International Journal of Computer and Communication Technology

Volume 3 | Issue 1 Article 1

January 2012

Predicting Fault-prone Software Module Using Data Mining Technique and Fuzzy Logic

Ajeet Kumar Pandey
Reliability Engineering Centre Indian Institute of Technology Kharagpur Kharagpur, W.B. -721302, ajeet.mnnit@gmail.com

Neeraj Kumar Goyal
Indian Institute of Technology - Kharagpur, ngoyal@hijli.iitkgp.ernet.in

Predicting Fault-prone Software Module Using Data Mining Technique and Fuzzy Logic

Ajeet Kumar Pandey*, Neeraj Kumar Goyal
Reliability Engineering Centre
Indian Institute of Technology Kharagpur
Kharagpur, W.B. -721302
Email: ajeet.mnnit@gmail.com, ngoyal@hijli.iitkgp.ernet.in

Abstract--**This paper discusses a new model towards reliability and quality improvement of software systems by predicting fault-prone module before testing. Model utilizes the classification capability of data mining techniques and knowledge stored in software metrics to classify the software module as fault-prone or not fault-prone. A decision tree is constructed using ID3 algorithm for existing project data in order to gain information for the purpose of decision making whether a particular module id fault-prone or not. The gained information is converted into fuzzy rules and integrated with fuzzy inference system to predict fault-prone or not fault-prone software module for target data. The model is also able to predict fault-proneness degree of faulty module. The goal is to help software manager to concentrate their testing efforts to fault-prone modules in order to improve the reliability and quality of the software system. We used NASA projects data set from the PROMOSE repository to validate the predictive accuracy of the model.**

1 INTRODUCTION

Due to the increased dependency of modern system on software-based system, software reliability and quality has become the primary concern during the software development. It is difficult to produce fault-free software due to the problem complexity, complexity of human behaviors, and the resource constrains. System failures due to the software failure are common and results in undesirable consequences, which can adversely affect both reliability and safety of the system. A software system consists of various modules and, in general, it is known that a small number of software modules are responsible for majority of the failures. Moreover, it is also known that early identification of faulty module can help in producing software of quality and reliability more cost effectively. Therefore, it is desirable to classify the software module as fault-prone (FP) or not fault-prone (NFP) just after the coding phase of software development. Once the modules are classified as FP or NFP, more testing efforts can be put on the fault-prone module to produce reliable software.

A fault is a defect in source code that causes failures when executed [1]. Software modules are said to be fault-prone, when there are a high probability of finding faults during its operation. In other words, a fault-prone software module is the one containing more number of expected faults than a given threshold value. The threshold value can take any positive value and depends on the project specific criteria. In general, when a software module has been identified as fault-prone, more and more testing efforts are applied to improve its quality and reliability. The amount of testing effort applied to a faulty module should be proportional to its degree of fault-proneness. In other words, it is undesirable to apply equal amount of testing efforts to all the software modules. The testing efforts should be allocated on the basis of its degree of fault-proneness.

This paper proposes a model for prediction of fault-prone software module using fuzzy logic [2] and data mining techniques [3]. Furthermore, fault-prone modules are converted into fuzzy sets [4] to predict their degree of fault-proneness.

Rest of the paper is organized as follows: the following section presents literature survey related with the problem. Section 3 gives the brief idea about of data mining techniques and fuzzy set theory. Section 4 describes the proposed model. Section 5 contains results and discussion whereas conclusions are presented in Section 6.

2 LITERATURE SURVEYS

Predicting the fault-prone software modules is of a great interest among the software quality researchers and industry professionals. As a result of this, various efforts have been made for software fault prediction using methods such as Decision Trees [5], Neural Networks [6],

Support Vector Machines [7], Bayesian Methods [8], Naïve Bayes [9], Fuzzy Logic [10, 11], Dempster–Shafer Belief Networks [12], Genetic Programming [13], Case-based Reasoning [14, 15], and Logistic Regression [16, 17].

On reviewing literature, it is found that various machine learning approaches [18] such as supervised, semi-supervised, and unsupervised have been used for building a fault prediction models. Among these, supervised learning approach is widely used and found to be more useful for predicting fault-prone modules if sufficient amount of fault data from previous releases are available. Generally, these models use software metrics of earlier software releases and fault data collected during testing phase. The supervised learning approaches can not build powerful models when data is limited. Therefore, the some researchers presented a semi-supervised classification approach [19, 20] for software fault prediction with limited fault data.

Menzies et al. [9] shown that defect predictors can be learned from static code attributes since they are useful, easy to use, and widely used. Taking clues from [9] Pandey and Goyal [21] have presented an early fault prediction model using process maturity and software metrics. Software metrics have been shown to be good indicators of software quality, and the number of faults present in the software. Software metrics, which represent the software product and its process, can be collected relatively earlier during software development. Catal and Diri [22] investigated the effects of data size, metrics, and the feature selection techniques for software fault prediction. They have also presented a systematic review of various software fault prediction studies [23].

From the literature, it has been found that the decision tree induction algorithms such as CART, ID3 and C4.5 are efficient technique for module classification. These algorithms uses crisp value of software metrics and classify the module as a fault-prone or not fault-prone. It has been found that most of the early phase software metrics have fuzziness in nature and crisp value assignment seems to be impractical. Also a software module can't be completely fault-prone or not fault-prone. In other words it is unfair to assign a crisp value of software module representing its fault proneness.

3 RESEARCH BACKGROUND

3.1 Data Mining

Today's advanced information systems have enabled collection of increasingly large amounts of data. To analyze these data, the interdisciplinary field of Knowledge Discovery in Databases (KDD) has emerged. KDD comprises of many steps namely, data selection, data preprocessing, data transformation, data mining and data interpretation and evaluation. Data mining forms a core activity in KDD [3].

Data mining entails the overall process of extracting knowledge from large amounts of data. Different types of data mining are discussed in the literature such as regression, classification and associations. The focus here is on classification technique, which is the task of classifying the data into a predefined class to its predictive characteristics. The result of a classification technique is a model which makes it possible to classify future data points based on a set of specific characteristics in an automated way. In the literature, there are many classification techniques, some of the most commonly used being ID3, C4.5, logistic regression, linear and quadratic discriminant analysis, k-nearest neighbor, Artificial Neural Networks (ANN) and Support Vector Machines (SVM) [3]. Among these ID3 (Iterative Dichotomiser), a well known decision tree induction algorithm, is found to be an efficient, widely used and practical classification technique. This paper integrate ID3 algorithm with fuzzy inference system to predict software module as FP or NFP.

3.2 Fuzzy set theory

3.2.1 *Crisp and fuzzy sets.* Crisp or Classical set can be defined as a collection of well defined distinct object. In other words, crisp sets contain objects that satisfy precise properties of membership. For crisp set, an element x in the universe X is either a member of some crisp set (A) or not. This binary issue of membership can be can be represented by a characteristic function as:

$$\chi_A(x) = \begin{cases} 1, if x \in A \\ 0, if x \notin A \end{cases}$$

where, $\chi_A(x)$ gives an unambiguous membership of the element, x in a set A.

A fuzzy set is a set containing elements that have varying degree of membership in the set. Unlike crisp set, elements in a fuzzy set need not be complete and can also be member of other fuzzy sets on the same universe. In this way, fuzzy set allows partial membership as well as binary membership. Let $\tilde{A}$ is a fuzzy set of A, if an

element in the universe, say, x is a member of fuzzy set $\tilde{A}$ then this mapping is given by a membership function $\mu_{\tilde{A}}(x)$. The membership function $\mu_{\tilde{A}}(x)$, gives the degree of membership for each element in the fuzzy set Ã and is defined in range [0, 1] where 1 represents elements that are completely in Ã, 0 represents elements that are completely not in Ã, and values between 0 and 1 represent partial membership in Ã. Formally, a fuzzy set Ã can be represented using Zadeh's notation [4] as:

$\tilde{A} = \left\{ \frac{\mu_1}{x_1} + \frac{\mu_2}{x_2} + \frac{\mu_3}{x_3} + .. + \frac{\mu_n}{x_n} \right\}$, where $\mu_1, \mu_2 \ldots \mu_n$ are the membership value of the elements $x_1, x_2 \ldots x_n$ respectively in the fuzzy set $\tilde{A}$.

Let $F\tilde{P}$ is a fuzzy set representing collection of fault-prone modules m1, m2 . . . mn. Let µ1, µ2 …µn are the membership value of module m1, m2 . . . mn. respectively. These membership values represent the degree of fault-proneness of a faulty module. Taking these points into consideration, a fuzzy set of fault-prone module can be described using Zadeh's notation [4] as:

$F\tilde{P} = \left\{ \frac{\mu_1}{m_1} + \frac{\mu_2}{m_2} + \frac{\mu_3}{m_3} + .. + \frac{\mu_n}{m_n} \right\}$.

3.2.2 *Fuzzy profiles development.*

Software metrics are measurement of the software development process and product. These software metrics have been shown to be good indicator of software quality, and the number of faults present in the software and can be collected relatively earlier during software development [21]. Knowledge is stored in the software metrics can be used for quality prediction of the developing software. IT has also been found that early phase software metrics have fuzziness in nature and crisp value assignment seems to be impractical [21]. Also a software module can't be completely fault-prone or not fault-prone. In other words it is unfair to assign a crisp value of software module representing its fault proneness.

Membership functions of a software metrics can be developed by selecting a suitable method from the various available methods such as triangular, trapezoidal, gamma and rectangular [2]. However, triangular membership functions (TMFs) are widely used for calculating and interpreting reliability data because they are simple and easy to understand [10]. Here, we have considered various software metrics as input variable for the model. Furthermore, we assume that these software metrics are of logarithmic nature and can be divided into five linguistic categories namely very low (VL), low (L) medium (M), high (H) and very high (VH), depending on their actual value. Considering these, fuzzy profile ranges (FPR) of software metrics are developed using the following formula and shown in fig. 1.

$$FPR = \left[1 - \frac{\{\log_{10}(1:5)\}}{\{\log_{10}(5)\}} \right]$$

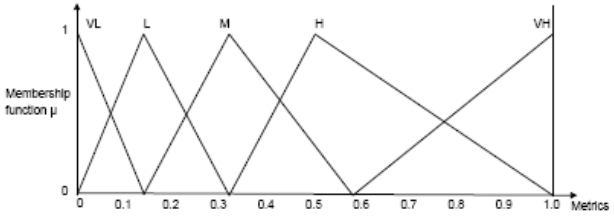

Figure 1. Fuzzy profiles of input variables

Fuzzy profile of each input variables may take the values: VL (0; 0; 0.14), L (0; 0.14; 0.32), M (0.14; 0.32; 0.57), H (0.32; 0.57; 1.0) and VH (0.57; 1.0; 1.0).

Output of the model is DFP (Fault-prone Degree), which is also assumed to fallow logarithmic scale and can be categorizes into seven linguistic categories such as: very very low (VVL), very low (VL) medium (M), high (H), very high (VH) and very very high (VVH). Therefore fuzzy profile range (FPR) of DFP can be developed using the following formula and shown in fig. 2.

$$FPR = \left[1 - \frac{\{\log_{10}(1:7)\}}{\{\log_{10}(7)\}} \right]$$

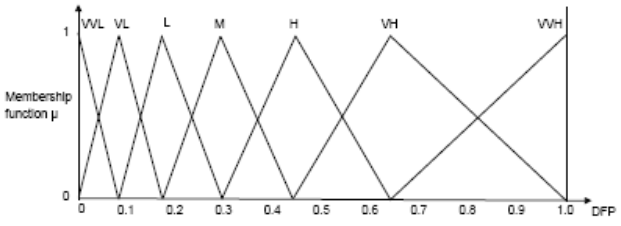

Figure 2. Fuzzy profiles of output variables

Fuzzy profile of output variable may take the values: VVL (0; 0; 0.08), VL (0; 0.08; 0.17), M (0.17; 0.29;

0.44), H (0.29; 0.44; 0.64), VH (0.44; 0.64; 1.0) and VVH (0.64; 1.0; 1.0).

4 PROPOSED MODEL

4.1 Model architecture

The model architecture is shown in Fig. 3. The model is implemented in MATLAB utilizing fuzzy logic toolbox. The basic steps of the model are identification of input-output variables, development of fuzzy profile of these input/output variables, defining relationships between inputs and output variables using fuzzy inference system (FIS). Software metrics, as listed in Table 1, are considered as input variables for the proposed model and output variable of the model is degree of fault-prone (DFP) of a module, that decides whether module is a fault-prone module (FP) or a not fault-prone (NFP).

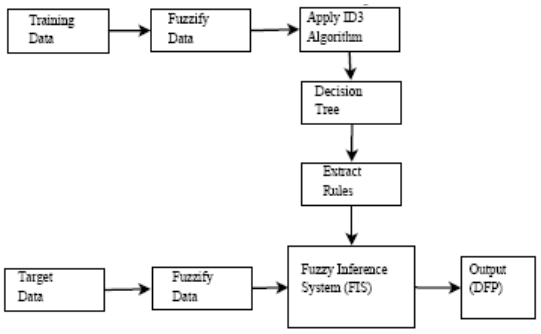

Figure 3. Proposed model architecture

4.2 Data collection

This study makes the use of KC2 project data [24] which is a public domain data set and was made available through Metric Data Program (MDP) at NASA. The KC2 project is the science data processing unit of a storage management system used for receiving and processing ground data for missions, and written in C++ language. This project data set contains 522 program modules, of which 107 modules have one or more faults while remaining 415 modules are fault-free i.e. have no software faults.

Each program module in the KC2 was characterized by 21 software metrics (5 different lines of code metrics, 3 McCabe metrics, 4 base Halstead metrics, 8 derived Halstead metrics, 1 branch-count) and 1 target metric, which says whether a software module is fault-prone or not. Out of these 21 software metrics, only 13 metrics (5 different lines of code metrics, 3 McCabe metrics, 4 base Halstead metrics, and 1 branch- count) are utilized because 8 derived Halstead metrics do not contain any extra information for software fault prediction. Theses metrics are given in Table 1.

Table 1. Software metrics inside KC2 project [24]

Metrics	Information
LOC	McCabe's line count of code
EL	Executable LOC
CL	Comment LOC
BL	Blank LOC
CCL	Code and comment LOC
n1	No. of unique operators
n2	No. of unique operands
N1	Total no. of operators
N2	Total no. of operands
CC	McCabe's cyclomatic complexity
EC	McCabe's essential complexity
DC	McCabe's design complexity
BC	Branch count of flow graph

4.3 Decision tree construction

Decision tree is one of the most efficient classification techniques. Many algorithms have been proposed for building decision trees. The most popular are ID3 (Iterative Dichotomiser 3) introduced by Quinlan [25], and its modifications, e.g. C4.5 which makes a decision tree for classification from symbolic data.

A decision tree is a flow-chart-like structure where each internal node denotes a test on an attribute, each branch represents outcome of the test, and leaf node represent the target class. Internal nodes are denoted by rectangles, and leaf nodes are denoted by ovals [3]. Figure 4 shows a simple decision tree for a software module, where testing attributes are various software metrics, branch represent the outcome of the test *e.g.* low (L), medium (M) or high (H) and target classes are FP and NFP.

Decision trees are comprised of two major procedures (1) building procedure or induction (2) classification procedure or inference. Building procedure starts with an empty tree to select an appropriate test attribute for each decision node using some attribute selection measure. The process continues for each sub decision tree until reaching leaves with their corresponding class. The knowledge represented in decision tree can be extracted and represented in the form of classification "IF-THEN"

rules. Classification procedure uses these "IF-THEN" rules to classify new instances, having only values of all its attributes. We use the associated label to obtain the predicted class value of the target data.

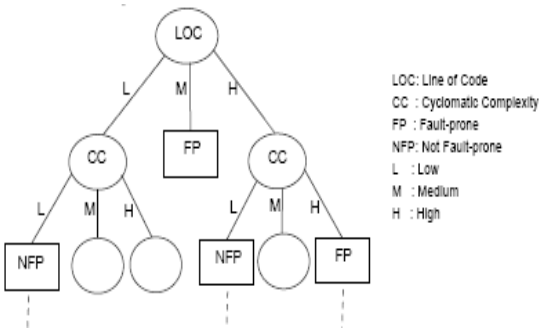

Figure 4. A simple decision tree

Attribute selection is generally based on information gain, serves as a criterion in choosing test attribute at each decision node. Partitioning consists of partitioning the training set according to all possible attributes values which leads to the generation of one partition for each possible value of the selected attribute. Stopping, stops the partitioning process if (i) all the remaining objects belongs to only one class, then the node is declared as a leaf labeled with this class value (ii) there is no further attribute to test. For software module classification, ID3 algorithm steps can be summarized as follows:

1. Take all the metrics of project data and count their information gain.
2. Choose a metric as a test-metric with highest information gain value.
3. Create nodes for this test-metric and do partition.
4. Repeat (1)-(3) recursively until
   - (i) If all data for a selected node belongs to the same class.
   - (ii) If there are no more remaining metrics on which the data may be further partitioned.
   - (iii) If there are no data for the selected node.

These steps can be applied to a set of data to generate decision trees. Classification rules are extracted form the resulting decision tree by tracing a path from the root to a leaf node. The accuracy of the derived classifier can be estimated using Confusion Matrix and discussed in the following section.

4.4 Estimating classifier accuracy

Existing project data [24] are used data to derive the classifier after then different target data sets are applied to the classifier to estimate the accuracy of the model. Accuracy of the model is considered as the comparison factor with the earlier traditional models and may be obtained using Confusion Matrix [26] as given in Table 2. A confusion matrix contains information about actual and predicted classifications done by a classification system.

Table 2. A confusion matrix

		Predicted	
	Label	FP	NFP
Actual	FP	True Positive (TP)	False Negative(FN)
	NFP	False Positive (FP)	True Negative (TN)

- TP and TN are the number of correct predictions that an instance is positive and negative respectively.
- FP and FN are the number of incorrect predictions that an instance is positive and negative respectively.

Authors such as, El-Emam et al. [27], Elish et al. [7] have used confusion matrix as the basis for determining predictive accuracy of the classifier. The predictive accuracy, also known as correct classification rate, is defined as the ratio of the number of modules correctly predicted to the total number of modules. It is calculated as follows:

$$Accuracy = \left[\frac{(TP + TN)}{(TP + TN + FP + FN)} \right] \times 100$$

For example, consider a model which predicts for 10,000 software modules whether each module is fault-prone or not fault-prone. This model correctly predicts 9,700 not fault-prone, and 100 fault-prone. The model also incorrectly predicts 150 modules which are not fault-prone to be fault-prone and 50 modules which are fault-prone to be not fault-prone. The confusion matrix results the model accuracy to 98% and shown in Table 3.

Table 3. Confusion matrix of software modules

Predicted		
Label	FP	NFP

Actual	FP	100	50
	NFP	150	9700

4.5 Proposed algorithm

Step 1 Identify the target class C {FP, NFP}, for a given project data.

Step2 Compute the information gain value for each metric in the project data.

Step 3 Choose a metric as a test-metric with highest information gain value.

Step 4 Create a node for this test-metric and do partition.

Step 5 Repeat Step 1-Step 3 recursively until

Step 5.1 if all data for a selected node belongs to the same class

Step 5.2 if there are no more remaining metrics on which the data may be further partitioned.

Step 5.3 if there are no data for the selected node.

Step 6 Returns (Decision Tree).

Step 7 Extract classification rules form the decision tree.

Step 8 Use these rules as fuzzy rules in to FIS at step 9.

Step 9 Develop a FIS (inputs are software metrics and outputs are FP or NFP modules).

Step 10 Classify the project data into target class (FP or NFP) using developed FIS.

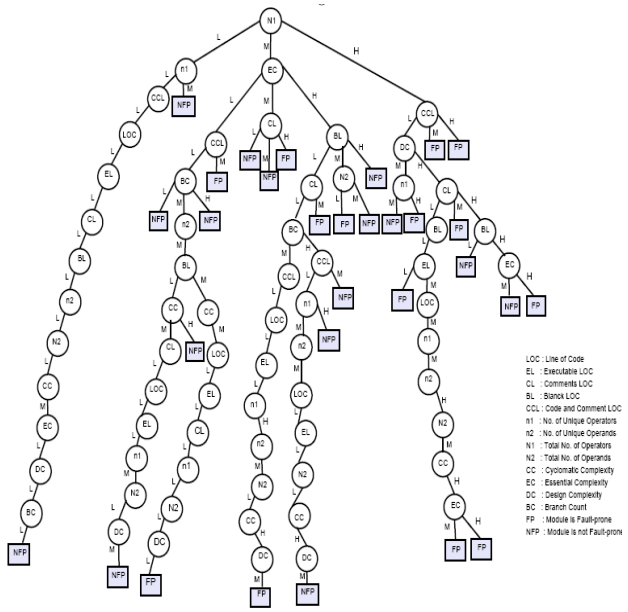

Figure 5. A decision tree using 60% of KC2 data set

5 RESULTS AND DISCUSSION

We examined the decision tree learning algorithm ID3 and implement this algorithm using MATLAB programming. Various decision trees are generated using 20%, 40%, 60% and 80% of KC2 project data set [24]. Figure 5 shows a decision tree using 60 percent of KC2 data set. After generating decision trees, we can derive different classifiers and accuracy of each classifier is estimated on different mutually exclusive target data as shown in Table 4. The experiment is repeated ten times and each experiment type has been chosen as "Train/Test Percentage" of the data. The accuracy of each repetition is shown on Table 4. It is obvious form Table 4 that the prediction accuracy increases with increasing training data.

Model accuracy is estimated as the overall average accuracy obtained from ten different experiment results. The accuracy of the proposed model using KC2 data set is found to be 87.37 percent, which is better than the earlier models [28, 29] and are given in Table 5.

Next, we show the dependence of prediction accuracy on the training data. For this we have developed six different FIS namely, MP5_95, MP10_90, MP20_80, MP40_60, MP60_40, and MP20_80, using KC2 data set. Initially, when training data size is very small (5 percent), prediction accuracy is found to be 70.22 percent and it suddenly grows to 83.47 as soon as size of training data is increased to 10 percent. It is found that on further increasing the size of training data, the prediction accuracy grows slightly and reaches up to 95.08 percent as shown in Table 6.

6 CONCLUSIONS

This paper presented a model for prediction of fault-prone module for a large software system using ID3 algorithm and fuzzy inference system. Model is also able to predict the fault-prone degree of a faulty module when it is converted into a fuzzy set. By doing so, the project manager can prioritize their testing effort to achieve software system having more quality and reliability. Model results are promising when compared with some of the earlier models for KC2 dataset. Some useful directions for future work may include analyzing the amount of testing efforts saved by the model.

Table 4. Prediction accuracy of the model on KC2 dataset

Experiments	Training (%)	Test (%)	Number of FP modules (predicted)	Accuracy (%)	Average (%)
1	20	20	17	81.73	81.21
2		40	42	78.85	
3		60	54	80.13	
4		80	32	84.12	
5	40	20	6	84.62	86.04
6		40	19	87.50	
7		60	26	86.02	
8	60	20	25	85.58	87.16
9		40	17	88.74	
10	80	20	10	95.08	95.08

Table 5. Performance results of the model on KC2 dataset

Model	Class Prediction	Rank Prediction	Accuracy (%)
Catal et al.[28]	Yes	No	82.22
Saravana et al. [29]	Yes	No	81.72
Proposed Model	Yes	Yes	87.37

Table 6. Training effect on prediction accuracy

	MP5_95	MP10_90	MP20_80	MP40_60	MP60_40	MP80_20
Training (%)	5	10	20	40	60	80
Testing (%)	95	90	80	60	40	20
Accuracy (%)	70.22	83.47	84.12	85.09	87.84	95.08

REFERENCES

1. J. D. Musa, A. Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, and Application* (McGraw-Hill Publication, 1987).
2. T. J. Ross, *Fuzzy Logic with Engineering Applications* (Willy-India 3rd Edition, 2010).
3. J. Han, M. Kamber, *Data Mining: Concepts and Techniques* (Morgan Kaufmann Publishers, USA, 2001).
4. L. A. Zadeh, Fuzzy Sets, *Information and Control*, **8** (1965) 338-353.

5. T.M. Khoshgoftaar and N. Seliya, Software quality classification modeling using the SPRINT decision tree algorithm, *In the proceedings of the 4th IEEE International Conference on Tools with Artificial Intelligence*, Washington, DC, 2002, pp. 365-374.

6. M.M. Thwin and T. Quah, Application of neural networks for software quality prediction using object-oriented metrics, *In the proceedings of the 19th International Conference on Software Maintenance*, Amsterdam, The Netherlands, 2003, pp. 113-122.

7. K.O. Elish and M.O. Elish, Predicting defect-prone software modules using support vector machines, *Journal of Systems and Software*, **81** (2008) 649-660.

8. G.J. Pai and J.B. Dugan, Empirical analysis of software fault content and fault proneness using bayesian methods, *IEEE Transactions on Software Engineering*, **33** (2007) 675-686.

9. T. Menzies, J. Greenwald and A. Frank, Data Mining Static Code Attributes to Learn Defect Predictors, *IEEE Transactions on Software Engineering*, **33** (2007) 2-13.

10. O. P. Yadav, N. Singh, R. B. Chinnam, and P. S. Goel, A fuzzy logic based approach to reliability improvement estimation during product development, *Reliability Engineering and System Safety*, **80** (2003) 63-74.

11. N. J. Pizzi, *Software quality prediction using fuzzy integration: a case study*, (Springer-Verlag, 2007), pp. 67-76.

12. L. Guo, B. Cukic and H. Singh, Predicting fault prone modules by the Dempster-Shafer belief networks, *In the proceedings of the 18th IEEE International Conference on Automated Software Engineering*, IEEE Computer Society, Montreal, Canada, 2003, pp. 249-252.

13. M. Evett, T. Khoshgoftaar, P. Chien and E. Allen, GP-based software quality prediction, *In the proceedings of the 3rd Annual Genetic Programming Conference*, San Francisco, CA, 1998, pp. 60-65.

14. T. M. Khoshgoftaar, N. Seliya and N. Sundaresh, An Empirical Study of Predicting Software Faults with Case-Based Reasoning, *Software Quality Journal*, **14** (2006) 85-111.

15. K. El Emam, S. Benlarbi, N. Goel and S. Rai, Comparing case-based reasoning classifiers for predicting high risk software components, *Journal of Systems and Software*, **55** (2001) 301-320.

16. H.M. Olague, S. Gholston and S. Quattlebaum, Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes, *IEEE Transactions on Software Engineering*, **33** (2007) 402-419.

17. N. F. Schneidewind, Investigation of Logistic Regression as a Discriminant of Software Quality, *In the proceedings of 7th International Software Metrics Symposium*, London, UK, 2001, pp. 328-337.

18. I. Gondra, Applying machine learning to software fault-proneness prediction, *Journal of Systems and Software*, **81** (2008) 186-195.

19. N. Seliya N and T. M. Khoshgoftaar, Software Quality Estimation with Limited Fault Data: A Semi-Supervised Learning Perspective, *Software Quality Journal*, **15** (2007) 327-344.

20. N. Seliya and T. M. Khoshgoftaar, Software Quality Analysis of Unlabeled Program Modules with Semi-Supervised Clustering, *IEEE Transactions on Systems, Man and Cybernetics-Part A: Systems and Humans*, **37** (2007) 201-211.

21. A. K. Pandey and N. K. Goyal, A Fuzzy Model for Early Software Fault Prediction Using Process Maturity and Software Metrics, *International Journal of Electronics Engineering*, **1** (2009) 239-245.

22. C. Catal and B. Diri, Investigating The Effect Of Dataset Size, Metrics Set, and Feature Selection Techniques on Software Fault Prediction Problem, *Information Sciences*, **179** (2009) 1040-1058.

23. C. Catal and B. Diri, A Systematic Review of Software Fault Predictions studies, *Expert Systems with Applications*, **36** (2009) 7346-7354.

24. S. J. Sayyad and T. J. Menzies, The PROMISE Repository of Software Engineering Databases (2005), http://promise.site.uottawa.ca/SERepository.

25. J. R. Quinlan, Induction on decision trees, *Machine Learning*, **1** (1986) 81-106.

26. I. Witten, and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques* (Morgan Kaufmann, San Francisco, 2005).

27. K. El-Emam, W. Melo and J. C. Machado, The prediction of faulty classes using object-oriented design metrics, *Journal of Systems and Software*, **56** (2001) 63-75.

28. Catal and B. Diri, A Fault Prediction Model With Limited Fault Data to Improve Test Process, *In the proceedings of the 9th International Conference on Product Focused Software Process Improvement*, LNCS 5089, 2008, pp. 244-257.

29. Saravana Kumar K., *Early Software Reliability and Quality Prediction* (Ph.D. Thesis, IIT Kharagpur, Kharagpur, India, 2009).