International Journal of Electronics Engineering, 1(2), 2009, pp. 239-245

A Fuzzy Model for Early Software Fault Prediction Using Process Maturity and Software Metrics

Ajeet Kumar Pandey & N. K. Goyal

Reliability Engineering Centre, IIT Kharagpur, INDIA

Abstract: *Knowing the faults early during software development helps software manager to optimally allocate resources and achieve more reliable software within the time and cost constraints. A model is proposed in this paper to predict total number of faults before testing using a fuzzy expert system. The proposed model predicts number of faults at the end of each software development phase using reliability relevant software metrics and the level of developer's Capability Maturity Model (CMM) level. This paper illustrates how fuzzy expert system can predict the number of faults in the software and thereafter reliability of the software. The proposed model has been applied to the various project data and the results show that prediction results are quite realistic.*

Keywords*: Early Software Fault Prediction, Number of Faults, CMM Levels, Fuzzy Expert System, Software Metrics.*

1. INTRODUCTION

In the last four decades, human dependency on software in their daily lives has increased so much that today it is difficult to imagine living without devices controlled by software. Software has become an integral part of most of the application domains including medical applications, power plants, air traffic control and railway signaling. The development of these software applications is challenging, because system engineers have to deal with a large number of quality requirements such as safety, security, availability, reliability, maintainability and performance. The human dependence on software gives rise to the possibility of crises from its failure. The impact of these failures ranges from malfunctioning of home appliances to economic damage to loss of lives. Therefore, there is a growing need ensure the reliability of software system. Moreover, it is well known that earlier a reliability problem can be identified, the better and more cost effectively this problem can be fixed. Therefore, there is a need to predict these reliability indices early during software development.

IEEE defines software reliability as "the probability of a software system or component to perform its intended function under the specified operating conditions over the specified period of time" [1]. In other words this can also be defined as "the probability of failure-free software operation for a specified period of time in a specified environment". Software reliability is generally accepted as the key factor of software quality since it quantifies software failures, which makes the system inoperative or risky [2]. A software failure is defined as "the departure of external result of program operation from requirements", whereas a fault is defined as "the defect in the program that, when executed under particular conditions, causes a failure" [3]. To further elaborate, a software fault is a defective, missing, or extra instruction or set of related instructions that is the cause of one or more actual or potential failures. This can be summarized to say that execution of a fault results in failure of software.

Software reliability has roots in each step of the requirements, design & coding process [4] and can be improved by inspection and review of these steps. Also this can be accurately assessed only after the testing or after the product completion. Generally, software reliability can be estimated or predicted using various available software reliability models [3, 5] using failure data collected during testing. This becomes too late and sometimes infeasible for taking corrective actions. The solution to this problem is to predict the software reliability early stage of development process i.e. before testing. This early reliability information can help in project management in reducing the development cost by reducing the amount of the rework.

Since the failure data is not available in the early phases of software life cycle, we have to dependent on the information such as reliability relevant software metrics, developer's maturity level, and expert opinions. To our knowledge, these issues are not properly addressed in the software reliability engineering literature. This paper proposes a comprehensive framework to gather the reliability relevant information from early phase of software development life cycle, processing it, and integrating it with the fuzzy logic system to predict the number of faults before testing. Rest of the paper is organized in the following way. Section 2 presents literature survey related with the problem. Section 3 describes the proposed model. Section 4 provides

*Corresponding Author: ajeet.mnnit@gmail.com

a framework of fuzzy logic system for early fault prediction. Section 5 contains the case studies and results whereas conclusions are presented in Section 6.

2. RELATED WORKS

A lot of efforts have been made for software reliability prediction and assessment using various models [3, 5]. Gaffney and Davis [6, 7] of the Software Productivity Consortium developed the phase-based model. It makes use of fault statistics obtained during the technical review of requirements, design, and the coding to predict the reliability during test and operation. One of the earliest and well known efforts to predict software reliability in the earlier phase of the life cycle was the work initiated by the Air Force's Rome Laboratory [8]. For their model, they developed prediction of fault density which they could then transform into other reliability measures such as failure rates. To do this the researchers selected a number of factors that they felt could be related to fault density at the earlier phases. Agresti and Evanco [9] have presented a model to predict defect density based on the product and process characteristics for Ada program. There are many papers advocating statistical models and software metrics [10, 11]. Most of them are based on size and complexity metrics. In order to achieve high software reliability the number of faults in delivered code should be reduced. The faults are introduced in software in each phase of software life cycle and these faults pass through subsequent phases of software life cycle unless they are detected through testing or review process. Finally, undetected and uncorrected faults are delivered with software. In order to achieve the target software reliability efficiently and effectively, faults should be identified at early stages of software development process. During early phase of software development testing/field failure data is not available. Therefore, the prediction is carried out using various factors relevant to reliability. A study was conducted by Zhang and Pham [12] to find the factors affecting software reliability. The study found 32 potential factors involved in various stages of the software life cycle. In another recent study conducted by Li and Smidt [13], reliability relevant software engineering measures have been identified. They have developed a set of ranking criteria and their levels for various reliability relevant software metrics, present in the first four phases of software life cycle. Recently, Kumar and Misra [14] made an effort for early software reliability prediction considering the six top ranked measures given by [13] and software operational profile. Sometimes, it may happen that some of these top ranked measures are not available, making the prediction result unrealistic. Also they have considered only product metrics and ignored process metrics that have a great impact on software reliability [15].

The Capability Maturity Model (CMM) has become a popular and widely accepted methodology to develop high quality software within budget and time [16]. For example, as a software unit at Motorola improved from CMM level 2 to level 5, the average defect density reduced from 890 defects per million assembly-equivalent lines of code to about 126 defects per million assembly-equivalent lines [17]. In an empirical study using 33 software products developed over 12 years by an IT company, Harter *et al.* [18] found that 1% improvement in process maturity resulted in 1.589% increase in product quality. In another study, Krishnan and Kellner [16] found process maturity and personnel capability to be significant predictors (both at the 10% level) of the number of defects. From the above literature we have observed that

i) Software reliability is a function of the number of the remaining faults.
ii) Early fault prediction is more useful for both software professionals and the developing organization.
iii) Software metrics plays a vital role in early fault prediction in the absence of failure data.
iv) Process maturity has a great impact on the software reliability.

Review of literature indicates that traditional models have not considered the both software metrics and development process maturity, for the early fault prediction. Therefore this paper proposes a model for early software fault prediction considering software metrics and process maturity together.

3. PROPOSED MODEL

Early faults prediction attracts both software professional as well as management as it provides an opportunity for the early identification of software quality, cost overrun and optimal development strategies. During the requirements, design or coding phase predicting the number of faults can lead to mitigating actions such as additional reviews and more extensive testing [7]. The model considers two most significant factors, software metrics and process maturity together, for fault prediction. The model architecture is shown in Figure 1. Software metrics can be classified in three categories: product metrics, process metrics, and resources

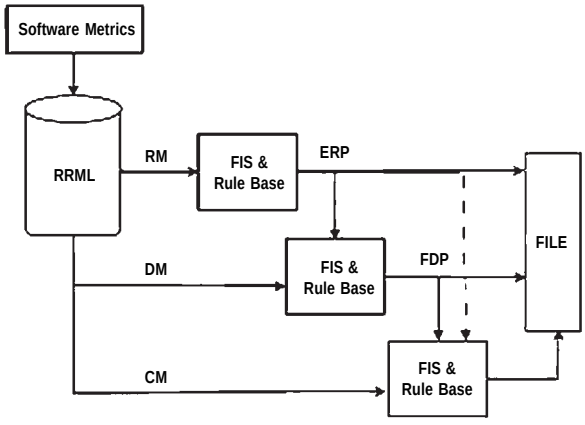

Figure 1: Early Fault Prediction Model

metrics [19]. Product metrics describe characteristics of the product such as size, complexity, design features, performance and quality level etc. Process metrics can be used to improve software development process and maintenance. Resources metrics describe the project characteristics and execution. Approximately thirty software metrics exist, which can be associated with different phases of software development life cycle. Among these metrics some are significant predictor to reliability [13].

The CMM framework includes 18 key process areas such as quality assurance, configuration management, defect prevention, peer review, and training [15]. A software process is assigned the highest maturity level if the practices in all 18 key process areas of the CMM are adopted. The CMM practices aid in reducing defect injection and in early identification of defects. As a consequence, the number of errors detected in testing and remaining in the delivered software will become lesser [17].

The proposed model maintains a reliability relevant metric list (RRML) from various available software metrics. The model has considered three requirements metrics (RM) i.e. Requirements Change Request (RCR), Review, Inspection and Walk through (RIW), and Process Maturity (PM) as input to the requirements phase. Similarly at design phase three design metrics (DM) i.e. design defect density (DDD), fault days number (FDN), and data flow complexity (DC) have considered as input. Two coding metrics (CM) such as code defect density (CDD) and cyclomatic complexity (CC) have been taken as input at coding phase. The outputs of the model will be the number of faults at the end of Requirements Phase (FRP), number of Faults at the end of Design Phase (FDP), and number of Faults at the end of Coding Phase (FCP).

4. IMPLEMENTATION

The model is implemented in MATLAB utilizing fuzzy logic toolbox. The basic steps of the model are identification of reliability relevant input/output variables, development of fuzzy profile of these input/output variables, defining relationships between inputs and output variables and fault prediction at the end of each phase of software life cycle using fuzzy inference system (FIS). These basic steps can be grouped into three broad phases as follows: (1) Early information gathering phase, (2) Information processing phase, and (3) Fault prediction phase.

4.1. Information Gathering Phase

The quality of the fuzzy approximation depends mainly on the quality of information collected (subjective knowledge) and expert opinion. The information gathering phase is often considered the most vital step in developing a fuzzy logic system. The development of fuzzy profiles for identified input/output variables and fuzzy rules are assumed as building blocks of the fuzzy inference system and includes three steps as discussed below.

4.1.1. Identify the Input and Output Variables

Appendix A, shows the list of reliability relevant metrics with their fuzzy profile which can be considered at each phase of software life cycle. These reliability relevant metrics can be identified using [13], [20]. We have identified total eight input variables and three output variables for the purpose of early fault prediction as shown in Table 1. Input variables are the reliability relevant software metrics and output variables are the number of faults at the end of each phase. In order to justify the selection of these software metrics in the proposed model, a regression analysis [21] is performed for different metrics and corresponding correlation coefficient has been identified. Besides this, we have also considered other factors such as time, ease, and cost in finding these software metrics.

Table 1
Model Input/Output Variables

Phase	*Input Variables*	*Output Variables*	*No. of Rules*
Requirement	RCR, RIW, PM	FRP	(3)^3 = 27
Design	FRP, DDD, FDN, DC	FDP	(5)^1 * (3)^3 = 135
Coding	FRP, FDP, CDD, CC	FCP	(5)^2 * (3)^2 = 225

4.1.2. Develop Fuzzy Profile of Identified Variables

This is the first step in incorporating human knowledge into engineering systems in a systematic and efficient manner. The data, which may be useful for selecting appropriate linguistic variable, is generally available in one or more forms such as expert's opinion, software requirements, user's expectations, record of existing field data from previous release or similar system, etc. [22].

Input/output variables gathered at the previous steps are fuzzy in nature and is characterized by membership function. We have considered either triangular or trapezoidal membership function [23], [24], for each variable. Fuzzy membership functions are generated utilizing the linguistic categories such as Very Low (VL), Low (L), Moderate (M), High (H) and Very High (VH), identified by a human expert to express his/her assessment. Figure 2-12, shows membership function and fuzzy profiles of all the selected input/output variables for visualization purpose.

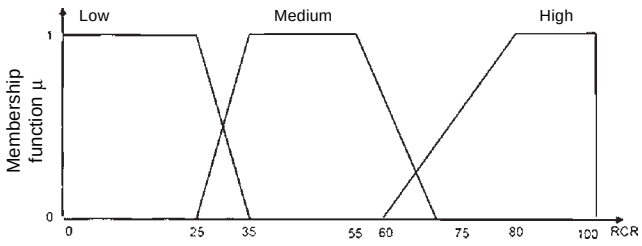

Figure 2: Fuzzy Profile of RCR

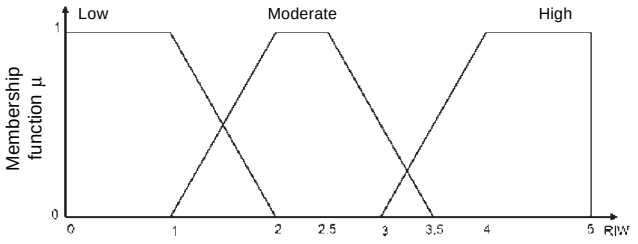

Figure 3: Fuzzy Profile of RIW

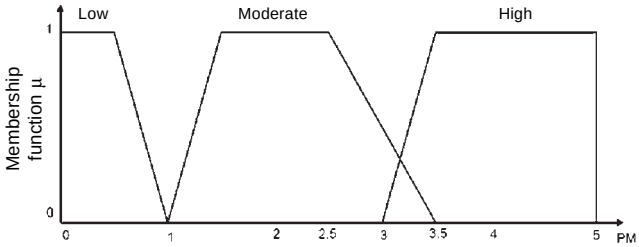

Figure 4: Fuzzy Profile of PM

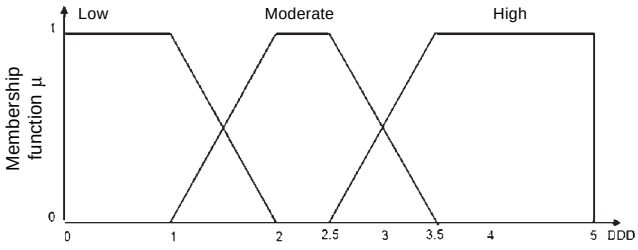

Figure 5: Fuzzy Profile of DDD

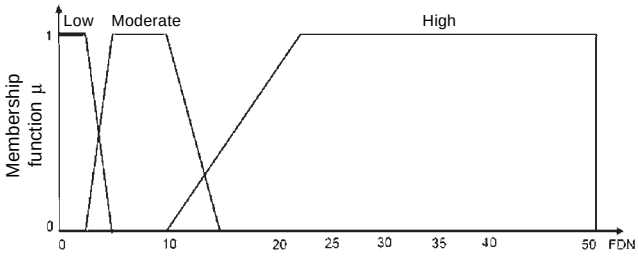

Figure 6: Fuzzy Profile of FDN

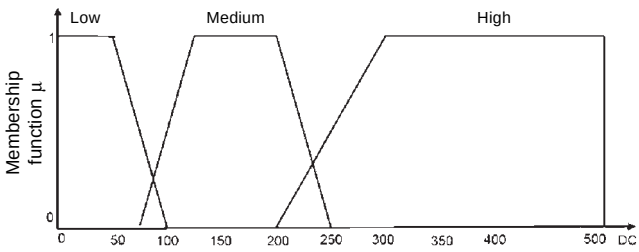

Figure 7: Fuzzy Profile of DC

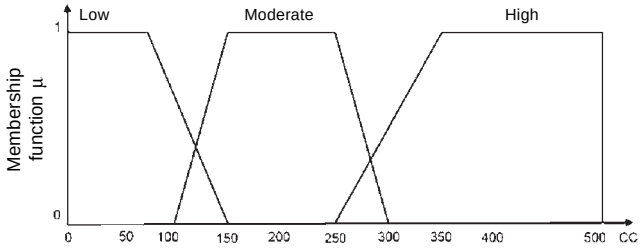

Figure 8: Fuzzy Profile of CC

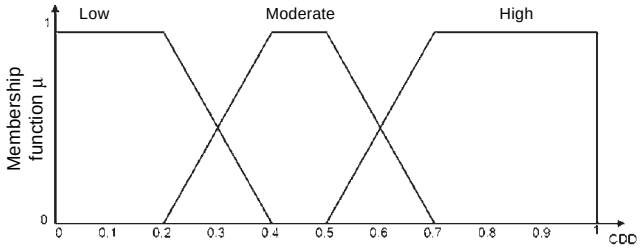

Figure 9: Fuzzy Profile of CDD

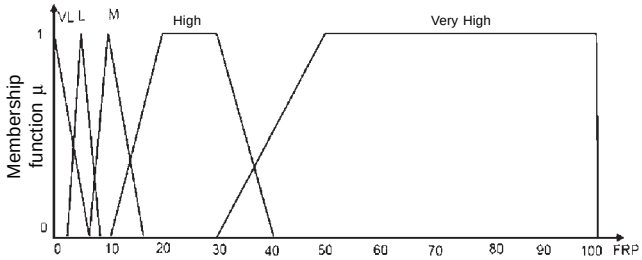

Figure 10: Fuzzy Profile of FRP

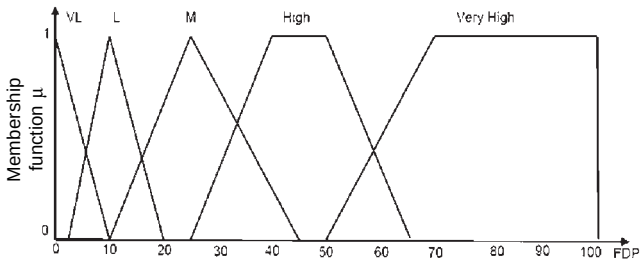

Figure 11: Fuzzy Profile FDP

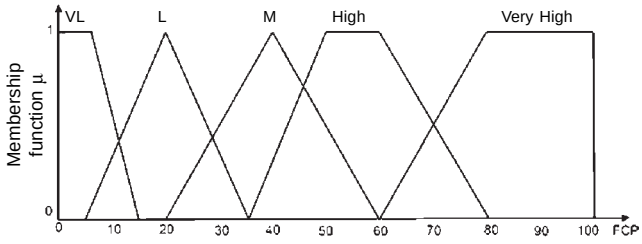

Figure 12: Fuzzy Profile of FRP

4.1.3. Develop Fuzzy Rule Base

The most important part of the early reliability prediction system is the rules, and how they interact with each other in order to generate results. The rules come from the experts so that the expert system can emulate the inference of an actual expert. To develop fuzzy rule base, we can acquire knowledge from different sources such as domain experts, historical data analysis of similar or earlier system, and engineering knowledge from existing literature's [25], [12]. In our experiments, we generated some rules from the software engineering point of view, and all of them take the form of 'If A then B'. Table 2, 3 & 4, shows the fuzzy if-then rules required for each phase of software life cycle.

Table 2
Fuzzy Rules at Requirements Phase

Rule	*RCR*	*RIW*	*PM*	*FRP*
1	L	L	L	VL
2	L	L	M	L
3	L	L	H	M
.	.	.		.
25	H	H	L	M
26	H	H	M	VH
27	H	H	H	VH

Table 3
Fuzzy Rules at Design Phase

Rule	*FRP*	*DDD*	*FDN*	*DC*	*FDP*
1	VL	L	L	L	VL
2	VL	L	L	M	L
3	V L	L	L	H	M
.	.	.		.	.
133	V H	H	H	H	H
134	VH	H	H	H	VH
135	VH	H	H	H	VH

Table 4
Fuzzy Rules at Coding Phase

Rule	*FRP*	*FDP*	*CC*	*CDD*	*FCP*
1	VL	VL	L	L	VL
2	VL	VL	L	M	VL
3	VL	VL	L	H	L
.	.	.	.	.	.
223	VH	VH	H	L	H
224	VH	VH	H	M	VH
225	VH	VH	H	H	VH

4.2. Information Processing Phase

In this phase, the fuzzy system maps all inputs on to an output. This process of mapping inputs on to output is known as fuzzy inference process or fuzzy reasoning [26], [24]. Basis for this mapping is the number of fuzzy IF-THEN rules, each of which describes the local behavior of the mapping. The Mamdani fuzzy inference system [27] is considered here for all the information processing.

4.2.1. Defuzzification

Defuzzification is the process of deriving a crisp value from a fuzzy set using any defuzzification methods such as Centroid, Bisector, Middle of maximum, Largest of maximum, and Smallest of maximum [23]. The most commonly used method is the Centroid method, which returns the center of area under the curve, is used in here for defuzzification.

4.3. Fault Prediction Phase

Stages present in the proposed structure are shown in Figure 1. The model resemble waterfall model [28]. It divides the structure into three consecutive phase I, II, and III i.e. requirement phase, design phase, and coding phase respectively. Phase-I predicts the number of faults at the end of requirement phase using requirement metrics such as RCR, RIW, and PM. Phase-II predicts the number of faults at the end of design phase using design metrics such as DDD, FDN, and DC. Since most of the software faults traced back to requirements error, FRP is considered as input metric to this phase. Similarly at phase-III besides the coding metrics CC and CDD, FRP and FDP are also considered as input to predict the number of faults at the end of design phase. The Mamdani fuzzy inference system [27] is considered here for fault prediction at each phase.

5. RESULTS

In the proposed model, in order to analyze the impact of individual phase of software life cycle on the prediction of software faults, the values of the metrics from three different software projects are considered.

The number of faults at end of each phase is shown in the Table 5, 6 & 7. Result of the best case and worst case input metrics applied to the proposed model state that the proposed model could be useful to predict the software faults that may range from 0 to 85. The number of fault at the end of requirement phase is 70 in worst case, 10 in average case

Table 5
Faults Prediction at Requirements Phase

	RCR	***RIW***	***PM***	***FRP***
Worst Case	100	0	0	70
Avg. Case	50	2.5	2.5	10
Best Case	0	5	5	1.33

Table 6
Faults Prediction at Design Phase

	DDD	***FDN***	***DC***	***FDP***
Worst Case	5	50	500	79.8
Avg. Case	2.5	25	250	45
Best Case	0	0	0	3

Table 7
Faults Prediction at Coding Phase

	CDD	*CC*	*FCP*
Worst Case	1	500	84.7
Avg. Case	0.5	250	56.7
Best Case	0	0	5.67

and 1.33 in the best case. The number of fault at the end of design phase is 79.8 in worst case, 45 in average case, and 3 in the best case. Finally the number of fault at the end of coding phase is 84.7 in worst case, 56.7 in average case, and 5.67 in the best case.

6. CONCLUSIONS

A model for the early prediction of software fault is presented in this paper. The model is based on reliability relevant software metrics and developing organization's CMM level. Total 8 reliability relevant metrics has identified and using fuzzy inference system, total number of faults at the end of each phase of software life cycle is predicted. For software professionals, this model provides an insight towards software metrics and its impact on software fault during the development process. For software project mangers, the model provides a methodology for allocating the resources for developing reliable and cost-effective software.

REFERENCES

[1] ANSI/IEEE, Standard Glossary of Software Engineering Terminology, STD-729-991, ANSI/IEEE, (1991).

[2] Agrawal, M., and Chari, K., Software Effort, Quality and Cycle Time: A Study of CMM Level 5 Projects, *IEEE Trans. on Software Eng.,* **33**, (3), (2007), 145-156.

[3] Musa, J. D., Iannino, A., and Okumoto, K., Software Reliability: Measurement, Prediction, Application, McGraw-Hill, (1987).

[4] Kaner, C., Software Engineering Metrics: What do they Measure and How do we Know?, 10th International Software Metrics Symposium, METRICS, (2004).

[5] Pham, H., System Software Reliability, Reliability Engineering Series, Springer, (2006).

[6] Gaffney, J. E., and Davis, C. F., An Approach to Estimating Software Errors and Availability, SPC-TR-88-007, Version 1.0, March 1988, Proc. 11th Minnow Brook Workshop on Software Reliability, (1988).

[7] Gaffney, J. E., and Pietrolewiez, J., An Automated Model for Software Early Error Prediction (SWEEP), Proc. 13th Minnow Brook Workshop on Software Reliability, (1990).

[8] Rome Laboratory (RL), Methodology for Software Reliability Prediction and Assessment, Technical Report RL-TR-92-52, **1 & 2**, (1992).

[9] Agresti, W. W., and Evanco, W. M., Projecting Software Defect form Analyzing Ada Design, *IEEE Trans. on Software Eng.,* **18**, (11), (1992), 988-997.

[10] Yu, T. J., Shen. V. Y., and Dunsmore, H. E., An Analysis of Several Software Defect Models, *IEEE Trans. on Software Eng.,* **14**, (9), (1988), 261-270.

[11] Khoshgoftaar, T. M., and Munson, J. C., Predicting Software Development Errors Using Complexity Metrics, *IEEE Journal on Selected Areas in Comm.,* **8**, (2), (1990), 253-261.

[12] Zhang, X., and Pham, H., An Analysis of Factors Affecting Software Reliability, *The Journal of Systems and Software,* **50**, (1), (2000), 43-56.

[13] Li, M., and Smidts, C., A Ranking of Software Engineering Measures Based on Expert Opinion, *IEEE Trans. on Software Eng.*, **29**, (9), (2003), 811-24.

[14] Kumar, K. S., and Misra, R. B., An Enhanced Model for Early Software Reliability Prediction using Software Engineering Metrics, Proc. 2nd Int'l Conf. on Secure System Integration and Reliability Improvement, (2008), 177-178.

[15] Paulk, M. C., Weber, C. V., Curtis, B., and Chrissis, M. B., Capability Maturity Model Version 1.1, *IEEE Software,* **10**, (3), (1993), 18-27.

[16] Krishnan, M. S., and Kellner, M. I., Measuring Process Consistency: Implications Reducing Software Defects, *IEEE Trans. on Software Eng.,* **25**, (6), (1999), 800-815.

[17] Diaz, M., and Sligo, J., How Software Process Improvement Helped Motorola, *IEEE Software,* **14**, (5), (1997) 75-81.

[18] Harter, D. E., Krishnan, M. S., and Slaughter, S. A., Effects of Process Maturity on Quality, Cycle Time and Effort in Software Product Development, Management Science, **46**, (2000), 451-466.

[19] Fenton, N., Software Metrics-A Rigorous Approach, Chapmann & Hall, London, (1991).

[20] IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software, IEEE Std 982.2, New York: IEEE, (1988).

[21] Montgomery, D. C., Design and Analysis of Experiments, Willy-India 5th Edition, (2005).

[22] Saravana K. K., Misra R. B. and N. K. Goyal, "Development of Fuzzy Software Operational Profile", *International Journal of Reliability, Quality and Safety Engineering,* **15**, (6), (2008), 581-597.

[23] Ross, J. T., Fuzzy Logic with Engineering Applications, Willy-India 2nd Edition, (2005).

[24] Zadeh, L. A., Knowledge Representation in Fuzzy Logic, *IEEE Transactions on Knowledge and Data Engineering,* **1**, (1989), 89-100.

[25] Xie, M., Hong, G. Y., and Wohlin, C., Software Reliability Prediction Incorporating Information from a Similar Project, *The Journal of Systems and Software,* **49**, (1999), 43-48.

[26] Bowles, J. B., and Pelaez, C. E., Application of Fuzzy Logic to Reliability Engineering, *Proc. IEEE,* **83**, (3), (1995), 435-449.

[27] Mamdani, E. H., Applications of Fuzzy Logic to Approximate Reasoning Using Linguistic Synthesis, *IEEE Trans. on Computers,* **26**, (12), (1977), 1182-1191.

[28] Pressman, R. S., Software Engineering: A Practitioner's Approach, McGraw-Hill, 6th Edition, (2005).

Appendix A. Fuzzy Profiles of Reliability Relevant Metrics at Requirements, Design and Coding Phase

#	*Metrics*	*Rank*	*Fuzzy Range*	*Low*	*Moderate*	*High*
1.	Code defect density	0.83	[0-1]	(0;0;0.2;0.4)	(0.2;0.4;0.5;0.7)	(0.5;0.7;1;1)
2.	Design defect density	0.75	[0-1]	(0;0;0.2;0.4)	(0.2;0.4;0.5;0.7)	(0.5;0.7;1;1)
3.	Cyclomatic complexity	0.74	[0-500]	(0;0;100;150)	(100;150;250;300)	(250;350;500;500)
4.	Fault density	0.73	[0-5]	(0;0;1;2)	(1;2;2.5;3.5)	(2.5;3.5;5;5)
5.	Fault-days number	0.71	[0-50]	(0;0;2;5)	(1;4;8;15)	(8;20;50;50)
6.	Requirement change requests	0.69	[0-100]	(0;0;25;35)	(25;35;55;75)	(60;80;100;100)
7.	Error Distribution	0.65	[0-10]	(0;0;3;4.5)	(3;4.5;5.5;6.5)	(5.5;8;10;10)
8.	Minimal unit test case determination	0.64	[0-10]	(0;0;2;4)	(3;5;6)	(5;8;10;10)
9.	Reviews, inspection and walkthroughs	0.61	[0-5]	(0;0;1;2)	(1;2;2.5;3.5)	(3;4;5;5)
10.	Man hours per major defect detected	0.61	[0-5]	(0;0;1;2)	(1;1;2;3)	(2;3;5;5)
11.	Software capability maturity level	0.60	[0-5]	(0;0;0.5;1)	(1;1.5;2.5;3)	(3;4;5;5)
12.	Dataflow complexity	0.59	[0-500]	(0;0;50;100)	(80;120;200;250)	(200;300;500;500)
13.	Requirement traceability	0.56	[0-1]	(0;0;0.2;0.4)	(0.2;0.4;0.5;0.7)	(0.5;0.7;1;1)
14.	Function point analysis	0.55	[0-100]	(0;0;5;10)	(5;10;20;30)	(25;50;100;100)
15.	System design complexity	0.53	[0-10]	(0;0;2;4)	(3;4;5;6)	(5;8;10;10)
16.	Requirement compliance	0.50	[0-100]	(0;0;1;5)	(3;5;10;20)	(20;40;100;100)
17.	Feature point analysis	0.50	[0-100]	(0;0;10)	(10;20;30)	(25;50;100;100)
18.	No. of faults remaining (error seeding)	0.47	[0-100]	(0;0;5)	(5;10;15;25)	(20;40;100;100)
19.	Graph-theoretic static architecture complexity	0.46	[0-10]	(0;0;2;4)	(3;4;5;6)	(5;8;10;10)
20.	Bugs per line of code	0.46	[0-5]	(0;0;1;2)	(1;1;2;3)	(2;3;5;5)
21.	Cause & effect graphing	0.40	[0-1]	(0;0;0.2;0.4)	(0.2;0.4;0.5;0.7)	(0.5;0.7;1;1)
22.	Cohesions	0.36	[0-1]	(0;0;0.2;0.4)	(0.2;0.4;0.5;0.7)	(0.5;0.7;1;1)
23.	Completeness	0.36	[0-1]	(0;0;0.2;0.4)	(0.2;0.4;0.5;0.7)	(0.5;0.7;1;1)