With the continuing increase in software production more and more resources are spent on maintenance. As such, a closer look at improving our maintenance approach is needed. Software quality can be defined as the totality of features and characteristics of a product that could satisfy a given set of requirements. Some of the factors of software quality include reliability, reusability, maintainability and portability. These quality factors are then broken down into lower level quality criteria which serve as attributes of software. Contrary to perceived belief that much has been discussed about maintainability attributes, there is not much available in terms of a refined and organized attributes model. Software maintenance is defined as process of modifying existing operational software while leaving its primary functions intact. Every software needs to be modified to meet customer's requirement in its life Cycle. Software maintenance encompasses a broad range of activities including error corrections, enhancement of capabilities, and deletion of obsolete capabilities and optimization. The value of Software can be enhanced by meeting additional requirements, making it easier to use, more efficient and employing newer technologies. Even though it is an important task, it is poorly managed. The fact that you cannot control, what you cannot measure, makes measurement of maintainability very important. Maintainability is a measure of characteristics of software e.g. source code readability, documentation quality and cohesiveness among source code and documents. We will investigate how the "maintainability" of a piece of software changes as time passes and it is being maintained by performing measurements on industrial system. Across the 70's and 80's several authors have studied the maintenance phenomenon with the aim of identifying the reason that originates the needs for changes and their relative frequencies and costs. As a result

of these studies, several classifications of maintenance activities have been defined; these classifications of maintenance and its implications on the cost and the quality of the system in use. Dividing the maintenance efforts into categories has first made evident that software maintenance is more than correcting errors.

Ideally, maintenance operations should not degrade the reliability and the structure of the subject system; neither should they degrade its maintainability, otherwise future changes will be progressively more difficult and costly to implement. Unfortunately, this is not the case for real- world maintenance, which often includes a phenomenon of aging of the subject system; this is expressed by the second law of Lehman: "As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving the semantics and simplifying the structure." Accordingly, several authors consider a fourth category of maintenance, named preventive maintenance, which includes all the modifications made to a piece of software to make it more maintainable.

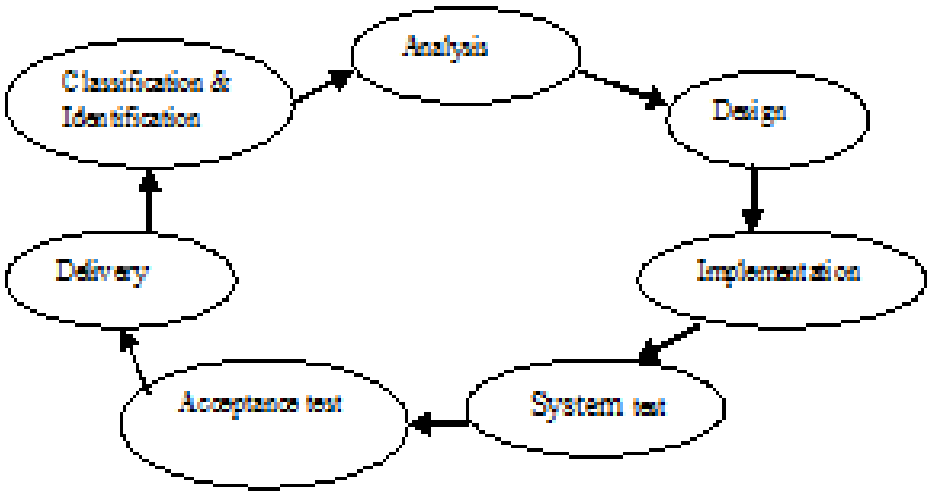


Figure1: Maintenance Process

- Corrective Maintenance: Reactive modification of a software product performed after delivery to correct discovered problems. It deals with fixing bugs in the code.
- Adaptive Maintenance: Modification of a software product performed after delivery to keep a software product usable in a changed or changing environment. It deals with adapting the software to new environment.
- Perfective Maintenance: Modification of a software product performed after delivery to improve performance or maintainability. It deals with updating the software according to changes in user requirements.
- Preventive Maintenance: Modification of a software product performed after delivery to detect and correct latent faults in the software product before they become effective faults. It deals with updating documentation and making the software more maintainable.

Need of Maintenance

Maintenance is needed to ensure that the software continues to satisfy user requirements.

- Correct Faults
- Improve the design
- Implement enhancements
- Interface with other system
- Migrate legacy software
- Retire software

Fuzzy Logic

Before illustrating the mechanisms which make fuzzy logic machines work, it is important to realize what fuzzy logic actually is. Fuzzy Logic is a superset of conventional (Boolean) logic that has been extended to handle the concepts of partial truth-table values between "completely true" and "completely false." As its name suggests, it is the logic underlying models of reasoning which are approximates rather than exact. The importance of fuzzy logic derives from the fact that most modes of human reasoning and especially common reasoning are approximate in nature.

The essential characteristics of fuzzy logic as founded by Zadeh Lofti are as follows.

- In fuzzy logic, exact reasoning is viewed as a limiting case of approximate reasoning
- In fuzzy logic everything is a matter of degree.
- Any logical system can be fuzzified.
- In fuzzy logic, knowledge is interpreted as a collection of elastic or, equivalently, fuzzy constraint on a collection of variables.
- Inference is viewed as a process of propagation of elastic constraints.

The third statement hence, defines Boolean logic as subset of fuzzy logic. The definitions –

"Fuzzy logic is form of multi-valued logic derived from fuzzy set theory to deal with reasoning that is approximate rather than precise."

Why Fuzzy Logic

Fuzzy logic offers several unique features that make it a particularly good choice for many control problems.

1. It is inherently robust since it does not require precise, noise –free inputs and can be programmed to fail safely if a feedback sensor quits or is destroyed. The output control is a smooth control function despite a wide range of input variations

2. Since the FL controller processes user-defined rules governing the target control system, it can be modified and tweaked easily to improve or drastically alter system performance. New sensors can easily be incorporated into the system simply by generating appropriate governing rules.
3. FL is not limited to a few feedback inputs and one or two control outputs, nor is it necessary to measure or compute rate of change parameters in order of it to be implemented. Any sensor data that provides some indication of a system's action and reaction is sufficient. This allows the sensor of inexpensive and imprecise thus keeping the overall system cost and complexity low.
4. Because of the rule-based operation ,any reasonable number of inputs can be processed (1-8 or more) and numerous outputs (1-4 or more) generated , although defining the rule base quickly becomes

complex if too many inputs and outputs are chosen for a single implementation since rules defining their inter relations must also be defined. It would be batter to break the control system into smaller chunks and use several smaller FL controllers distributed on the system, each with more limited responsibilities.

5. FL can control nonlinear systems that would be difficult or impossible to model mathematically. This opens doors for control systems that would normally be deemed unfeasible for automation.

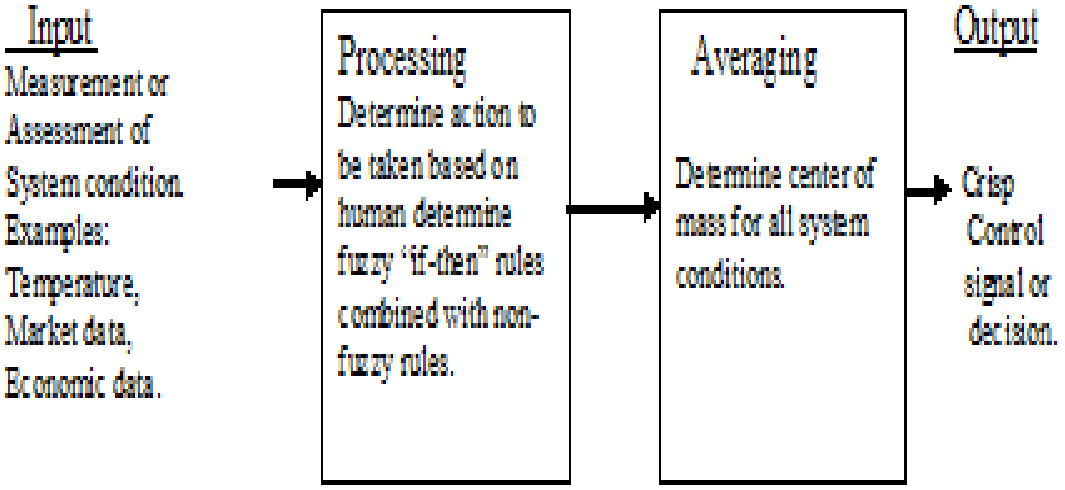
How is Fuzzy Logic used

1. Define the control objectives and criteria: what am I trying to control? What do I have to do to control the system? What kind of response do I need? What are the possible system failure modes?
2. Determine the input and output relationships and choose a minimum number of variables for input to the FL engine.
3. Using the rule base structure of FL, break the control the problem down into a series of IF X AND Y THEN Z rules that define the desired system outputs response for given system input conditions the number and complexity of rules depends on the number of input parameters that are to be processed and number fuzzy variable associated with each parameters. If possible, use at least one variable and its time derivative. Although it is possible to use a single, Instantaneous error parameter without knowing its rate of change, this cripples the system's ability to minimize overshoot for a step inputs.
4. Create FL membership functions that define the meaning of input/output terms used in the rules.
5. Create the necessary pre and post processing FL routines if implementing in software, otherwise program the rules into the FL hardware engine.
6. Test the system, evaluate the results, tune the rules and membership functions, and retest until satisfactory results are obtained.

The Fuzzy Logic Method

The fuzzy logic analysis and control method is, therefore:

1. Receiving of one, or a large number, of measurement or other assessment of conditions existing in some system we wish to analyze or control.
2. Processing all these inputs according to human based, fuzzy "If-Then" rules, which can be expressed in plain language words.
3. Averaging and weighting the resulting outputs from all the individuals' rules into one single output decision or signal which decides what to do or tells a controlled system what to do. The output signal eventually arrived at is a precise appearing, defuzzified, "crisp" value. Please see the following Fuzzy Logic Control/ Analysis Method Diagram:



Fuzzy Perception

A fuzzy perception is an assessment of a physical condition that is not measured with precision, but is assigned an intuitive value. It will be seen below that fuzzy perceptions can serve as a basis for processing and analysis in a fuzzy logic control system. Measured, non-fuzzy data is the input for the fuzzy logic method. Examples: temperature measured by a temperature transducer, motor speed, economic data, and financial markets data etc. Then humans with their fuzzy perceptions and fuzzy rules take over. Human perceptions and rules are placed in-the-loop in the fuzzy logic based control system.

LITERATURE REVIEW

Chandrashekhar Rajaraman, Micheal R Lyu[8] describes some difficulties that one encounters in the testing and maintenance of C++ programs, which may result in unreliable program. Inheritance and polymorphism are the key concept in object oriented programming (OOP), and are essential for achieving reusability and extendibility, but they also make programs more difficult to understand. They tried to show by arguments and by some empirical evidence that widely used complexity metrics like line of code, cyclomatic complexity, and Software metrics may not be appropriate to measure the complexity of C++ programs and those written in other object oriented languages, because they do not address concepts like inheritance and encapsulation. Some measures using a notion from the world of functional decomposition – coupling, are defined for C++ programs. Two of them- Class coupling and AMC and equivalent ones for the three widely used complexity metrics (for comparison) are computed for five C++ programs. Their results shows that the coupling measures correlate better with difficulties of testing and maintenance than the three widely used complexity metrics.

K.K. Aggarwal et. al. [9] describes that Software maintenance is a task that every development group has to face when the software is delivered to the customers' site, installed and is operational. The time spent and effort required for keeping software operational consumes about 40-70% of cost of entire life cycle. They have proposed a four parameter integrated measure of software maintainability using a fuzzy model. They also includes empirical data of maintenance time of projects which has been used to validate the model.

Alain April et. a.[10] have addressed the assessment and improvement of the software maintenance function by proposing improvements to the software maintenance standards and introducing a proposed maturity model for daily software maintenance activities: Software Maintenance Maturity Model (SMMM). The software maintenance function suffers from a scarcity of management models to facilitate its evaluation, management, and continuous improvement. This model addresses the unique activities of software maintenance while preserving a structure similar to that of the CMM maturity model. It is designed to be used as a complement to this model. The SMMM is based on practitioners. Experience, international standards, and the seminal literature on software maintenance.

George E. Stark says that software maintenance is central to the mission of many organizations. Thus, it is natural for managers to characterize and measure those aspects of products and processes that seem to affect cost, schedule, quality, and functionality of a software maintenance delivery. The work answers basic questions about software maintenance for a single organization and discussed some of the decision made based on the answers. Attributes of both the software maintenance process and the resulting product were measured to direct management and engineering attention towards improvement areas, track the improvement over time, and help to make choices among alternatives.

M. Burgin et. al. have described that software reuse is an important and relatively new approach to software engineering. They have focused on development of a methodology and mathematical theory of software metrics for evaluation of software reusability. They have demonstrated that reusability is a form of usability. This allows one to use experience in the development and utilization of software reuse metric. They considered different types and classes of software metrics and compare them. They also studied software metrics and their properties in a formalized context. Their research is oriented at the advancement of software engineering and, in particular, at creation of more efficient reuses metrics.

CONCLUSION

This paper concludes that maintenance is a time consuming and expensive phase of SDLC. We have reviewed the work of many researcher which helps to understand the different techniques to maintain the software. We also concluded that systematic improvement of maintenance requires knowledge of the existing problems, the ability to tailor existing methodologies, and a commitment to monitor the effectiveness of these methodologies.

References

1. Tsun S. Chow, "Software Quality Assur-ance: A Practical Approach," IEEE Computer Society, Worldway Postal Center, Los Angeles, California.
2. J. Martin and J. J. Odell, "Object-Oriented Analysis and Design," Prentice Hall, Englewood Cliffs, New Jersey.
3. O m an P , Hagemeister J , A s h D . A d efi nition and taxonomy f or software maintainability. Technical R epor t 91-08, Software Engineering Test Laboratory, University of Idaho, 1991.
4. Belad L , Lehman M. A n Introduction to Growth Dynamics in Statistical Computer Performance E valuation, Freiberg er W (ed.) . Academic Press: New York NY, 1972; 503–511.
5. Jorgensen M. Experience with the accuracy of software maintenance task effort p rediction models. IEEE Transactions on Software E ngineer ing 1995; 21(8):674–681.

6. Dart S, Christie AM, Brown AW. A case study in software maintenance. R epor t CMU/SE I-93-TR - 8, Software Engineering Institute, Carnegie Mellon University: Pitts burg h PA , 1993.

7. Rikard Land Mälardalen "Software Deterioration And Maintainability – A Model Proposal" in 1995 University Department of Computer Engineer

8. Khairuddin Hashim and Elizabeth Key " A SOFTWARE MAINTAINABILITY ATTRIBUTES MODEL" Malaysian Journal of Computer Science

9. C. van Koten 1 and A.R. Gray 'An application of Bayesian network for predicting object-oriented software maintainability' in 2005 Department of Information Science,University of Otago, P.O.Box 56, Dunedin, New Zealand

10. K.K. Aggarwal et. al. 'Measurement of Software Maintainability Using a Fuzzy Model' Journal of Computer Sciences 1(4):538-542, 2005

