https://doi.org/10.48009/2_iis_2007_238-244

USE OF FUZZY LOGIC IN SOFTWARE DEVELOPMENT

Shradhanand, shardha_nand@yahoo.co.in, NSIT(Formerly Delhi Institute of Technology)
Amarjeet Kaur, uppalz_amar@yahoo.com , NSIT(Formerly Delhi Institute of Technology)
Dr. Satbir Jain , jain_satbir@yahoo.com, Delhi University, New Delhi

ABSTRACT

The systems and software development industry is characterized by a paradigm of project failure. One of the known contributing causes of these project failures is poor requirements engineering and management, which has been repeatedly and widely discussed and documented. But there are other factors also like poor project management practices, poor design strategy and inefficient testing principles also contributing to project failures. And the root cause of all these factors is the fact that we use classical two-valued logic system for decision making. The output of decision making process is either yes or no in two-valued logic system. The Maxim of Uncertainty in Software Engineering (MUSE) states that uncertainty is inherent and inevitable in software development processes and products. It is a general and abstract statement applicable to many facets of software engineering.
The above problems could be easily countered by the Fuzzy logic, because fuzzy logic has ability to deal with uncertainty and multi valued logic e.g. an entity in a problem domain has 0.5 possibility or 0.8 possibility like that to be taken as a class, whereas in classical two valued logic , there is only two possibility values either 0 or 1. So, quantization levels increases from 2 to more than two, consequently the quantization error will reduce and there will be less information loss at early stages of development. Fuzzy logic uses membership functions to incorporate linguistic variables and quantifiers. Fuzzy Logic could also be used in project *estimation* purposes efficiently by gathering size data on previously developed programs. Fuzzy logic based estimation provides reasonably good estimates where new work is like prior experience. Fuzzy logic concepts could also be used at testing phase of software development. As a rule of thumb we can say if some decision making or human communication involve during development process we can use the concept of fuzzy logic to improve s/w development processes and products.
Keywords: Fuzzy Logic, Software Project Management, Object Oriented Modeling, Software Testing, Requirement Engineering.

INTRODUCTION

The systems and software development industry is characterized by a paradigm of project failure (Standish 1995). The situation has been described by Cobb's Paradox (Voyages 1996),which stated "*We know why projects fail, we know how to prevent their failure --so why do they still fail?"* One of the known contributing causes of these project failures is poor requirements engineering and management, which has been repeatedly and widely discussed and documented for at least 10 years (Hooks 1993; Kasser and Schermerhorn 1994; Jacobs 1999; Carson 2001; etc.).

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 *TradingQuantiy is Heavy and RoomTemprature is High*, here *TradingQuantiy* and *RoomTemprature* 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. Zadeh distinguishes between 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.

During the last decade, a considerable number of software development methods have been introduced. Methods aim at creating software artifacts through the application of a large number of rules and all the rules are based on classical 2-valued logic system. For example, the method OMT introduces rules for identifying and discarding object-oriented artifacts such as classes, associations, and part-of and inheritance relations. These methods do not introduce means to model the desired inconsistencies and therefore, aim at resolving inconsistencies whenever they are detected. For example, in object-oriented methods a candidate class is generally identified by applying the rule:
If an entity in a requirement specification is relevant and exist autonomously then select it as a candidate class.

While applying the object-oriented intuition of what a class should be, this rule follows the consistency constraint "an entity is either a candidate class or not a candidate class but not both". In this example, the software engineer has to determine whether the entity being considered is relevant or not for the application domain. The software engineer may conclude that the entity partially fulfils the relevance criterion, and may prefer to define the relevance of an entity, for instance, as substantially relevant. This definition would imply the classification of the entity as a partial class, which is considered as an inconsistent class definition by the current object-oriented methods. Therefore, the consistency constraint in current methods forces the software engineer to take abrupt decisions, such as accepting or rejecting the entity as a class. This results in loss of information because the information about the partial relevance of the entity is not modeled and therefore in the subsequent phases cannot be considered explicitly. Software engineers, however, must be able to defer rejecting artifacts until such decisions can be justified with sufficient confidence.

Fuzzy-logic based heuristics

We think that the number of classification levels in object-oriented methods must be increased. To this aim, the consistency constraints have to be weakened to allow partial classification. On the other hand, having too many levels may not be meaningful for the software engineer. For instance, to express the relevance of an entity with a real number such as 0.75 is not very intuitive. Further, the accuracy of such an expression is highly questionable. This means that one needs to balance intuition and reduction of the classification error.

To represent multiple levels conveniently, the rules of current object-oriented methods have to be enhanced. Consider, for example, the rule Candidate Class Identification:
If an entity in a requirement specification is relevance-value *relevant and can exist* autonomy-value *autonomous in the application domain, then select it as a* relevance-value *relevant candidate class.*

Here, an entity and a candidate class are the artifact types to be reasoned, relevance and autonomy are the properties, and relevance-value and autonomy-value indicate the domains of these properties. For example, relevance-value may represent the set of values {Weakly, Slightly, Fairly, Substantially, Strongly}, and autonomy-value may represent the set of values {Dependently, Partially Dependently, Fully Autonomously}. Multiple classification levels can be expressed using disjoint sets resulting in an abrupt separation of elements between different sets. One of the major problems of disjoint sets is that they do not provide an effective means for capturing the approximate, inexact nature of the software development process. It is therefore worth to investigate other forms of logic than classical logic. In fuzzy logic, the concept of vagueness is introduced by the definition of fuzzy set. A fuzzy set S of a universe of discourse U is characterized by a membership function is $\mu_s : U \rightarrow [0,1]$ which associates with each element y of U a number in the interval [0,1] which represents the grade of membership of y in S. Based on the definition of fuzzy sets, the concept of linguistic variables is introduced to represent a language typically adopted by a human expert. A linguistic variable is a variable whose values, called linguistic values, have the form of phrases or sentences in a natural language. An important issue here is to determine the linguistic values (i.e. Weakly, Strongly). Several researchers have investigated definitions of linguistic values and it has been showed that commonly used English terms could be defined with a reasonable accuracy. Formalizing the rules of current object-oriented methods using fuzzy-logic is an extension of previous work, which modeled the heuristic rules of object-oriented methods using two-valued logic.

FUZZY LOGIC IN REQUIREMENT ENGINEERING

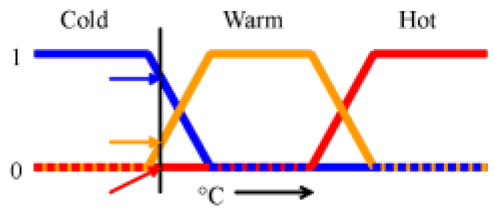
Traditional requirement engineering technique such as functional analysis have lots of draw back with

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 fare 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.
Let us assume that N is the number of quantization levels and A is the maximum value of the signal. Then, the RMS value. of the quantization error is computed as:

$$\bar{\varepsilon} = \frac{A}{2(N-1)} \cdot \sqrt{\frac{1}{3}}$$

It is clear from that RMS value of the quantization error decreases with the increase of the number of quantization levels. In current software development methods, high quantization errors arise from the fact that rules adopt only two quantization levels.

The drawbacks with OOA could be easily countered by the fuzzy logic, because fuzzy fogic has ability to deal with multi valued logic e.g. an entity in a problem domain has 0.5 possibility or 0.8 possibility like that to be taken as a class, whereas in classical two valued logic , there is only two possibility values either 0 or 1. So, quantization levels increases from 2 to more than two consequently the quantization error will reduce and there will less information loss at early stages of development. Fuzzy logic could empower software development especially requirement engineering phase because it can incorporate linguistic variables that is very common in human communication. For instance, a temperature measurement for anti-lock brakes might have several separate membership functions defining particular temperature ranges needed to control the brakes properly. Each function maps the same temperature value to a truth value in the 0 to 1 range. These truth values can then be used to determine how the brakes should be controlled.



In this image, *cold*, *warm*, and *hot* are functions mapping a temperature scale. A point on that scale has three "truth values" — one for each of the three functions. For the particular temperature shown, the three truth values could be interpreted as describing the temperature as, say, "fairly cold", "slightly warm", and "not hot".

Yen et al. present a formal approach for assessing the relative priority by analyzing the customer's trade-off preferences among imprecise conflicting requirements. The relative priority can be derived from the analysis of the customer's trade- off between attributes on which requirements impose constraints based on the marginal rate of substitution in decision science. These priorities are then transformed into numeric priorities of requirements so that they can be used in aggregation of conflicting requirements to resolve conflicts. Requirements trade-off analysis technique proposed by Lee et al. is mainly on the formulation or' vague requirements based on Zadeh's canonical form in test-score semantics and an extension of the notion of soft conditions. The trade-off among vague requirements is analyzed by identifying the relationship between requirements, which could be either conflicting, irrelevant, cooperative, counterbalance, or independent. A compromise overall requirement can be obtained through the aggregation of individual requirements based on the requirements hierarchy. The proposed approach provides a framework for formally analyzing and modeling conflicts between requirements, and for users to better understand relationships among their requirements.

Fuzzy logic could be applied to object oriented modeling. Various fuzzy OO(object oriented) modeling have also been proposed for modeling imprecise requirements, fuzzy objects, and fuzzy typing etc. Gyseghem et al. represent fuzzy information as fuzzy sets and uncertainty by means of generalized fuzzy sets. A generic fuzzy class is introduced to capture fuzziness associated with attributes. Uncertain information is modeled by a kind of generalized fuzzy sets in which each element of the universe is associated with a fuzzy truth value. Graham concentrates on the derivation of unknown values of attributes through the use of A kind-of relation (AKO), generalized modus ponen, and defuzzification techniques. An object is extended to fuzzy object by fuzziness attributes' values and AKO degrees. The AKO degree between classes is assumed to be known by a system, and unknown attributes' values are derived through AKO, generalized modus ponen, and defuzzification techniques. Pedrycz et al. applied fuzzy neural networks to the parametric learning and aggregated the results of matching for the successive attributes. The matching process applies to the corresponding attributes, and these partial results are aggregated giving rise to the overall degree of matching the object to the class. The values of the attributes are characterized using linguistic terms. The realization of inheritance in the considered setting exploits linguistic approximation; the results of approximation are used to identify objects.

Lee proposes a fuzzy OO modeling technique (FOOM) to capture and analyze imprecise requirements through the following two steps:
(1) to identify the possible types of fuzziness involved in the modeling of imprecise requirements, such as classes, rules, attributes, ISA/AKO, and associations and (2) to investigate the potential impacts of incorporating the notion of fuzziness on the features of OO. FOOM uses fuzzy inclusion technique to compute the compatibility degree between a class and an object, and a class and its subclass. Marin et. al. offer a new perspective for representing the fuzzy type associated to a class, tackling the problem of vagueness in the database schema, and defining the fuzzy type (structural and behavioral components). A structural component is a fuzzy set defined over the set of all the possible attributes and the behavioral component is a fuzzy set defined over the set of all the possible methods in the data model. Both the mechanisms of inheritance and instantiation have been modified to take advantage of fuzzy types. The mechanism of instantiation can be permitted to choose the a-cut of attributes of the type to represent objects.

Yazici et al. introduce a semantic data modeling approach for spatiotemporal database application. This approach extends UML (unified modeling language) for handling static and dynamic spatiotemporal information, uncertainty, and fuzziness especially at conceptual level of database design. Three types of uncertainty [30] in the attribute level are dealt with, including incompleteness, null, and fuzzy. The "part" object participating in the aggregation/ composition relationship has a certain membership degree being a part .of the "whole" object. The semantic data modeling approach not only can get the present status of the database, but also can look back into the history of the system or predict its future state when coupled with fuzzy logic.

FUZZY LOGIC IN PROJECT MANAGEMENT

One problem faced by managers who are using project management models is the elicitation of numerical inputs. Obtaining these with any degree of confidence early in a project is not always feasible. Related to this difficulty is the risk of precisely specified outputs from models leading to over commitment. These problems can be seen as the collective failure of software measurements to represent the inherent uncertainties in managers' knowledge of the development products, resources, and processes. It is proposed that fuzzy logic techniques can help to overcome some of these difficulties by representing the imprecision in inputs and outputs, as well as providing a more expert-knowledge based approach to model building. The use of fuzzy logic for project management however should not be the same throughout the development life cycle. Different level of available information and desired precision suggest that it can be used differently depending on the current phase, although a single model can be used for consistency.

One of the important benefits of fuzzy logic for software engineering project management is the flexibility available in terms of the types of input and output variables. Input variables can be expressed as simple fuzzy labels (a *large* number of entities in the data model), fuzzy numbers *(about 250* entities), or using precise values *(265* entities). Similarly, the output can be expressed in the same way, as a label (a *short* development time), fuzzy number (about *400* developer-hours), or precise values (378 developer-hours).

Fuzzy Logic could be used in size estimation purpose. Size estimation can be done in advance to make better plans (to better size the job, to divide the job into separable elements), to assist in tracking progress (can judge when job scope changes, can better measure the work), to learn and build estimating skills. Estimation is an uncertain process no one knows how big the product will be. Estimation can be biased by business and other pressures and earlier the estimate, the less is the

accuracy of estimation. This is how we can use concept of fuzzy logic in size estimation:
1. Gather size data on previously developed programs , 2. Divide the historical product size data into size ranges as very large, large, medium, small, very small, 3. Compare the planned product with these prior products, 4. Based on this comparison, select the size that seems most appropriate for the new product.
Example: You have historical data on 5 programs as follows:
A file utility of 1,844 LOC, a file management program of 5,834 LOC, a personnel record keeping program of 6,845 LOC, a report generating package of 18,386 LOC, and an inventory management program of 25,943 LOC. Suppose the new program has following requirements:
analyze marketing performance by product line, project the likely sales in each product category,
allocate these sales to marketing regions and time periods, produce a monthly report of these projections and the actual results. Now we will see how fuzzy fogic can be used to estimate the size of the new program.
We will establish 5 size ranges, as follows:
log(1844) = 3.266, log(25,943) = 4.414, the difference is 1.148
1/4th this difference is 0.287, the logs of the five ranges are thus spaced 0.287 apart,
the limits or these ranges are at 0.1435 above and below the midpoint of each range.
The 5 size ranges are thus
very small - 1,325 to 2,566: file utility
small - 2,566 to 4,970: no members
medium - 4,970 to 9,626: file management and personnel record program
large - 9,626 to 18,641: report generator
very large - 18,641 to 36,104: inventory management
In comparing the new program to the historical data you make the following judgments:
It is a substantially more complex application than either the file management or personnel programs. It is not as complex as the inventory management program and appears to have significantly more function than the report package.
You conclude that the new program is in the lower end of "very large," or from 18 to 25 KLOC.

Fuzzy logic based estimation provides reasonably good estimates where new work is like prior experience, easy to use and requires no special tools or training. But there are certain limitations with this approach of estimating size. These are**:** it requires a lot of data, the estimators must be familiar with the historically developed programs, it only provides a crude sizing, it is not useful for new program types and not useful for programs much larger or smaller than the historical data.

FUZZY LOGIC IN SOFTWARE TESTING

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.

According to MUSE, 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.

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. An example of assigning confidence values to path selection criteria is given below.

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, that 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 memebership.

CONCLUSION

As uncertainty is inherent and inevitable in software development processes and products. So, software uncertainties should be modeled and managed explicitly. Give its ability to represent differing levels of uncertainty for inputs and outputs whilst still basing inference on the same model, fuzzy logic is well suited to a life-cycle approach to software development process. The ensuing consistency, communicatability, and economy make this an attractive modeling technique for such applications as effort estimation, requirement engineering and testing. The other advantages of data-free (or data-poor) model building, more robust models, and improved communication further enhance the opportunities from using fuzzy logic for software metrics.

Hence, we can say that application of fuzzy logic in software development will definitely improve development processes by considering and in turn reduce information loss and provide greater design alternatives at later stages of development.

REFRENCES

1. M. Aksit and F. Marcelloni, *Deferring Elimination of Design Alternatives in Object-Oriented Methods*, in: Concurrency Practice and Experience, 2000.
2. M. D. Hakel, "How Often is Often", American Psychologist, pp. 533-534, 1968.
3. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, "Object-Oriented Modeling and Design", Prentice-Hall, 1991.
4. F. Marcelloni and M. Aksit, *Reducing Quantization Error and Contextual Bias problems in Software Development Processes by Applying Fuzzy Logic*, Proceedings of the 18th International Conference of the North American Fuzzy Information Processing Society (NAFIPS'99), pp. 268-272, NewYork, June 1999.
5. F. Marcelloni and M. Aksit, *Improving Object-Oriented Methods by Using Fuzzy Logic*, in: ACM Applied Computing Review, 2000.
6. B. Tekinerdogan and M. Aksit, "Modeling Heuristic Rules of Methods", University of Twente, Department of Computer Science, 1998.
7. B. Tekinerdogan, *Synthesis Based Software Architecture Design*, Ph.D. thesis, University of Twente, The Netherlands, March 2000.
8. E. Yourdon, "Modern Structured Analysis", Englewood Cliffs, New Jersey: Yourdon Press, 1989.
9. L.A. Zadeh, "Outline of a New Approach to the Analysis of Complex Systems and Decision Processes", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-3, No.1, pp. 28-44, January, 1973.
10. Graham. Fuzzy objects: Inheritance under uncertainty. In Object Oriented Methods, pages 403-433. Addison Wesley, 1994.
11. N.V. Gyseghem, R.D. Caluwe, and R. Vandenberghe. UFO: Uncertainty and fuzziness in an object oriented model. In proceedings of the IEEE International Conference on Fuzzy Systems, pages 773-778, 1993.
12. J. Lee, N.L. Xue, K.H. Hsu and S.J. Yang. Modeling imprecise requirements With fuzzy objects. Information Sciences, 118:101-119, 1999.

13. W. Pedrycz and Z.A. Sosnowski. Fuzzy object-oriented system design. Fuzzy Sets and Systems, 99:121-134, 1998.
14. L.A. Zadeh. The role of fuzzy logic in the management of uncertainty in expert systems. Fuzzyy Sets and systems, 11:194-227, 1983.
15. L.A. Zadeh. Test-score semantics as a basis for a computational approach to the representation of meaning. Literacy Linguistic Computing, 1:24-35, 1986.
16. L.A. Zadeh. Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic. *Fuzzy* Sets and Systems, 0:111-127, 1997.
17. Brooks. The Mythical Man-Month. Addison-Wesley, Reading, MA, 1975.Alan M. Davis. 201 Principles of Software Development. McGraw Hill, New York, 1995.
18. Frederick P. Brooks. No silver bullet: Essence and accidents of software engineering. IEEE Computer, 20(4):10{19, April 1987.
19. Adele Goldberg and Kenneth S. Rubin. Succeeding with objects : decision frameworks for project management. Addison-Wesley, Reading, MA, 1995.
20. Debra J. Richardson, Stephanie Leif Aha, and T. Owen O'Malley " Specification-based test oracles for reactive systems". In Proceedings of the Fourteenth International Conference on Software Engineering, pages 105-118, Melbourne, Australia, May 1992.
21. Jonathan Lee, Jong-Yih Kuo and Nien-Lin Xue " Current Approaches to Extending Fuzzy Logic to Object-Oriented Modeling".0-7803-7078-3/0l/$l0.00(C)2001 IEEE.