A Mathematical Study of Fuzzy Logic Techniques in Software Engineering Measurements

Er. Kailash Aseri
Ph. D. Research Scholar
Department of Computer Science & Engineering
kailash.aseri@gmail.com

***Abstract*— Estimation models in software engineering are used to predict some important attributes of future entities such as software development effort, software reliability and programmer productivity. Estimation by fuzzy logic techniques is one of the most attractive techniques in software effort estimation field. In this paper I propose a new approach based on reasoning by fuzzy logic to estimate effort .**

Keywords-fuzzy logic, productivity

I. INTRODUCTION

Fuzzy Logic

In 1948, Alan Truing wrote a paper, which marked the beginning of a new era, the era of the intelligent machine. To allow computers to mimic the way humans think, the theories of fuzzy sets and fuzzy logic was created. Classical logic deals with crisp knowledge where statements can only be either true or false, while fuzzy logic deals with vaguely formulated or uncertain knowledge.
Dr. Lotfi Asker Zedeh first used the term fuzzy in the engineering journal, "Proceedings of the IRE" in 1962. Fuzzy sets were introduced by Zadeh (ZADE65) as an extension of the classical nation of the set. Fuzzy sets are sets whose elements have degrees of membership. Fuzzy sets generalize classical sets, since the indicator functions of classical sets are special cases of the membership functions of fuzzy sets, if the latter only take values O or 1. In fuzzy set theory, classical sets are usually known as crisp sets. Fuzzy logic is a methodology, based on fuzzy set theory, classical sets are usually known as crisp sets. Fuzzy logic is a methodology, based on fuzzy set theory to solve problems, which are too complex, to be understood quantitatively (ZADE65). Fuzzy logic is a superset of conventional logic that has been extended to handle the concept of partial truth. Fuzzy logic can be thought of as the application side of fuzzy set theory. It is an effective technique to solve uncertainties due to imprecise data.
Fuzzy Number

A fuzzy number is a quantity whose value is imprecise, rather than exact as in the case of ordinary single valued numbers. A fuzzy number is represented by a membership function, whose domain is a fuzzy set. The membership function associates a real number (0, 1) with each point in the fuzzy set, called degree of uncertainty or grade of membership. The membership μ A (x) of an element x of a classical set A, as subset of the universe x, is defined by :

$$\mu_A(x) = \begin{cases} 1 & iff \quad x \in A \\ 0 & iff \quad x \notin A \end{cases} \qquad 1.1$$

In many respects, fuzzy numbers depict the physical world more realistically than single valued numbers. Suppose we are driving along a highway where the speed limit is 80 km/hr, we try to hold the speed at exactly 80 km/hr, but our car lacks cruise control, so the speed varies from moment to moment. If we note the instantaneous speed over a period of several minutes and then plot the result in rectangular co-ordinates, the resultant curve may look like one of the curves shown below; however, there is no restriction on the shape of the curve. The curves in the below Figure show a triangular fuzzy number, a trapezodial fuzzy number, and bell shaped fuzzy number.

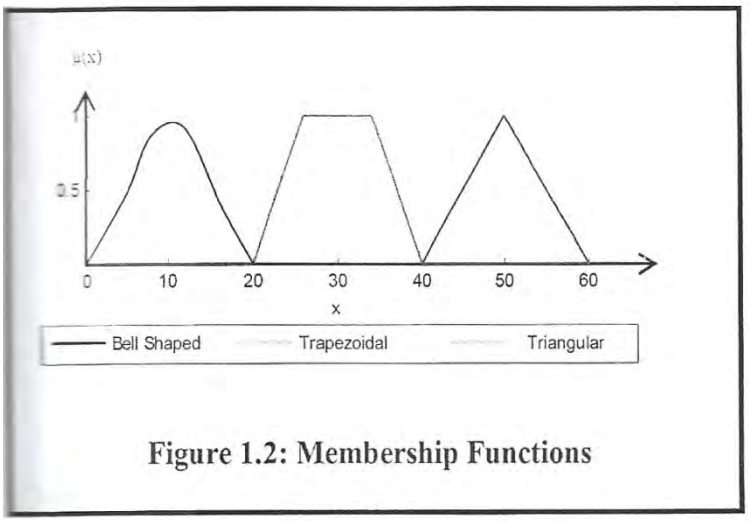
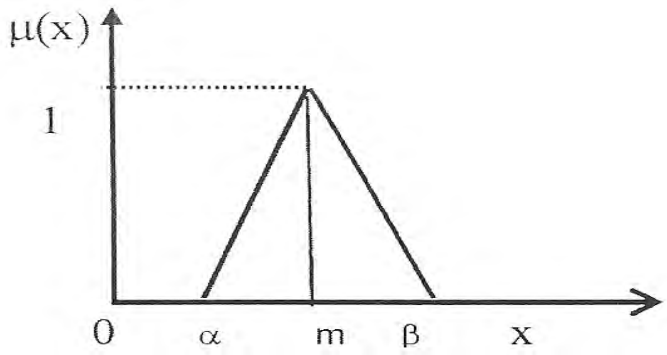


Figure 1.2: Membership Functions

A triangular fuzzy number (FTN) is described by a triplet (, m,), where m is the model value, and are the right and left boundary respectively (figure 1.3). The membership function (x) for TFN is defined as :

$$\mu_A(x) = \begin{cases} 0, & x \le a \\ \dfrac{x-a}{m-a}, & a \le X \le m \\ \dfrac{b-x}{b-m}, & m \le X \le b \\ 0, & X \ge b \end{cases} \qquad 1.2$$



Fuzziness:

Fizziness is explored as an alternative to randomness for describing uncertainty. Fuzziness relates to the un-sharp boundaries of the parameters of the model. Fuzziness of TFN (, m,) is defined as [MUSI00] :

$$\text{Fuzziness of TFN (F)} = \frac{\beta - a}{2m}, \quad 0 < F < 1 \qquad 1.3$$

The higher the value of fuzziness, the fuzzier is TFN. The value of fuzziness taken depends upon the confidence of the estimator. A confident estimator can take a smaller value of F. Let (m, 0) internally divide the base of the triangle in ratio k : 1, where k is real positive number. So that,

$$m = \frac{a + k\beta}{k + 1} \quad 1.4$$

As by definition of fuzziness

$$F = \frac{\beta - a}{2m} \quad 1.5$$

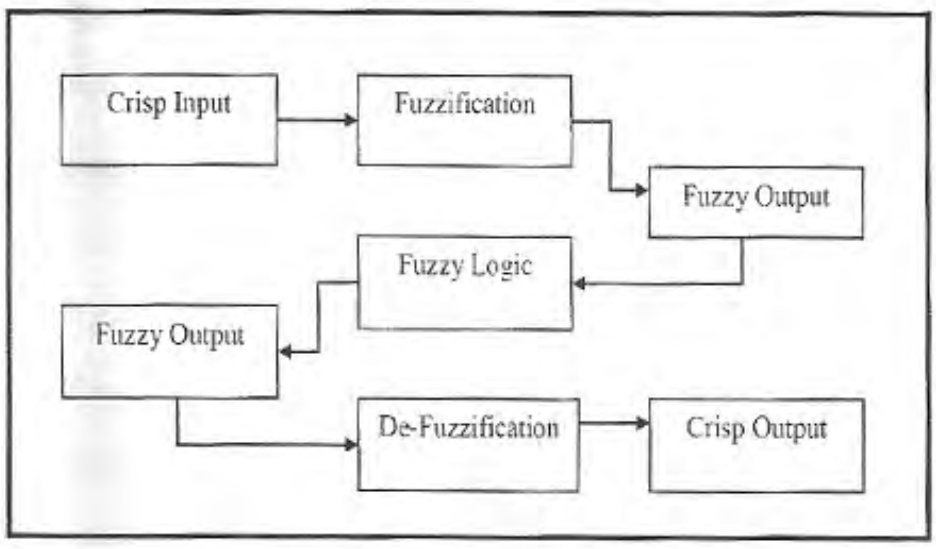
So

$$a = \left(1 - \frac{2kF}{k+1}\right) * m \text{ and } \beta = \left(1 + \frac{2F}{K-1}\right) * m \quad 1.6$$

Fuzzy Logic Process

The fuzzy logic process involves mainly following three sub-processes :

- Fuzzification
- Application of fuzzy logic
- Defuzzification



Fuzzification : Fuzzification is a process whereby crisp values are converted to fuzzy values, represented by membership function. First step is to select appropriate linguistic/fuzzy system variables followed defining fuzzy sets to represent concepts for each fuzzy variable. Assignment of fuzzy values may be done either by intuition or by some algorithm. Intuition involves contextual and semantic knowledge about an issue.

- Application of fuzzy logic : Identification of rules to relate inputs to outputs, selecting techniques for correlation of inputs to outputs and composition of rules.
- Defuzzification : Defuzzification is the process of converting fuzzy values to crisp values. Several techniques have been proposed for defuzzification. The simplest method is the

Maximum method is which a fuzzy set with maximum membership function is selected. Some popular defuzzification methods are discusses below :

1. Centroid : Centroid defuzzification returns the center of area under the curve. If you think of the area as a plate of equal density, the centroid is the point, about which this shape would balance.
2. Bisector : The bisector is the vertical line that will divide the region into two sub-regions of equal area. It is sometimes, but not always coincident with the centroid line.
3. Middle, Smallest, and Largest of Maximum : MOM, SOM and LOM stand for middle, smallest and largest of maximum, respectively. These three methods key off the maximum value assumed by the aggregate membership function. If the aggregate membership function has unique maximum, then MOM, SOM and LOM all take on the same value.
4. Miscellaneous criteria : We can formulate criteria that are not directly related to any theoretical concepts or foundations, but that are of more practical importance [LEEK99].

There is, however, no simple answer to the question 'which of these methods is the right one?' However, if you want to get started quickly the centroid method is the best option. Later you can always change your defuzzification method to see if another method works better. The fuzzy logic technique used in this work is based on formulating individual criteria for defuzzification based on miscellaneous criteria described above.

REVIEW OF LITERATURE

SOFTWARE COST ESTIMATION

Software cost estimation is the process of predicting the effort required to develop a software system. In the early days of computing, software costs constituted a small percentage of the overall computer-based system cost. An order of magnitude error in estimates of software cost had relatively little impact. Today, software is the most expensive element of virtually all computer-based systems. A large cost estimation error can make the difference between profit and loss. Cost overruns can be disastrous for the developer. Of the three principal components of software cost i.e. hardware, travel and training, and effort costs, the effort cost is dominant. Software cost estimation starts at the proposal state and continues throughout the life of a project.

2.2.1 Characteristics of Software Cost Estimation

The following are four important characteristics of software cost estimation :

* Budgeting : provides accuracy of the overall estimate of spending throughout the project is the most desired capability.
* Trade off and risk analysis : illuminates the cost and schedule sensitivities of software project decisions like staffing, tools, code reuse, etc.
* Project planning and control : provides cost and schedule breakdowns by component, stage and activity.
* Software improvement investment analysis : is important additional capability is to estimate the costs as well as the benefits of such strategies as tools, reuse and process maturity.

2.2.2 Software Development Effort Estimation:

Software development effort estimates are the basis for project bidding, budgeting and planning. These are critical practices in the software industry because poor budgeting and planning can have dramatic consequences. When budgets and plans are too pessimistic, business opportunities can be lost, while over-optimism may lead to significant losses.

2.2.2.1 Software Sizing:

Software sizing is used to estimate the size of a software application or component in order to implement other software management activities. Size is an inherent characteristics of a piece of software just like weight is an inherent characteristic of a tangible object. The accuracy of a software project estimate depends on a number of factors:

* The degree to which the planner has properly estimated the size of the product to be built.
* The ability to translate the size estimate into human effort, calendar time and money.
* The degree to which the project plan reflects the abilities of the software team.
* The stability of product requirements and the environment that supports the software engineering effort.

In the context of project planning, size refers to a quantifiable outcome of the software project. With a direct approach, size is measured in Lines of Code (LOC). With an indirect approach, Function Points (FP) represents size. Although LOC and FP estimation are distinct estimation techniques, both have a number of characteristics in common.

2.2.2.2 Techniques of Software Cost Estimation:

Boehm in 1981 [BOEH81] discussed seven techniques of software cost estimation, which are given in Table 2.1 Algorithm cost models are discussed in detail due to their widespread use and importance. Algorithm models, also called parametric models, are designed to provide some mathematical equations. An algorithmic cost model can be built by analyzing the costs and attributes of completed projects and finding the closest fit formula to actual experience. LOC and function point based models are algorithm models. Some prevalent algorithm techniques are discussed in this section.

Table 2.2 : Cost Estimation Techniques

Technique	***Description***
Algorithmic Cost Modeling	A mode is developed using historical cost information that relates some software metric (usually its size) to the project cost. An estimate is made of that metric and the model predicts the effort required.
Expert	Several experts on the proposed software development techniques and the application domain are consulted. Each one of them estimate the project cost. These estimates are compared and discussed. The estimation process iterates until an agreed estimate is reached.
Estimation by analogy	This technique is applicable when other projects in the same application domain have been completed. Cost of a new project is estimated by analogy with these completed projects. Mayers [MYER 89] gives a clear description of this approach.
Parkinson's Law	Parkinson's Law states that work expands to fill the time available. The cost is determined by available resources rather than objective assessment. If the software has to be delivered in 1 month and 5 people are available, the effort required is estimated to be 60 person-months.
Pricing to win	The software cost is estimated to be whatever the customer offers to spend on the project. The estimated effort depends on the customer's budget and not on the software functionality.
Top-down Estimation	Cost estimate is established by considering the overall functionality and how that functionality is provided by interacting sub-functions. Cost estimates are made based on the logical function rather than the components.
Bottom-up Estimation	The cost of each component is estimated. All these costs are added to produce a final cost estimate.

In 1981, Bailey and Basil i [BAIL 81] presented a model generation process, which permits the development of a resource estimation model for any particular organization. They presented the following model for effort estimation:

Effort = 0.73 (size) 1.16 + 3.5 2.1

The model is based on data collected by an organization, which captures its particular environmental factors and the differences among its particular projects. The process provides the capability to produce a model tailored to the organization that can be expected to be more effective than any model originally developed for another environment. It is demonstrated using data collected from the software engineering laboratory at NASA/Goddard Space Flight Centre.

Effort = a (size)b 2.2

Constructive Cost Model (COCOMO) is a regression-based software cost estimation model developed by Barry Boehm [BOEH 81] in 1981. This model was developed on 63 software projects. The model helps in defining the mathematical relationship between the software development time, the effort in person-months and the maintenance effort. It is considered the most plausible, best known and the most cited of all traditional cost prediction models. COCOMO can be used to calculate the amount of effort and the time schedule for software projects. This model structure is classified based on the type of projects to be handled. The model structure is of the form:

Where, a and b are model parameters that depends on project characteristics. Normally the model parameters are fixed for these models based on the software project size [BOEH81, BOEH95]. Projects fall into three categories : organic, semidetached and embedded, characterized by their size as shown in Table 2.2

Table 2.2 COCOMO (Overview of Categories)

Project Type	Size	Deadline/Cosntraints
Organic	Small	Not Tight
Embedded	Large	Tight
Semidetached	Medium	Medium

There have been a few different versions of COCOMO basic, intermediate and detailed. Parameters for basic model that uses only source size are given in table 2.3.

Table 2.3 : COCOMO' 81 Parameter (Basic Model)

Parameter	Organic	Semidetached	Embedded
A	2.4	3.0	3.6
B	1.05	1.12	1.20

The intermediate version of COCOMO' 81 model is an extension of the basic version, which allows generalized estimates using only software size and the project model. In addition to size, the intermediate version takes into account 15 other cost drivers, which are generally related to the software environment. Each cost driver is measured using a rating scale of six linguistic values : very low, low, nominal, high, very high, extra high. The COCOMO' 81 intermediate model is more accurate than the basic COCOMO model. The model structure takes the form :

Effort = a (size) b * c 2.3

Where a, b are parameters similar to basic version, and c is the product of the 15 cost drivers. Parameters and cost drivers of the COCOMO'81 intermediate model are shown in Table 2.4 and Table 2.5 respectively.

The Detailed COCOMO Model introduces two more capabilities.

*** Phase-sensitive effort multipliers :** The detailed COCOMO model applies the same cost drivers used in the Intermediate version to each sub-phase of development.

*** Three-level product hierarchy:** The product is divided in three-level product hierarchy i.e. system, subsystem, and module.

Table 2.4: COCOMO' 81 Parameters (Intermediate Model)

Parameter	Organic	Semidetached	Embedded
A	3.2	3.0	2.8
B	1.05	1.12	1.20

Table 2.5: COCOMO'81 Cost Drivers

Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Analyst Capability	1.46	1.19	1.00	0.86	.71	-
Experience	1.29	1.13	1.00	0.91	0.82	-
Product Complexity	0.70	0.85	1.00	1.15	1.30	1.65
Database Size	-	0.94	1.00	1.08	1.16	-
Language Experience	1.14	1.07	1.00	0.95	-	-
Modern Prog. Practices	1.24	1.10	1.00	0.91	0.82	-
Programmer Capability	1.42	1.17	1.00	0.86	0.70	-
Software Reliability	0.75	0.88	1.00	1.15	1.40	-
Reqd. Development Schedule	1.23	1.08	1.00	1.04	1.10	-
Storage Constraint	-	-	1.00	1.06	1.21	1.56
Time Constraint	-	-	1.00	1.11	1.30	1.66
Software Tools	1.24	1.10	1.00	0.91	0.83	-
Turnaround Time	-	0.87	1.00	1.07	1.15	-
Virtual Machine Volatility	-	0.87	1.00	1.15	1.30	-

Larry Putnam developed the Software Life-cycle Model (SLIM) in the late 1970s (PUTN 92), according to which development effort is given by the following equation :

E = 0.3945B 2.4

Where

E = Development effort, and

B = Total Lifetime Effort in Man Months, which is calculated by :

$$B = \frac{S^3}{T^4 C^3} \qquad 2.5$$

Where

T = Required development time in years, and

C = Technology Constant

S = Size estimate in LOC, which is calculated as :

$$S = C B^{1/3} T^{4/3} \qquad 2.6$$

Alla F. Sheta [ALAA06], introduced three model structures to estimate the effort required for the development of software projects using Genetic Algorithms (GAs), which are modified versions of the famous COCOMO model, designed to explore the effect of the adopted software development methodology on effort computation. The performance of the developed models was tested on the NASA software project dataset. The three models are

I. GA Model

Effort = 4.9067 (KLOC) ^{07311}

II. Model 1

Effort = 3.1938 (KLOC) ^{08209}

III. Model 2

Effort = 3.3602 (KLOC) ^{08116} -4.4524 (ME) + 17.8025

Where, ME is methodology factor for the given project. Methodology for design of a project may be classified in various categories [BAIL 81].

- Tree Charts
- Top-down Designs
- Design Formalisms
- Formal Documentation
- Code Reading
- Chief Programmer Teams
- Formal Test Plans Unit
- Development Folders
- Formal Training

A.J. Albrecht [ALBR79, ALBR 84] introduced function points in an attempt to use as factor independent of programming language for measure of size. Since 1986, when the International Function Point User Group (IFPUG) was set up, several versions of the Function Point Counting Practices Manual have been published by IFPUG [SPR090, IFPU94]. FPA begins with the decomposition of a project or application into its data and transactional functions.

LIMITATIONS

The present work identifies rationalization of fuzzy logic techniques in software engineering measurements. Effective utilization of fuzzy logic technique results in budgeting, accurate time schedules, increased productivity and quality, and the lesser costs is the long run. It shows that fuzzy logic can be applied

to estimate almost every software attribute, more accurately than non-fuzzy approaches. The selection of membership function, fuzziness in raw data and Defuzzification method used also affects the quality of estimation and precision of estimates.

CONCLUSION

The outcome of present work and scope for future work. Today the customer is quality conscious and software companies face hard competition to deliver good quality software at a low cost. The models developed here are sure to fulfill this need. The approach used in the thesis represents a substantive departure from the conventional fuzzy logic techniques. It is recommended that triangular fuzzy numbers can represent almost all linguistic software variables. The results shows remarkable improvements in the existing models and are certain to reduce the development cost of software.

REFERENCES

[1] Burnstein, I., Suwannasart, T., Carlson, R.C., Developing a testing maturity model : Part-II, In Crosstalk Journal. Department of Defense: UT, September 1996.

[2] McCabe, T., A Complexity Measure, IEEE Transactions on Software Engineering, 2 (4) : 308-320, 1976.

[3] McCall, J.A., Richards, P.K. and Walters G.F., Factors in Software Quality, Vol. I, III, III, AD/A-049-014/015/055, Springfield, V.A. : National Technical Information Service, 1977.

[4] Boehm B. Abts and Chulani S. Software Development Cost Estimation Approaches – A Survey, Technical Report USC-CSE-2000-505, University of Southern California – Center for Software Engineering, USA, 177-205, 2000.

[5] Chuk Yau, Raymond H.L. Tsoi, Assessing the Fuzziness of General System Characteristics in Estimating Software Size, IEEE, 189-193, 1994.

[6] Chidamber, S., Kemerer, C., 1994. A Metrics Suite for Object – oriented Design, IEEE Transactions on Software Engineering, Vol 20, Issue 6, pp : 476-493.

[7] Clements, P.C., Northrop, L.M., 1996. Software Architecture : An Executive Overview (CMU/SEI-96-TR-003). Pittsburgh, PA : Software Engineering Institute, Carnegie Mellon University.

[8] Conrow, E.H., Shishido, P.S., Implementing risk management on software intensive projects, IEEE Software, 14 (3) 83-89, 1997.

[9] Cote, M., Suryn, W. Georgiadou, E., 2006, Software Quality Model Requirements for Software Quality Engineering, Proceedings of Software Quality Management and INSPIRE Conference, pp : 31-50.

[10] Devooght, J., Model uncertainty and model inaccuracy, Reliability Engineering and System Safety, 59, 171-185, 1998.

[11] Devanbu, P. Karstu, S., Melo, W., Thomas, W., Analytical and Empirical Evaluation of Software Reuse Metrics, Proceedings of 18th International Conference on Software Engineering, Berlin, Germany, pp : 189-199, 1996.

[12] DIJKSTRA, E.W. “The humble programmer”, Comm. ACJ 15, 10, 859-866, October, 1972.

[13] Doolan, E.P., Experience with Fagan’s inspection method, Software Practice Experience, 22, 173-182, 1992.

[14] Dromey, R.G., A Model for Software Product Quality, IEEE Transactions on Software Engineering, Vol. 21, Issue 2, pp : 146-162, 1995.

[15] Easterbrook, S., Callahan, J., Formal methods for verification and validation of partial specifications : a case study, Journal of Systems software 40, 199-210, 1998.

[16] Ebenau, R.G., Predictive quality control with software inspection, in software inspection : an industry best practice, IEEE computer society press, Silver Spring, MD, pp. 147-156, 1996.

[17] Ebnert, C., Rule-bases fuzzy classification for software quality control. Fuzzy sets syst. 63, 3, 349-358, may 1994.

[18] Emilia Mendes, Nile Mosley, Web Cost Estimation : An introduction, web engineering : principles and techniques, Idea Group, Inc., Ch. 8, 2005

[19] Engel, A., Barad, M., A methodology for modeling VVT risks and costs, Systems engineering journal 6 (3), 135-151, Wiley inter Science, 2003, online ISSN : 1520-6858, Print ISSN: 1098-1241.

[20] Engel, A., Shachar, S., Measuring and potimizing systems quality costs and project duration, systems engineering journal 9 (3), 259-280, 2006.