A Fuzzy Logic Expert System to Predict Module Fault Proneness using Unlabeled Data

Golnoush Abaei, Ali Selamat, Jehad Al Dallal

PII: S1319-1578(18)30024-7
DOI: https://doi.org/10.1016/j.jksuci.2018.08.003
Reference: JKSUCI 485

To appear in: *Journal of King Saud University - Computer and Information Sciences*

Received Date: 10 January 2018
Revised Date: 29 July 2018
Accepted Date: 2 August 2018

Please cite this article as: Abaei, G., Selamat, A., Al Dallal, J., A Fuzzy Logic Expert System to Predict Module Fault Proneness using Unlabeled Data, *Journal of King Saud University - Computer and Information Sciences* (2018), doi: https://doi.org/10.1016/j.jksuci.2018.08.003

A Fuzzy Logic Expert System to Predict Module Fault Proneness using Unlabeled Data

Golnoush Abaei[a], Ali Selamat[a, b,c,d, *], Jehad Al Dallal[e]

[a]*Software Engineering Research Group (SERG), Department of Software Engineering, Faculty of Computing, Universiti Teknologi Malaysia, 81310, UTM Johor Bahru, Johor, Malaysia*

[a] *School of Computing, Faculty of Engineerin, UTM & UTM and Media and Games Center of Excellence (MagicX), Universiti Teknologi Malaysia, Johor Bahru, Malaysia*

[b] *Malaysia Japan International Institute of Technology (MJIIT), Universiti Teknologi Malaysia (UTM), Johor Bahru, Malaysia*

[c] *Faculty of Computer Science, National University of Modern Languages, Islamabad, Pakistan*

[d]*Center for Basic and Applied Research, Faculty of Informatics and Management, University of Hradec Kralove*

Rokitanskeho 62, 500 03 Hradec Kralove, Czech Republic

[e]*Department of Information Science, Kuwait University, P.O. Box 5969, Safat 13060, Kuwait*

ABSTRACT

Several techniques have been proposed to predict the fault proneness of software modules in the absence of fault data. However, the application of these techniques requires an expert assistant and is based on fixed thresholds and rules, which potentially prevents obtaining optimal prediction results. In this study, the development of a fuzzy logic expert system for predicting the fault proneness of software modules is demonstrated in the absence of fault data. The problem of strong dependability with the prediction model for expert assistance as well as deciding on the module fault proneness based on fixed thresholds and fixed rules have been solved in this study. In fact, involvement of experts is more relaxed or provides more support now. Two methods have been proposed and implemented using the fuzzy logic system. In the first method, the Takagi and Sugeno-based fuzzy logic system is developed manually. In the second method, the rule-base and data-base of the fuzzy logic system are adjusted using a genetic algorithm. The second method can determine the optimal values of the thresholds while recommending the most appropriate rules to guide the testing of activities by prioritizing the module's defects to improve the quality of software testing with a limited budget and limited time. Two datasets from NASA and the Turkish white-goods manufacturer that develops embedded controller software are used for evaluation. The results based on the second method show improvement in the false negative rate, f-measure, and overall error rate. To obtain optimal prediction results, developers and practitioners are recommended to apply the proposed fuzzy logic expert system for predicting the fault proneness of software modules in the absence of fault data.

Keywords: Fuzzy logic system, Genetic algorithm, Data-base, Rule-base, Threshold.

***Corresponding author**

Ali Selamat*, [a]Software Engineering Research Group (SERG), *Department of Software Engineering, Faculty of Computing*, Universiti Teknologi Malaysia, 81310 UTM Johor Bahru, Johor, Malaysia (aselamat@utm.my)

1. Introduction

Prediction of software reliability is become vital and crucial in software process due to the fast growing of software programs in terms of size and complexity [1]. The main causes for faults in a software system include wrong specifications and inappropriate configuration development [2] as well as incorrect design or implementation. Software fault prediction methods improve the testing process by focusing and allocating more resources to fault-prone modules [3].

Based on a deep analysis investigation, Radjenovicet al. [4] and Catal and Diri [5]reported that different sets of software measurement metrics are considered to be appropriate data for building fault-prediction models. Software quality metrics and faulty data (referred to here as data labels) of systems that are similar to the current system or previous versions (historical data) of the considered software system can be used to build fault prediction models; this approach is called the supervised learning approach and it is popular in the area of software fault prediction [6-10]. In contrast, unsupervised approaches such as clustering can be employed for quality assessment when data are unlabeled [11-13]. Clustering algorithms consider a set of similar entities as a group based on specific features and then a qualified software engineering expert [14] can label clusters as faulty or non-faulty based on the characteristics of the data inside each cluster. However, finding a qualified expert is a potentially difficult task [14]. To overcome this problem, some researchers [13, 15-17] use rule conditions and/or thresholds for quality metrics such as line of code, cyclomatic complexity, total number of operands, and total number of operators, provided by Integrated Software Metrics, Inc. (ISM) [18] for labeling the clustered groups as including faulty or non-faulty modules. The key limitation for such an approach is the existence of variations in metrics threshold values, thresholds intervals, and rules conditions, which greatly affect the module labeling results. Typically, vagueness and fuzziness in metrics threshold values and built rules are noticed, which potentially prevents the gathering of certain results for the optimal prediction solution. To overcome the fuzziness and uncertainty problem, expert systems based on fuzzy logic can be considered. To find an optimal solution, it is necessary to account for the data-base and rule-base sections of the expert system. Although these two sections can be adjusted manually, it is better to use optimization techniques, such as genetic algorithms, to obtain the optimal results. In this paper, we propose two models to design optimal data-base and rule-base expert systems. The two models consider the intervals of the metrics thresholds and the selection of rules for the fault prediction expert system in the absence of faulty data. Our expert system can be used in both supervised and unsupervised fault prediction modeling. Apart from mentioned issues, since most new projects do not have historical data, using data from one project to predict defects in another come to the picture (cross-project prediction), however, most experiments in cross-project defect prediction report poor performance [19, 20].

Several methods have been proposed by researchers in other engineering areas to develop an optimal knowledge base of the fuzzy logic expert system. Some of these methods, such as those proposed by Karr [21], NG and Lee [22], Rao and Pratihar [23], and Vundavilli et al. [24], used genetic algorithms to identify the optimal data-base or rule-base expert system. In the present study, an expert system based on fuzzy logic is developed to identify the optimal values of the intervals of the metrics thresholds as well as the optimal rules. Our method allows practitioners to detect and prioritize modules based on their fault proneness estimations by using simple rules that

are based on software measurement thresholds. We evaluated the performance of the obtained fault prediction models using seven publicly available datasets [13] including three datasets from a Turkish white-goods manufacturer that develops embedded controller software and four datasets from the NASA repository. In addition, we empirically compared the fault prediction abilities of the proposed models with those based on other existing approaches. The results show that the generated rules can be used to guide testing efforts and improve the quality of the developed software systems, especially when developers have a limited time frame and budget. More specifically, the results show that suggested membership intervals and rules obtained from our proposed model can improve the overall performance of the fault prediction models compared to other existing models in terms of a false negative rate.

The main contributions of the paper are as follows:

1. Propose two fuzzy expert systems that can predict software fault proneness, especially when the faulty data are absent or historical data are not available.
2. Propose a set of membership value intervals based on six software measurement metrics as well as a selection of fuzzy rules by optimizing the fuzzy expert fault prediction model to improve the process of predicting software faults, Hence, the role of experts become less critical and more supportive.

The paper is organized as follows. Section 2 reviews the related research. The two methods for building fault-prediction models are presented in Section 3. Section 4 reports and discusses an empirical evaluation study. Section 5 discusses validity threats to the empirical study. Finally, Section 6 concludes the paper and outlines possible future work.

2. Related Research

In this section, we summarize and discuss relevant research in the area of fault-proneness prediction. In addition, we provide a brief overview for two research topics that are considered in this paper for building fault prediction models, including fuzzy profile development and genetic algorithms.

2.1. Overview of Relevant Work

Different techniques have been used for software fault prediction, such as neural networks [25], naïve Bayes [26], logistic regression [27], case-based reasoning [28], and the artificial immune recognition system algorithms in [3, 29-31]. As we studied the research papers, we found that most papers concentrated on supervised approaches; however, there are lesser studies related to semi-supervised and unsupervised learning in the field of software fault prediction, despite their importance and necessity. Thus, in this section, the few available semi-supervised and unsupervised studies on software fault prediction are reviewed.

Predicting fault-prone and non-fault-prone modules in the absence of faulty data is usually done based on one of the following methods:

1. Cluster the data and ask the expert to create a prediction based on statistical information about the data [1, 14, 32].
2. Cluster the data and use metrics thresholds provided by authorized companies to decide on the label of the data [13, 15, 16].
3. Use metrics thresholds provided by authorized companies to decide on the label of the data [15, 16].

Studies using these three methods are summarized and discussed as follows. Zhong et al. [14, 32] employed the clustering technique, k-means and neural-gas, with expert assistance, to build the model using KC2 datasets from NASA. Experts analysed the cluster fault proneness using statistical data provided by each cluster. Zhong et al. [14, 32] repeated their experiments on a large dataset from NASA, JM1. From both studies, they concluded that the neural-gas performed slightly poorer for JM1 and only slightly better for KC2 than k-means, considering the overall error rate as part of performance evaluation metrics. However, their approach depends on the availability of a software quality expert with at least 15 years of experience as they claimed, and the procedure cannot be fully automated as a result.

A constraint-based semi-supervised clustering model was proposed by Seliya and Khoshgoftaar [1], which used quality experts to identify clusters' fault proneness in an iterative manner. Six NASA datasets were used to evaluate the model performance based on information obtained during the semi-supervised approach using the largest dataset known as JM1. The clustering was applied to classify modules in several other remaining datasets. They concluded that their semi-supervised proposed model could make better predictions compared with unsupervised approaches. Their model also depends on the availability of software quality experts and therefore cannot be fully automated.

Catal et al. [15] employed clustering and software measurement metrics threshold values for building fault prediction models. They claimed that they could minimize the need for experts' assistance in identifying fault-prone modules from non-fault-prone modules. They proposed two prediction models. The first one is simply comparing the module's metrics with the corresponding metrics threshold values. However, in the second method, all modules inside the dataset were initially clustered into 20 groups using k-means, and the one-stage approach was performed using a representative of each cluster against the metrics threshold values. Their assumption for labeling the modules as faulty was that at least one metric of the representative module needed to be higher than the corresponding metrics threshold value. Because identifying the number of clusters was one of the model's drawbacks, Catal et al. [16] used x-means clustering to address the issue. They conducted the experiment on the same dataset as a Turkish white-goods manufacturer that developed embedded controller software [4]. They stated that their model could be automated; however, the accuracy of their proposed model still does not meet expectations. In addition, using a fixed condition for considering the fault proneness of the module is highly dependent on the nature of the dataset. Because they evaluated their models based on only three small datasets, the model cannot be generalized to all datasets.

The quad tree-based k-means (QDK) method proposed by Bishnu and Bhattacherjee [13] was one the recent models that is used for fault prediction. First, the initial cluster centres were identified using quad trees to improve the k-means algorithm. Furthermore, the quad tree-based approach was employed for the fault-prediction process. Both the QDK model and Catal et al. [15] used same metrics threshold values, datasets, and performance evaluation

metrics. They claimed that QDK could perform as well as the k-means algorithm and well-known supervised approaches namely, the linear discriminate analyses and naive Bayes. QDK also outperformed two models proposed by Catal et al. [15]. However, the QDK-based approach has limitations, such as deciding about the best cluster number. Second, similar to the Catal et al. [15] model, they both used small datasets to evaluate their models, which cannot be generalized to all industrial datasets. Finally, the model cannot be automated due to its strict dependency on the users.

The NSGLP method proposed by Zhang et al. [33] used graph based semi-supervised learning technique which employed Laplacian score sampling strategy for the labeled defect-free modules. Their method has three phases; a class-balance labeled training dataset is constructed at first and then a nonnegative sparse algorithm is used to compute the nonnegative sparse weights of a relationship graph that serve as clustering indicators. Lastly, a label propagation algorithm is employed on the non negative sparse graph to iteratively predict the labels of unlabeled software modules. They stated that NSGLP outperforms several representative state-of-the-art semi-supervised software defect prediction methods, however, the accuracy of their proposed model still could be improved and they used 21 method-level metrics. In addition, NSGLP may not perform well when only few labeled data is available [34].

Jiang et al. [35] claimed that they proposed a new model to overcome the main two challenges in the area of software fault prediction which are difficulty to collect a large amount of labeled training data and the problem of imbalanced data set (fewer defective modules compared to defect-free modules). They proposed a semi-supervised learning approach named ROCUS, which used random committee as the ensemble of classifiers and under-sampling as the class-imbalance learning technique. Jiang et al. stated that ROCUS works better than those semi-supervised learning methods that ignore the class-imbalance nature of the task and a class-imbalance learning method that does not make effective use of unlabeled data. This method also has limitation similar to those mentioned in [33].

FTF is another proposed method by Lu et al. [36] which used random forest as the base supervised learner, and the self-training algorithm is a variant of the original Yarowsky's algorithm. In the FTF algorithm, a base supervised learner is iteratively trained from both labeled and unlabeled data until some stopping criterion is met.They reported that semi supervised learning improves fault prediction only if the number of initially labeled software modules exceeds 5%. FTF method limitations are same as the ones reported in [33, 35].

Apart from the mentioned related works, there are other research papers, which proposed semi-supervised learning method but applied it on different sets of datasets. Li et al. [37] proposed CoForest method based on a sample and the remaining un-sampled modules. They also proposed another semi-supervised learning method called ACoForest, which can actively select several informative un-sampled modules for testing while automatically exploiting the remaining un-sampled modules for better performance. Lu et al. [38] proposed active learning as a way to automate the development of models which improve the performance of defect prediction between successive releases (three successive versions of Eclipse).

As mentioned before, managing prediction methods using unlabeled data is possible with the help of clustering and software experts or metrics thresholds. However, as reviewed previously, each has its own problems.

Therefore, we designed and developed our automatic hybrid fuzzy expert prediction model to overcome the above mentioned problems, especially the problem that occurs when module labels can be changed based on metrics threshold value intervals, and the results vary by changing the rule conditions. In fact, considering the fixed values for thresholds and rule conditions is not the optimal solution. Existing fuzziness in metrics threshold value intervals and rules lead us to use an expert system based on fuzzy logic, which will be explained in detail later in this paper.

It should be mentioned that, some other studies employed fuzzy logic based approach for fault prediction [39-41]; however, all of them used other measurement metrics such as object oriented metrics and hence, they cannot be compared with our model.

2.2. Fuzzy Profile Development

Fuzzy logic by Zadeh [42] was introduced based on multi-valued logic proposed by Lukasiewicz [43, 44]. A fuzzifier, fuzzy rules, fuzzy inference engine, and defuzzifier are four main components of any fuzzy logic system. Linguistic variables and membership functions are employed to convert input crisp values into a fuzzy set, which is called the fuzzification step. Fuzzy rules are generated after the fuzzification step is finished. A fuzzy rule has the same structure as an IF–THEN rule, which is presented in the rule-base. An inference engine, which is a set of rules defined in the fuzzy rule-base, is used as the basis for interpreting and employing reasoning fuzzy outputs that are generated. The defuzzifier maps the fuzzy sets to a single number as the output.

In other words, in a fuzzy classification approach, a collection of n data objects ($x^{(1)}, x^{(2)}, \dots, x^{(n)}$) is represented by a set of m attributes $x_1^{(n)}, x_2^{(n)}, \dots, x_m^{(n)}$. Each attribute in $x^{(n)}$ shows a set of t discrete linguistic variables $LV(x_m^{(n)}) = \{LV_{m1}, LV_{m2, \dots}, LV_{mt}\}$. Each input vector is classified into p different fuzzy sets, which are shown by $FS_1, FS_2, \dots, FS_p$. The membership function $\mu_{FS(i)}(x^{(j)})$ shows the degree to which $x^{(j)}$ belongs to FS_i. In addition, the membership value $\mu_{LV(i)}(x_q^{(j)})$ depicts the degree to which attribute q of input vector $j^{th}x$ belongs to linguistic variable i (Please refer to Eqs. 1 and 2).

The knowledge section of a fuzzy system consists of two parts: the data-base and rule-base [45]. The data-base part addresses the partitioning of the input space and the determination of the membership functions. According to the literature reviews, there are many ways to determine the membership functions. Some researchers classified the membership functions into four main categories, which are subjective evaluation and elicitation, converted frequencies and probabilities, physical measurements, and ad-hoc forms and methods [46, 47]. In subjective evaluation and elicitation, fuzzy sets can be determined by certain elicitation procedures, which are usually provided by the experts in the problem area. In converted frequencies and probabilities, membership functions are constructed based on the information probability curves and frequency histograms. Physical measurements use unique measurements of the features for defining membership functions. In the learning and adaptation approach, the membership functions of fuzzy sets can be learned and adapted from a given set of functions. Some other researchers [48, 49] categorized this method into three main sections: first, subjective evaluation and elicitation, including measurement-theoretic approaches [50], intuition-based approaches [51], and probabilistic approaches [52]; second, heuristic methods and parameterized functions [48]; and third, estimation methods using synthetic

and real datasets, including neural network techniques [53] and curve fitting methods [54]. In our experiment, which will be described in Section 5.3, we used a subjective evaluation and elicitation approach.

There are two major approaches for implementing a fuzzy logic system (FLS): the Mamdani and Assilian [55, 56] and Takagi and Sugeno [57] approaches. The consequent part of the rules in Takagi and Sugeno FLS are not fuzzy, whereas in Mamdani and Assilian, the fuzzy and the fuzzification process is also needed. The Mamdani and Assilian rule-base part, along with Takagi and Sugeno's, are represented in Eqs. 1 and 2, respectively.

$$\text{IF } (x_1^{(i)} \text{ is } LV_1^{(i)}) \text{ AND } (x_2^{(i)} \text{ is } LV_2^{(i)}) \text{ AND} \ldots \text{ AND } (x_k^{(i)} \text{ is } LV_k^{(i)}) \text{ THEN}(FS \text{ is } FS_k) \quad (1)$$

$$\text{IF } (x_1^{(i)} \text{ is } LV_1^{(i)}) \text{ AND } (x_2^{(i)} \text{ is } LV_2^{(i)}) \text{ AND } \ldots \text{ AND } (x_k^{(i)} \text{ is } LV_k^{(i)}) \text{ THEN } y = f(x) \quad (2)$$

As also explained at the beginning of this section, $x_1^{(i)}, x_2^{(i)}, \ldots, x_k^{(i)}$ are casual factors and FS is a decision. $LV_1^{(i)}, LV_2^{(i)}, \ldots LV_k^{(i)}$ are fuzzy sets representing the k^{th} rule and FS_k is the k^{th} rule of fuzzy set. In other word, $LV_1^{(i)}, LV_2^{(i)}, \ldots LV_k^{(i)}$ and FSk are represent linguistic terms. In Eq. 2, y is defined by a function $(f(x) = a_0 + a_1x_1 + a_2x_2 + \ldots + a_nx_n)$, that could be constant (0/1). To show how our problem fits to Eq. 2, one rule could be written as follows:

IF (*LOC* is *L*) AND (*CC* is *L*) AND (*UOPR* is *H*) … AND (*TOPND* is *L*) THEN *Fault = 0*

Or

IF (*LOC* is *H*) AND (*CC* is *H*) AND (*UOPR* is *L*) … AND (*TOPND* is *H*) THEN *Fault = 1*

Fuzzy rules are generated for each fuzzy partition; the number of these rules is equal to that of the fuzzy partitions. For example, in the case of a four-input and single output fuzzy system with 3 linguistic partitions for each input, 81 rules are generated. There are different automatic and manual ways of generating the rules. Fuzzy rules can be extracted through heuristic procedures, decision trees, neuro-fuzzy techniques, genetic algorithms, rough set theory, and so on. In general, the required knowledge about defining membership functions as well as rule induction can be gathered from experts, databases, course materials, and flow diagrams.

2.3. Genetic Algorithm

Genetic algorithm (GA) is a population-based algorithm [58] that is considered one of the meta-heuristic algorithms. The GA learning process is similar to a competition among the population of all candidates in a problem solution. The GA goal is to identify the best chromosome along with the best grouping based on its fitness value while applying the best fitness chromosome. The decision as to whether any chromosome will contribute to the next generation of the solution is made based on the value of the fitness function.

GA was inspired by the Darwinian theory of evolution using operators such as population, mutation and crossover to transform a population of solutions into a new population. The population is a collection of candidate solutions that are generated in the first step of the algorithm. To decide on the selection of the best solution, the fitness function is employed. The number of solutions is selected based on the fitness function values during the selection process. The selected solutions (individuals) are used in the breeding (or crossover) process based on their fitness values to produce two new individuals. In each generation, an individual can be mutated, which results in small changes in the individuals.

After each round of simulation, the goal is to identify how close the M new solutions are to the overall goal and delete a specific number of the worst solutions. This step was done based on analysing the fitness value, which is calculated from the fitness function that results from each solution. Eq. 3 shows the sample fitness of GA (MSE), where *N* is the total number of training sets, and act_i and $pred_i$ show actual and predicted values, respectively, for the i^t training sample.

$$\text{fitness} = \frac{1}{N}\sum_{i=1}^{N}(\text{act}_i - \text{pred}_i)^2 \qquad (3)$$

3. Proposed Methods

In the present study, we used Takagi and Sugeno's approach [57] of FLS to model the fault prediction process that is based on six inputs and one output. The *y* part of Eq. 2 is constant (0 or 1), which represents the faultiness or non-faultiness of the program module. The line of code (LOC), cyclomatic complexity (CC), unique number of operands (UOPRD), unique number of operators (UOPR), total number of operands (TOPRD), and total number of operators (TOPR) are considered as input parameters. We considered these six metrics because their empirically-based thresholds are provided by Integrated Software Metrics, Inc. (ISM) [18] and because they are considered in similar studies (i.e., [4, 7]), which allows for a result comparison. Only two linguistic variables are considered for each of the inputs, which are Low and High. The shape of the membership functions is considered trapezoidal. Furthermore, the faultiness or non-faultiness of the input module is considered an output. The schematic diagram, which shows the relationship between the inputs and outputs of the proposed system, is shown in Fig. 1.

This paper presents two methods – Method 1: Implementation of a manually designed FLS. Method 2: Adjusting the rule-base and data-base of the FLS with genetic algorithm (GA), which is explained below.

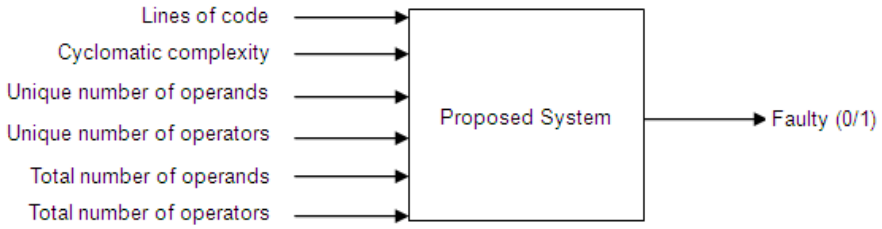


Fig.1. Input and output variables of the fault prediction process

3.1. Method 1: Implementation of Manually Designed Fuzzy Logic System (FLS)

Based on expert opinions, user expectations, software requirements, records of existing field data from a previous release or similar system [59], and metrics thresholds, a linguistic variable can be determined. Note that the membership function distributions and rule-base of the fuzzy logic system are developed with experts assistance, a literature review [3], and numerous trial and error processes. In this paper, we used method-level metrics thresholds. Menzies et al. [26] claimed that useful software measurements metrics for software fault prediction are method-level metrics [60], and knowledge about their corresponding threshold values can be found in both the literature and the predictive tool documentation provided by the integrated software metrics group [18] (Refer to Table 1).

Because there are two linguistic variables for each input, a total of 64 (i.e., 2^6) rules need to be accounted for initially for the manually constructed FLS. For visualization purposes, the membership functions and fuzzy profiles of all selected input/output variables are shown in Fig. 2. The half-base widths, two for each input variable *A1, A2, B1,B2, C1, C2, D1, D2, E1, E2, F1*, and *F2* of trapezoidal membership function distributions are all decided based on literature reviews, expert advice, and a number of tests. The manually constructed system was built based on different training datasets and was tested with other unseen datasets. The overall structure of Method 1 is shown in Fig. 3. The manually developed rule-base for the fault prediction process is given in Appendix A as well.

Table 1: Description of the parameters

No	Parameter description	Notations	Low	High	Threshold values
1	Line of code	LOC	40	80	65
2	Cyclomatic complexity	CC	8	15	10
3	Unique number of operands	UOPND	15	35	25
4	Unique number of operators	UOPR	25	60	40
5	Total number of operands	TOPND	110	145	125
6	Total number of operators	TOPR	50	90	70

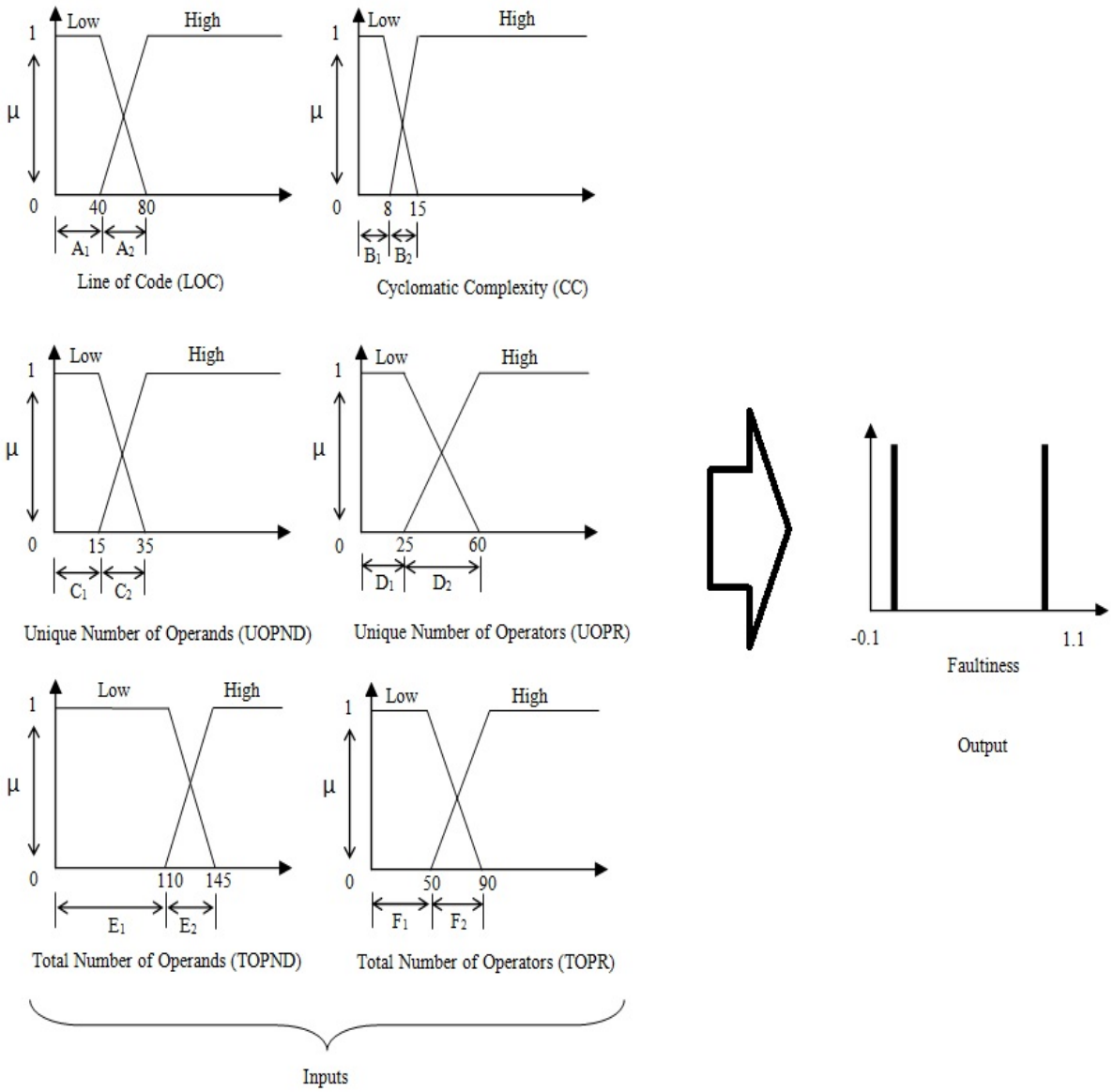


Fig. 2. Manually defined membership function distribution of the input-output variables

As shown in Fig. 3, building the FLS model starts with defining the rules and membership function intervals using the threshold values provided by Integrated Software Metrics, Inc. (ISM) [18]. The membership function's intervals as well as the rules' conditions have been continuously changed based on the trained datasets in order to show the improved performance results using the performance evaluation criteria listed in section 4.2.

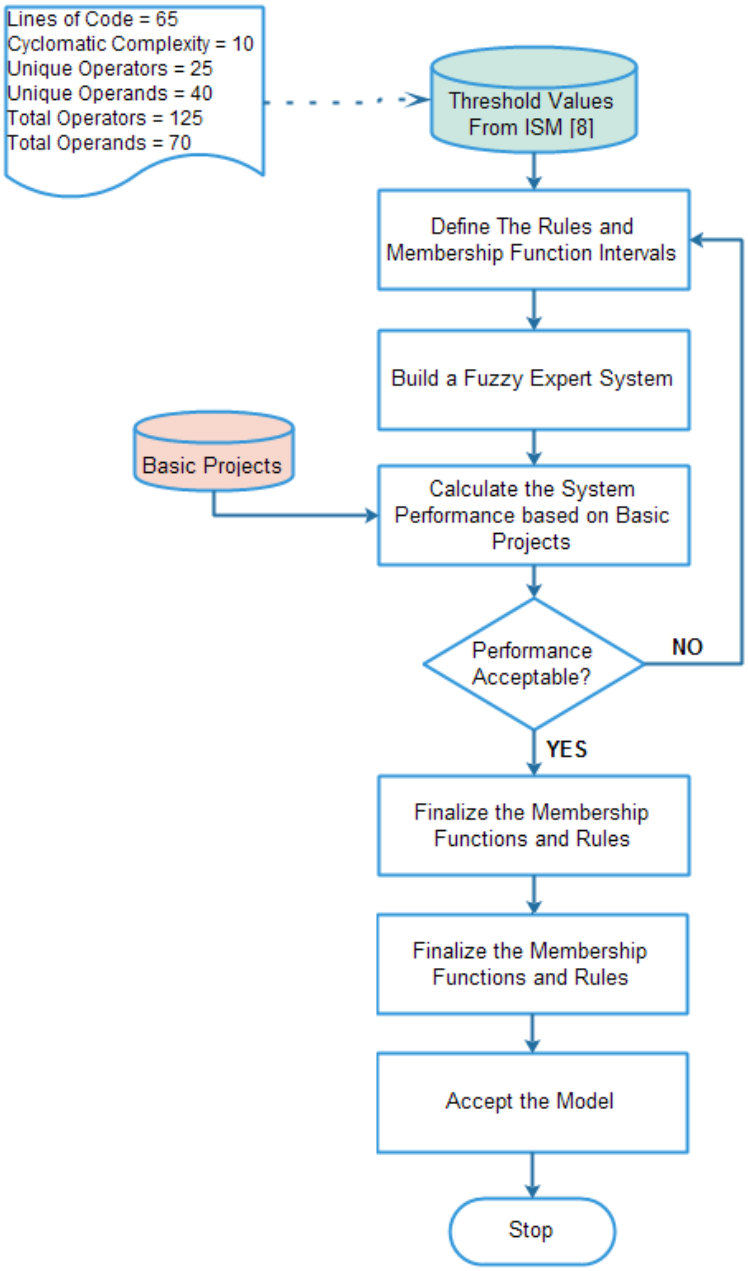
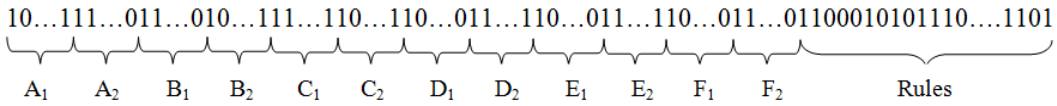


Fig. 3. Process steps in Method 1 (FLS)

3.2. Method 2: Adjusting the Rule-base and Data-base of the Fuzzy Logic System with Genetic Algorithm (GA_FLS)

In the second method, the manually constructed FLS containing the data-base and rule-base parts are both optimized by using the genetic algorithm. To apply a genetic algorithm, binary coding, a steady-state selection, uniform crossover, and a bitwise mutation are used. Each chromosome includes information represented by A through F values (refer to Fig. 2) of all input values, followed by the rule values as shown below. In addition, the following genetic algorithm parameters are found to be sufficient for giving the best results. The crossover and mutation probability are considered 0.6 and 0.005, respectively. Moreover, the population size is set to 150, and 50 is assigned for the maximum number of iterations.



The length of the chromosome is considered to be 184, which contains 10 bits for each A through F value of six inputs, which has a total of 120 bits, followed by 64 bits in which each one represents the presence or absence of each rule. For optimization purposes, the suitable ranges of variation for the A through F values are decided after careful study, and the initial ranges are shown in Table 2 for each input value. With this method, good rules are also identified in the rule-base. The overall structure of Method 2 is shown in Fig. 4.

Table 2: Initial base values (A through F) for optimization.

Representative letters	Value ranges	Representative letters	Value ranges
A_1	20 to 50	D_1	15 to 35
A_2	50 to 100	D_2	35 to 75
B_1	5 to 10	E_1	80 to 110
B_2	10 to 25	E_2	110 to 150
C_1	10 to 20	F_1	35 to 60
C_2	20 to 40	F_2	60 to 100

As shown in Fig. 4, the process of building GA_FLS starts by generating the binary string, including membership values and rules, by defining the membership ranges using Table 2, which are selected based on threshold values provided by Integrated Software Metrics, Inc. (ISM) [18]. In the next stage, the expert system is built using the fuzzy logic system and genetic algorithm. The performance of the model is calculated based on the fitness function and is evaluated using performance metrics. If the performance is not acceptable, the best binary string based on the fitness function is selected, the crossover and mutation operators are applied, and the new expert system is modeled based on new parameters.

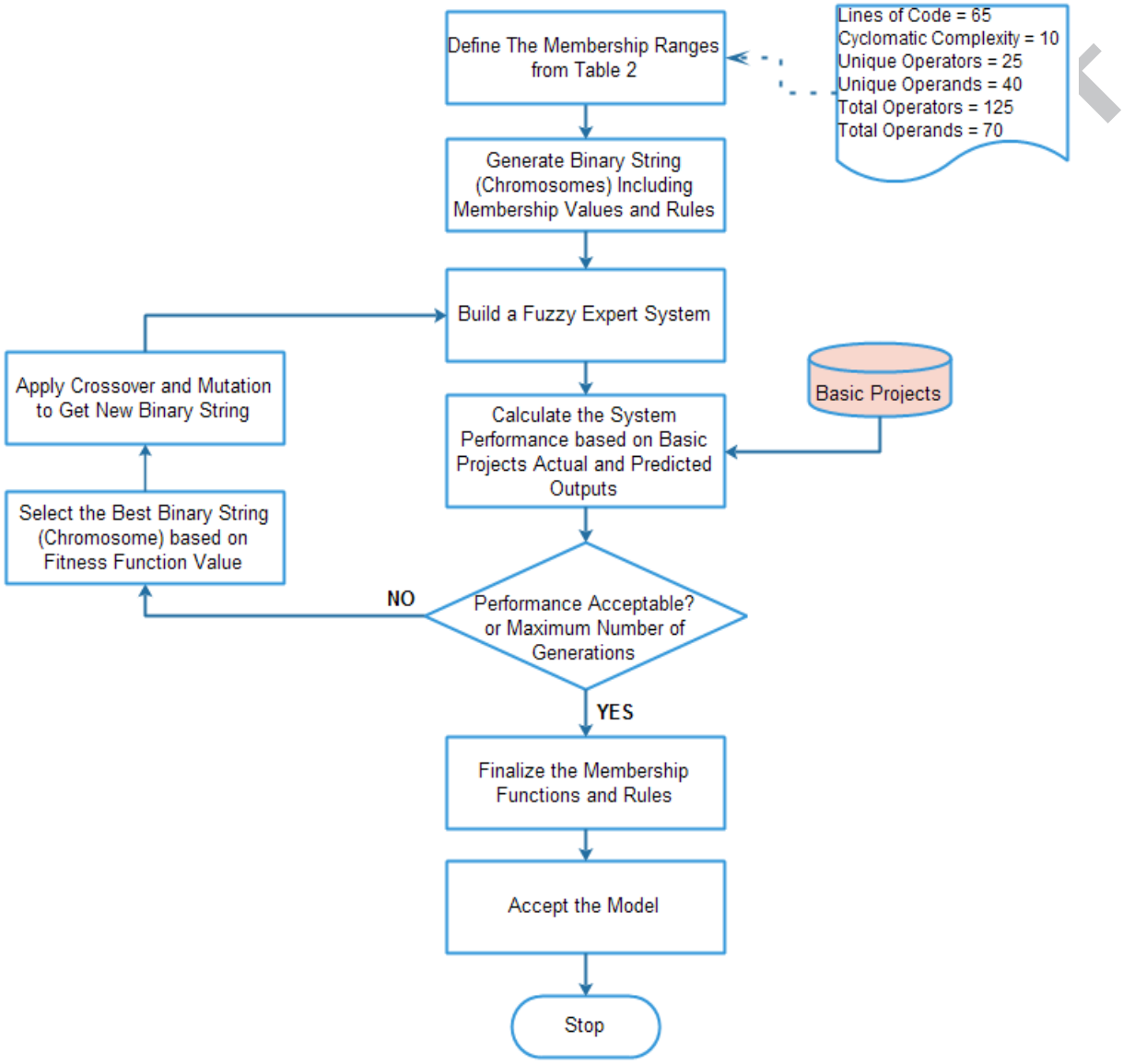


Fig. 4. Process steps in Method 2 (GA_FLS)

One of the key factors of optimizing the performance of the expert system is the way in which the chromosomes are selected for addition or elimination based on the fitness function value in the GA process. Because obtaining a good fault prediction model depends on minimizing the two metrics: false positive rate and false negative rate, we consider the fitness function as a combination of these two factors. The selected chromosomes for addition or elimination are chosen based on the minimum or maximum of the sum of these factors, respectively. An explanation about the false positive and false negative rates are presented in section 4.2.

4. Experimental Description and Results

4.1. Dataset Selection

The proposed models are evaluated based on seven datasets [61]. Three datasets are from a Turkish white-goods manufacturer developing embedded controller software, namely, AR3, AR4, and AR5, which we refer to as the Turkish set. Four more datasets, belonging to NASA software, were also chosen to analyse the performance of the proposed model through larger datasets. From now on, we will refer to them as NASA sets. Brief explanations about each of the datasets are presented in Table 3. The measurement metrics presented in the Turkish and NASA sets are varied; however, only six are chosen for the fault prediction process as the dimensions of the datasets should be the same as [13, 15] for comparison purposes. Table 4 provides the descriptive statistics for each of the six metrics being considered, including the minimum, 25% quartile, mean, median, 75% quartile, maximum value, and standard deviation. According to statistics presented in Table 4, All NASA datasets are having similar range of LOC, CC, UOPND values in terms of minimum, 25% quartile, mean, median, 75% quartile, and standard deviation, except for the maximum values, MW1, PC4, KC3, KC1, and CM1 have similar range of values. On the other side, PC1, PC2, and PC3 values are near each other. P5 and JM1 have similar maximum values since they are big datasets compared to others. In terms of UOPR, interestingly, MW1 and JM1 have similar maximum range values as well as KC3, KC2, and PC4. In addition, in terms of TOPR, PC3 and JM1 maximum range values are similar. Moreover, in TOPND, P5, PC3, and JM1 maximum values are close to each other. In Turkish set, AR3 and AR5 values are more similar. By comparing statistics between NASA and Turkish set, we identified that in terms of UOPND, TOPR, and TOPND value ranges, all three Turkish datasets are close to KC1 and KC3.

Table 3: Dataset descriptions

Name	Description	Programming Language	No. of modules	Defective Modules %
AR3	Controller software for washing machine	C	63	13%
AR4	Controller software for dishwasher	C	107	19%
AR5	Controller software for refrigerator	C	36	22%
CM1	NASA spacecraft instrument project	C	498	10%
PC1	Flight software for earth-orbiting satellite	C	1109	7%
PC2	Flight software for earth-orbiting satellite	C	5589	0.4%
PC3	Flight software for earth-orbiting satellite	C	1563	10%
PC4	Flight software for earth-orbiting satellite	C	1458	18%
PC5	Flight software for earth-orbiting satellite	C++	17186	5%
KC1	Storage management for receiving and processing ground data	C	1183	29%
KC2	Science data-processing unit of storage management system used for receiving and delivering ground data	C++	522	22%
KC3	Storage management for receiving and processing ground data	JAVA	458	9%
MW1	Software application from zero-gravity-combustion experiment	C	403	8%
JM1	A real time predictive ground system	C	10883	19%

Table 4: Descriptive Statistics for Each of the Metrics

Name	Metrics	Minimum	25% Quartile	Mean	Median	75% Quartile	Maximum	Standard Deviation
CM1	LOC	2	9	32.3485	19	33.5	423	44.7190
	CC	1	1	5.8109	3	6	96	8.7561
	UOPR	1	9	16.082	15	20	72	9.6657
	UOPND	0	8	27.8929	17	33	314	35.3258
	TOPR	1	19	96.8974	47	105	1261	140.6658
	TOPND	0	11	60.9954	29	69	814	90.7808
PC1	LOC	0	8	25.7177	14	29	602	37.2830
	CC	1	2	6.0866	3	7	136	9.52084
	UOPR	1	9	14.2124	13	18	99	8.1349
	UOPND	0	7	22.7938	14	27	538	30.6174
	TOPR	1	18	73.8826	37	80.75	1641	118.4475
	TOPND	0	13	56.6374	28	61	1144	91.3591
PC2	LOC	0	5	16.746309	10	17	663	33.827024
	CC	1	2	4.3637584	3	5	144	6.9375499
	UOPR	4	8	11.774497	11	14	46	4.954106
	UOPND	1	5	12.451007	8	14	245	17.752573
	TOPR	4	13	46.044295	24	44	1198	84.540643
	TOPND	1	9	31.748993	16	31	843	57.898782
PC3	LOC	0	10	29.922006	18	33	817	48.039341
	CC	1	3	7.273909	4	7	299	12.589538
	UOPR	4	11	15.232126	14	18	68	6.413129
	UOPND	2	10	27.753018	18	33	768	44.345806
	TOPR	5	25	90.471681	46	91	5590	225.77946
	TOPND	2	18	73.264624	36	73	4015	182.94978
PC4	LOC	0	6	20.6138546	12	26	210	25.234081
	CC	1	1	4.83950617	3	6	94	6.6682565
	UOPR	0	7	15.2321263	11	15	38	6.6167031
	UOPND	0	5	27.7530176	9	16	601	20.535942
	TOPR	0	15	90.4716806	33	73	1687	103.99983
	TOPND	0	9	73.264624	19	44	1403	71.446722
PC5	LOC	0	2	9.4085302	2	2	2072	62.373684
	CC	1	1	2.0381124	1	1	366	7.6985314
	UOPR	0	3	4.4416967	3	6	85	4.2796491
	UOPND	0	0	4.4765507	1	3	2241	25.93173
	TOPR	0	3	23.857326	3	7	10862	209.57392
	TOPND	0	0	14.668277	1	3	5169	126.34343
KC1	LOC	1	10	31.950972	20	41	288	34.974092
	CC	1	1	4.0912933	2	5	45	4.7717921
	UOPR	0	6	10.501268	10	14	37	5.7671797
	UOPND	0	5	14.91885	11	21	120	13.720876
	TOPR	0	11	50.282333	27	66	678	61.964496
	TOPND	0	7	30.79459	17	40	428	38.358549
KC2	LOC	1	11	52.6736	30	60.75	1275	93.1063
	CC	1	2	6.82634	4	7	180	13.2936
	UOPR	1	7	11.8383	11	16	47	6.0620
	UOPND	0	6	20.3473	15	27	325	25.2700
	TOPR	1	12	83.8113	43	94.75	2469	172.284
	TOPND	0	8	54.0389	28	66	1513	108.7268
KC3	LOC	0	3	16.919214	7	18	242	26.472066
	CC	1	1	3.4475983	1	4	36	4.4929418
	UOPR	1	6	10.543668	9	15	31	6.1739246
	UOPND	0	2	15.681223	9	19	159	20.973123
	TOPR	1	6	59.194323	23	67	857	98.178974
	TOPND	0	3	34.384279	13	36	556	59.918807
MW1	LOC	3	9	21.4802	15	28	112	18.6528
	CC	1	1	4.6728	3	6	28	4.8226
	UOPR	2	13	38.3192	24	53	396	40.1048
	UOPND	3	17	48.9920	32	63	493	50.2769
	TOPR	2	9	21.3667	16	31	107	16.8949
	TOPND	3	7	11.6279	10	15	44	6.4673

Table 4: Continue

Name	Metrics	Minimum	25% Quartile	Mean	Median	75% Quartile	Maximum	Standard Deviation
JM1	LOC	1	15	46.25084	26	51	3442	80.5752
	CC	1	2	6.781933	4	7	470	14.07788
	UOPR	0	9	13.81457	13	17	411	10.04654
	UOPND	0	8	21.3945	15	25	1026	28.93176
	TOPR	0	21	87.58879	44	90	5420	165.4444
	TOPND	0	14	59.78553	30	62.75	3021	109.2398
AR3	LOC	3	18	89.2698	57	114.5	670	116.7320
	CC	1	1	13.1746	5	16.5	85	17.9178
	UOPR	3	6	12.2539	11	16	31	7.0641
	UOPND	4	13.5	37.3015	25	51	142	32.7392
	TOPR	8	33.5	137.2222	74	166.5	817	163.6271
	TOPND	5	25.5	103.2222	58	124	575	117.4031
AR4	LOC	6	21.5	73.7619	50	85.5	324	75.6977
	CC	1	2	7.61904	5	10	37	8.1209
	UOPR	4	7	11.8412	12	16	28	5.7085
	UOPND	4	9.5	23.0952	20	30	94	17.3165
	TOPR	9	27	84.3492	57	110.5	401	87.6935
	TOPND	5	17.5	55.8095	37	73.5	261	56.347
AR5	LOC	5	20.5	75.8888	42	112.75	477	90.1296
	CC	1	1.75	13.7777	7	21	93	18.2069
	UOPR	3	6	12.5833	11	16.25	29	7.5114
	UOPND	5	14.75	35.0277	26.5	46.5	150	29.8677
	TOPR	9	34.75	127.4722	58.5	210.25	699	142.3760
	TOPND	5	28	93.3333	51.5	153.25	482	98.8265

4.2. Performance Measurement Criteria

Four performance evaluation metrics are used for evaluating the proposed model: false positive rate, false negative rate, overall error rate, and Matthews's correlation coefficient [62]. After specifying the module labels, each of the evaluation metrics is calculated based on the confusion matrix. The condition of false negative (FN) occurs when the actual label is faulty, and we predict that it is non-faulty and false positive (FP) otherwise. We obtain the condition of true positive (TP) and true negative (TN) when the predicted labels are the same as the actual labels. The following equations are used to calculate the three evaluation metrics.

$$\text{False positive rate (FPR)} = \frac{\text{FP}}{\text{FP} + \text{TN}} \quad (4)$$

$$\text{False negative rate (FNR)} = \frac{\text{FN}}{\text{TP} + \text{FN}} \quad (5)$$

$$\text{Overall error rate} = \frac{\text{FN} + \text{FP}}{\text{TP} + \text{FN} + \text{FP} + \text{TN}} \quad (6)$$

$$\text{MCC} = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP} + \text{FP}) \times (\text{TP} + \text{FN}) \times (\text{TN} + \text{FP}) \times (\text{TN} + \text{FN})}} \quad (7)$$

$$\text{F} - \text{measure} = \frac{2 * \text{TP}}{2 * \text{TP} + \text{FP} + \text{FN}} \quad (8)$$

4.3. Results and Discussion

Initially, the proposed system was developed based on the Mamdani and Assilian [55, 56] approach to show the degree of faultiness based on five linguistic output variables (very low, low, medium, high, very high). However, the model should be compared with existing prediction algorithms and other studies. Mamdani's method was substituted with Takagi and Sugeno's [57] to have the output data be either faulty or non-faulty. Two different methods have been developed to calculate the faultiness of the program modules and improve the prediction accuracy in the absence of class or module labels.

4.3.1. Method 1: Implementation of Manually Designed Fuzzy Logic System (FLS)

As described in Section 3.1, all A through F values related to each input membership function are decided manually based on literature reviews and software quality expertise (Fig. 2). After the development of the fuzzy expert system, the prediction accuracy, with the help of 14 different datasets (NASA set and Turkish set), is calculated. Table 5 shows the performance evaluation metrics in terms of the FPR, FNR, overall error rate, MCC, ands. These results will be compared and discussed later in this paper in section 4.3.3.

Table 5: Results of the module fault proneness using Method 1, manual fuzzy logic system (FLS)

Testing dataset	Overall error	FPR	FNR	MCC	F-measure
CM1	0.2764	0.2603	0.4130	0.2205	0.3930
KC1	0.2506	0.1083	0.6701	0.3428	0.4086
KC2	0.2067	0.1333	0.3979	0.4686	0.4493
KC3	0.1935	0.1962	0.1764	0.4793	0.3218
MW1	0.1680	0.1495	0.3928	0.3125	0.2666
JM1	0.2691	0.1919	0.5623	0.2325	0.4181
PC1	0.2949	0.2078	0.4193	0.2781	0.3255
PC2	0.1000	0.0852	0.7500	0.2901	0.1818
PC3	0.2603	0.2344	0.4402	0.1697	0.3703
PC4	0.2195	0.2013	0.4400	0.2266	0.2031
PC5	0.2819	0.3425	0.1529	0.4705	0.5068
AR3	0.3968	0.4181	0.2500	0.3142	0.4800
AR4	0.2710	0.2758	0.2500	0.3840	0.3710
AR5	0.2500	0.2857	0.1250	0.4969	0.3924

4.3.2. Method 2: Adjusting Rule-base and Data-base of the Fuzzy Logic System with Genetic Algorithm (GA_FLS)

According to the proposed model presented in Section 3, a genetic algorithm is used to adjust and tune all membership function distributions and rule-base of the fuzzy expert system. In this section, optimization is done based on the basic dataset from each of the NASA and Turkish sets. When the optimization is complete, the expert fuzzy system is tested for prediction accuracy with the rest of the remaining NASA and Turkish datasets. After the fuzzy expert system is optimized with each of the datasets and the final system is tested with the other remaining datasets, we observed that the results, after constructing the GA_FLS based on CM1 and AR4 from the respective NASA and Turkish datasets, show better comparison results with the others.

The prediction results on the tested datasets, based on the expert system that is constructed and optimized with CM1 and AR4, are presented in Tables 6 and 9. The A through F values, after optimization for CM1 and AR4, are

also shown in Tables 7 and 10, respectively. Moreover, the genetic algorithm selected 24 and 20 rules from a total of 64 rules (refer to Appendix A) based on the CM1 and AR4 optimization, which is shown in Tables 8 and 11, respectively. The rule number in the tables indicates the actual rule number, which is shown in Appendix A. Furthermore, Figs. 5 and 6 show the mean error performance changes in terms of the FPR, FNR, overall error rate, MCC, and F-measure during genetic optimization in different generations. The results presented in Tables 6 and 9 will be compared and discussed later in this paper in section 4.3.3.

Note that, some of the datasets such as KC1, KC2, JM1, and PC4 are more defective, hence, as reported in the result section (Table 5 and Table 6), their FNR values are not as good as other datasets; however, even with this issue, the GA_FLS model improves the performance of these datasets compared to manual FLS and almost other models.

Table 6: Results of the module fault proneness using Method 2 on the NASA set (GA_FLS)

Testing dataset	Overall error	FPR	FNR	MCC	F-measure
CM1	0.2424	0.2403	0.2608	0.3376	0.4102
KC1	0.2331	0.1003	0.6323	0.2997	0.4295
KC2	0.1586	0.1029	0.3283	0.5715	0.6766
KC3	0.2277	0.1901	0.4750	0.2119	0.5384
MW1	0.1481	0.1345	0.3181	0.3771	0.4054
JM1	0.2360	0.1637	0.5311	0.2910	0.4487
PC1	0.1982	0.1875	0.3529	0.2735	0.2959
PC2	0.1152	0.1051	0.5625	0.1199	0.1224
PC3	0.2385	0.2192	0.3877	0.2838	0.4117
PC4	0.2369	0.1798	0.5863	0.1125	0.2800
PC5	0.2841	0.2092	0.4772	0.3078	0.6575

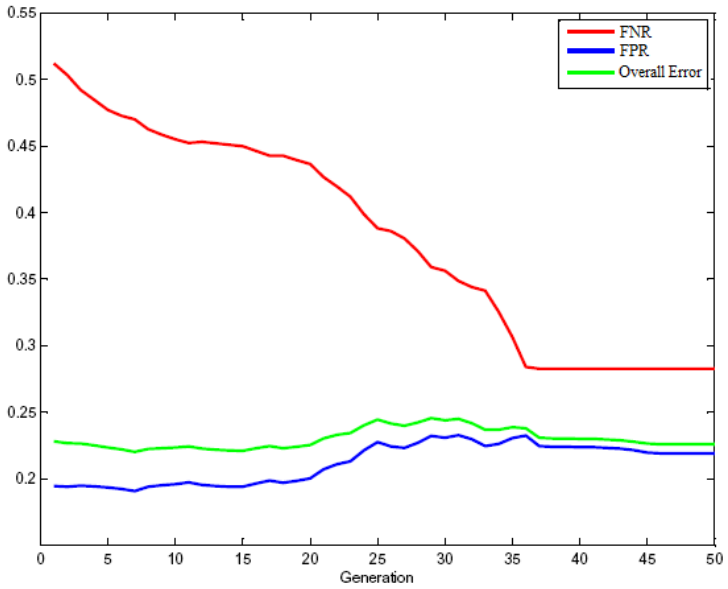


Fig. 5. Error performance changes during genetic optimization in different generations using Method 2 (GA_FLS) on CM1

As shown in Figs. 5 and 6, the FNR value in the optimization, based on CM1 from the NASA set, dramatically decreases, whereas the FPR and overall error rate remains approximately constant. All evaluation performance metrics are decreased based on AR4 from the Turkish set.

The reason we did not optimize the expert system based on the other systems in the Turkish set, including AR3 and AR5, is that they have very small data populations (i.e.,36 and 63, respectively).

Table 7: Base values (A through F) after optimization based on CM1

Representative letters	Value ranges	Representative letters	Value ranges
A_1	21	D_1	21
A_2	69	D_2	58
B_1	6	E_1	98
B_2	10	E_2	177
C_1	18	F_1	48
C_2	32	F_2	63

Table 8: Results of optimized rule-base of the fuzzy logic system obtained using Method 2 (GA_FLS) on the CM1 from NASA set, "L" Denotes "Low", "H" Denotes "High"

Rule	LOC	CC	UOPR	UOPND	TOPR	TOPND	Fault	Rule	LOC	CC	UOPR	UOPND	TOPR	TOPND	Fault
8	H	H	L	L	L	L	0	**32**	H	L	L	L	H	H	1
10	H	L	L	H	L	L	0	**33**	L	H	H	H	L	L	1
13	L	H	H	L	L	L	0	**36**	L	H	L	H	H	L	1
14	L	H	L	H	L	L	0	**37**	L	H	L	H	L	H	1
18	L	L	H	L	H	L	0	**38**	L	H	L	L	H	H	1
22	L	L	L	L	H	H	0	**43**	H	H	H	H	L	L	1
23	H	H	H	L	L	L	1	**48**	H	H	L	H	H	L	1
24	H	H	L	H	L	L	1	**49**	H	H	L	H	L	H	1
26	H	H	L	L	L	H	1	**50**	H	H	L	L	H	H	1
27	H	L	H	H	L	L	1	**56**	L	H	L	H	H	H	1
28	H	L	H	L	H	L	1	**62**	H	L	H	H	H	H	1
29	H	L	H	L	L	H	1	**64**	H	H	H	H	H	H	1

Table 9: Results of the module fault proneness using Method 2 on the Turkish Set (GA_FLS)

Testing dataset	Overall error	FPR	FNR	MCC	F-measure
AR3	0.3161	0.4250	0.3428	0.3076	0.4285
AR4	0.2065	0.2069	0.2912	0.4198	0.5490
AR5	0.1875	0.2083	0.1250	0.5962	0.7000

Fig. 6. Error performance changes during genetic optimization in different generations using Method 2 (GA_FLS) on AR4

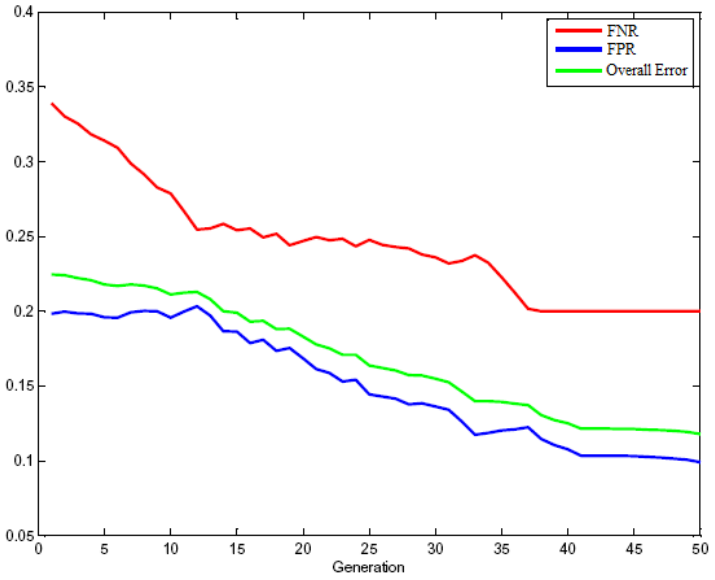


Table 10: Base values (A through F) after optimization based on AR4

Representative letters	Value ranges	Representative letters	Value ranges
A_1	20	D_1	31
A_2	84	D_2	58
B_1	8	E_1	126
B_2	19	E_2	144
C_1	19	F_1	42
C_2	36	F_2	95

Table 11: Results of optimized rule-base of the fuzzy logic system obtained using Method 2 (GA_FLS) on the AR4 from the Turkish set, "L" Denotes "Low", "H" Denotes "High"

Rule	LOC	CC	UOPR	UOPND	TOPR	TOPND	Fault	Rule	LOC	CC	UOPR	UOPND	TOPR	TOPND	Fault
1	L	L	L	L	L	L	0	**37**	L	H	L	H	L	H	1
3	L	H	L	L	L	L	0	**38**	L	H	L	L	H	H	1
7	L	L	L	L	L	H	0	**39**	L	L	H	H	H	L	1
10	H	L	L	H	L	L	0	**42**	L	L	L	H	H	H	1
20	L	L	L	H	H	L	0	**43**	H	H	H	H	L	L	1
23	H	H	H	L	L	L	1	**45**	H	H	H	L	L	H	1
28	H	L	H	L	H	L	1	**55**	L	H	H	L	H	H	1
29	H	L	H	L	L	H	1	**57**	L	L	H	H	H	H	1
34	L	H	H	L	H	L	1	**63**	L	H	H	H	H	H	1
36	L	H	L	H	H	L	1	**64**	H	H	H	H	H	H	1

4.3.3. Discussion and Comparison

Two proposed fault prediction methods based on fuzzy logic have been proposed, tested, and compared based on mentioned evaluation metrics with 14 different datasets. Two sets of comparison have been explored, which are

different as seen in the types of datasets (NASA/Turkish) and according to the available research studies. Moreover, the performance of our proposed model compared with selected well-performing supervised models based on the literature reviews [3, 63, 64] which are the random forest (RF), naive Bayes (NB), and logistic regression (LR).

In the first comparison, the results based on the NASA sets are compared with other existing approaches proposed by other researchers namely, the one (CO) and two-stage (CT) approaches [15], and three supervised learning algorithms random forest (RF), naive Bayes (NB), and logistic regression (LR). In the second comparison and in two more related articles, experiments were conducted based on the Turkish sets. These two additional methods are the quad tree-based k-means approach (QDK) [13] and the two-stage x-mean approach (CX) [16]. The explanation for each of these methods was given in section 2.1. For simplicity, we refer to the first and second proposed methods as FLS and GA_FLS. The visual representation of the comparison in the NASA and Turkish sets are shown in Figs.7 and 8, respectively.

The results of applying Wilcoxon test to the predictors' overall error rate are shown in Table 11 and Table 12. Since the p-value obtained from the Wilcoxon signed rank test are less than 0.05 for all the comparisons, the existence of a significant difference between the GA_FLS predictions and those achieved by the other predictors is substantially proved.

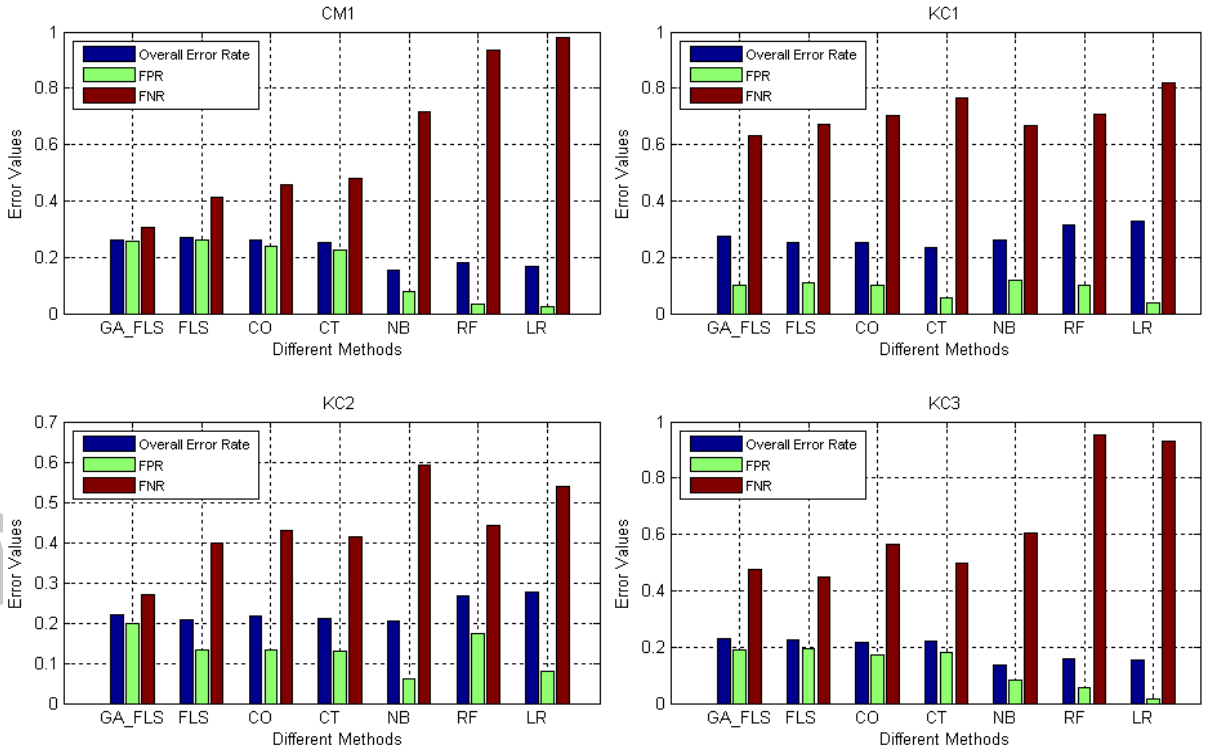


Fig. 7. Comparison results based on the NASA set

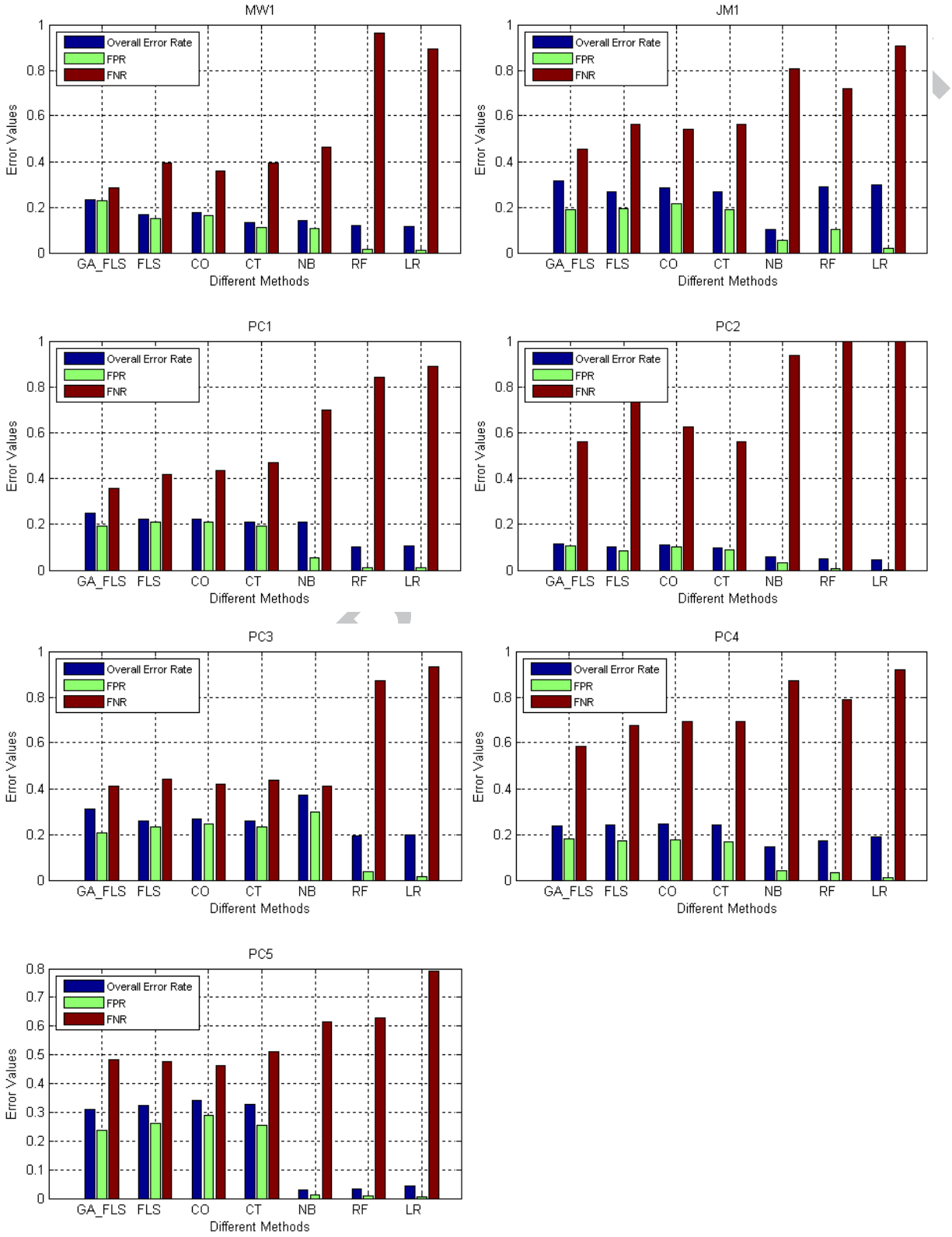


Fig. 7. Continue

Table 11: P-values of Wilcoxon test on overall error rate (GA_FLS vs. the other predictors) in NASA set

FLS	CO	CT	NB	RF	LR
0. 04	0.04	0.04	0.02	0.02	0.02

.

Based on the results reported in Fig. 7, the FNR value for GA_FLS is better than FLS and the other existing methods except for KC3, and PC5. For all the datasets, GA_FLS and FLS are comparable with RF, NB, and LR. FPR and the overall error rate values are not very different from FLS and GA_FLS. Although the FPR values in NB, RF, and LR are very low in all the datasets, the FNR values are very high, which makes the model unstable and unreliable. GA_FLS may not outperform the other methods in terms of overall error rate and FPR, but it gives balanced results with convincing performance while decreasing the FNR rate dramatically. Erroneous predictions of modules, which are misclassified, as non-faulty (FNR) is worse than incorrectly predict them as faulty. This is because a higher FNR value denotes that many fault-prone modules cannot be identified [16], which is critical. As shown in Fig. 7, the very low FPR value using NB, RF, and LR has a positive impact on the overall error rate as well; however, these results are not balanced and cannot be generalized.

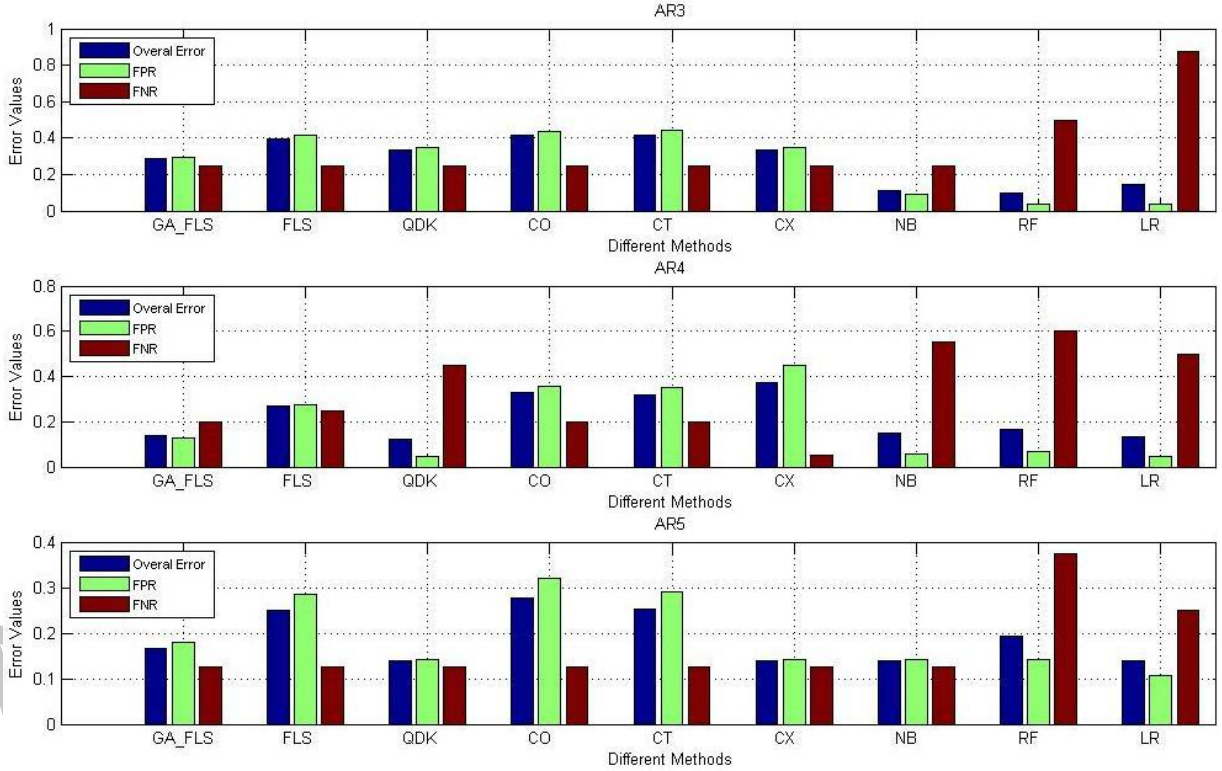


Fig. 8. Comparison results based on the Turkish set

Table 12: P-values of Wilcoxon test on overall error rate (GA_FLS vs. the other predictors) in Turkish set

FLS	QDK	CO	CT	CX	NB	RF	LR
0. 04	0.04	0.04	0.04	0.04	0.02	0.02	0.02

.

According to the results presented in Fig. 8, the overall error rate and FPR values for GA_FLS are better than manually designed FLS, QDK, CO, CT, and CX for AR3, aside from NB, RF, and LR. The overall error rate and FPR values using GA_FLS are also better than those for all of other considered approaches for AR4 and AR5, except for QDK, NB, RF, and LR. The performance of the GA_FLS using the FNR value for AR3 and AR5 are almost similar for all models; for AR4, it is better than manually designed FLS, QDK, NB, and RF and is the same as CO and CT. In AR4, the FNR value for GA_FLS is less than the values of all models except QDK. According to Fig. 8, the overall error rate using NB, RF, and LR are extremely small, especially in AR3 and AR4, which is due to the positive impact of a very small FPR. However, as explained earlier, the model performance cannot be solely decided based on one parameter and should be analysed based on the number of wrongly identified fault-prone modules and non-faulty modules.

For more comparison, the results of the proposed GA_FLS model is also compared with other papers which used NASA sets with F-measure as an evaluation metric namely, the (NSGLP)approach [33], iterative semi-supervised approach-fitting the fits (FTF) [36], and random committee with under-sampling (ROCUS) [35]. As can been seen in Table 13, GA_FLS outperformed almost all other methods except for KC1, PC1, and PC4 which NSGLP performed better.

Table 13: Comparison between GA_FLS and other method's F-measure rate on NASA set

Testing dataset	GA_FLS	FLS	FTF	ROCUS	NSGLP
CM1	**0.4102**	0.3930	0.30	0.33	0.37
KC1	0.4295	0.4086	0.37	0.41	**0.44**
KC3	**0.5384**	0.3218	0.33	0.36	0.40
MW1	**0.4054**	0.2666	0.22	0.23	0.34
JM1	**0.4487**	0.4181	0.38	0.44	0.43
PC1	0.2959	0.3255	0.23	0.24	**0.35**
PC3	**0.4117**	0.3703	0.25	0.26	0.34
PC4	0.2800	0.2031	0.36	0.39	**0.49**
PC5	**0.6575**	0.5068	0.45	0.46	0.59

5 Threats to Validity

In all research-based studies, researchers may encounter threats to validity. In this paper, three types of threats to validity, which are threats to internal validity, threats to evaluating the validity, and threats to external validity, are considered. Each is explained in the following sections.

5.1 Threats to Internal Validity

Usually in fault prediction studies, public datasets are used for the purpose of generalization. In such cases, researchers do not know much about the process of preparing these datasets as well the tools that are used. This problem cause the problem of internal validity could be a threat to internal validity. In addition, using only six software measurement metrics can be another threat as the prediction performance may be improved if more measurement metrics are employed. Furthermore, one could argue that more advanced and current software metrics

should be used such as object oriented and agile software metrics to make the prediction model more applicable in real development practice.

5.2 Threats to Evaluate Validity

The evaluation process is very important for identifying whether any model has performed well or not. Threats to evaluating validity have two sections: specifying the proportion of test data to train data in the dataset and determining the performance evaluation metrics.

In specifying the proportion of test data to train data, a very optimistic idea is to consider a part or an entire set of training data as testing set. In such case, the results do not reflect the real performance of the model. However; in this paper we followed two approaches, first, we used 10-fold cross validation technique to avoid any bias in the results and conclusions, and second, we applied the proposed model on number of new similar datasets to minimize the mentioned problem.

For determining the performance evaluation metrics, we should note that, these measurement metrics must be carefully selected according to what the study wants to measure otherwise the direction of the study goes wrong. Although there are other performance measurement metrics, selection of these metrics makes the results comparable with other prior studies. In fact, we used the FPR, FNR, and overall error rate to compare our results with those reported in the literature. However, the results could be analysed differently if these evaluation metrics are changed.

5.3 Threats to External Validity

The most important external threat in experiment-based studies is how the proposed estimation models can be generalized in the real world. Although two sets of industrial datasets from NASA and Turkish electrical goods software including 14 datasets containing different population size and defect rates are used for the purpose of generalization, however, our proposed models should be applied and need to be tested with other industrial datasets as well. In addition, the datasets of other programming languages, such as Java, can better generalize the obtained results.

5.4 Threats to Methods Selection Validity

In the current paper, a genetic algorithm was used as GA because it has been widely utilized in prior studies and it has presented convincing results. One may claim that GA is a slow algorithm; however, time is not an issue in our problem and this selection does not imply inefficiency of other methods. Since GA has presented an acceptable performance in many disciplines, we employed it in our proposed model and leave other optimization algorithms like particle swarm optimization, bee colony, ant colony, and many more for further investigation as future works. Furthermore, it should be mentioned that methods such as time series are not applicable since the observations are not collected over time and data values are not obtained at successive times or with equal intervals between them [65]. NASA and Turkish datasets that were used in this research are not collected as such.

5.5 *Threats to Construct Validity*

The predictors are sensitive to adjustments, alternatives and configurations. This issue must be considered in the construction process to avoid inefficient prediction models. The fitness function, parent and survival selection, the type of crossover and mutation operator, crossover and mutation probability, population size, and number of iterations are choices that may affect the performance of the proposed model. Actually, an exhaustive trial and error process was conducted to determine the initial parameters and different combination of mentioned parameters were verified to reach the best structure and to find the most efficient model; however, we may not claim that our setting adjustments are the best combination.

6 Conclusion and Future Work

Software fault prediction approaches can assist software developers in quality practices and software testing especially when the delivery of the project is delayed. However, building fault prediction models is very difficult when there is no information about the label of modules whether they are faulty or not and when there is no historical data available (cross-project prediction). Building these types of systems can be done using metrics thresholds, but the results are changed when varying the thresholds intervals and rules conditions. According to the analysis, fuzziness in threshold values intervals and rules are found. Thus, an expert system based on fuzzy logic is developed to model this fuzziness. Furthermore, GA is used to tune the rule-base and data-base of the expert system. GA_FLS is able to predict module fault proneness by identifying the appropriate threshold value intervals and selection of optimal rules. In this study, the threshold values are only used as initial parameters for expert systems.

The validation of two models has been performed using 14 industrial datasets. The empirical results have confirmed the efficiency of the proposed GA_FLS method over the manual FLS and other methods considering FNR values in all test cases. The performances of the manual FLS and GA_FLS method are found comparable to other proposed methods, such as random forest, naive Bayes, and logistic regression in most test cases.

Predicting a lower value in FNR means that most fault-prone modules can be detected prior to the system testing, which is useful for testers and project managers who wish to allocate their time and resources for these parts when there is a limited period and budget for testing a software project.

In the future, we plan to investigate the performance of the prediction models based on other metaheuristic algorithms rather than genetic algorithm such as particle swarm optimization and bee colonies. Furthermore, a deep analysis of other software measurement metrics, such as object-oriented and agile software metrics, will be performed with the hope of finding proper threshold values as well as accurate rules that can be employed as an initial parameter in the fuzzy expert system. In addition, use of alternatives linguistic approaches to be introduced in expert systems as type 2 Fuzzy sets, ordinal linguistic approach, multi-granular linguistic approach will also be investigated.

Acknowledgements
The authors wish to thank Universiti Teknologi Malaysia (UTM) under Research University Grant Vot-20H04 supported under Ministry of Education Malaysia for the completion of the research.

Appendices:

Appendix A
Manually constructed rule-base of the fuzzy logic system,
"L" Denotes "Low", "H" Denotes "High"

Rule	LOC	CC	UOPR	UOPND	TOPR	TOPND	Fault	Rule	LOC	CC	UOPR	UOPND	TOPR	TOPND	Fault
1	L	L	L	L	L	L	0	**33**	L	H	H	H	L	L	1
2	H	L	L	L	L	L	0	**34**	L	H	H	L	H	L	1
3	L	H	L	L	L	L	0	**35**	L	H	H	L	L	H	1
4	L	L	H	L	L	L	0	**36**	L	H	L	H	H	L	1
5	L	L	L	H	L	L	0	**37**	L	H	L	H	L	H	1
6	L	L	L	L	H	L	0	**38**	L	H	L	L	H	H	1
7	L	L	L	L	L	H	0	**39**	L	L	H	H	H	L	1
8	H	H	L	L	L	L	0	**40**	L	L	H	H	L	H	1
9	H	L	H	L	L	L	0	**41**	L	L	H	L	H	H	1
10	H	L	L	H	L	L	0	**42**	L	L	L	H	H	H	1
11	H	L	L	L	H	L	0	**43**	H	H	H	H	L	L	1
12	H	L	L	L	L	H	0	**44**	H	H	H	L	H	L	1
13	L	H	H	L	L	L	0	**45**	H	H	H	L	L	H	1
14	L	H	L	H	L	L	0	**46**	H	L	H	H	H	L	1
15	L	H	L	L	H	L	0	**47**	H	L	H	H	L	H	1
16	L	H	L	L	L	H	0	**48**	H	H	L	H	H	L	1
17	L	L	H	H	L	L	0	**49**	H	H	L	H	L	H	1
18	L	L	H	L	H	L	0	**50**	H	H	L	L	H	H	1
19	L	L	H	L	L	H	0	**51**	H	L	H	L	H	H	1
20	L	L	L	H	H	L	0	**52**	H	L	L	H	H	H	1
21	L	L	L	H	L	H	0	**53**	L	H	H	H	H	L	1
22	L	L	L	L	H	H	0	**54**	L	H	H	H	L	H	1
23	H	H	H	L	L	L	1	**55**	L	H	H	L	H	H	1
24	H	H	L	H	L	L	1	**56**	L	H	L	H	H	H	1
25	H	H	L	L	H	L	1	**57**	L	L	H	H	H	H	1
26	H	H	L	L	L	H	1	**58**	H	H	H	H	H	L	1
27	H	L	H	H	L	L	1	**59**	H	H	H	H	L	H	1
28	H	L	H	L	H	L	1	**60**	H	H	H	L	H	H	1
29	H	L	H	L	L	H	1	**61**	H	H	L	H	H	H	1
30	H	L	L	H	H	L	1	**62**	H	L	H	H	H	H	1
31	H	L	L	H	L	H	1	**63**	L	H	H	H	H	H	1
32	H	L	L	L	H	H	1	**64**	H	H	H	H	H	H	1

References

1. Seliya, N. and T.M. Khoshgoftaar, *Software quality analysis of unlabeled program modules with semisupervised clustering.* IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 2007. **37**(2): p. 201-211.
2. Dowd, M., J. McDonald, and J. Schuh, *The art of software security assessment: Identifying and preventing software vulnerabilities*. 2006: Pearson Education.
3. Catal, C. and B. Diri, *Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem.* Information Sciences, 2009. **179**(8): p. 1040-1058.
4. Radjenović, D., et al., *Software fault prediction metrics: A systematic literature review.* Information and Software Technology, 2013. **55**(8): p. 1397-1418.
5. Catal, C., *Software fault prediction: A literature review and current trends.* Expert systems with applications, 2011. **38**(4): p. 4626-4636.
6. Al Dallal, J., *Accounting for data encapsulation in the measurement of object-oriented class cohesion.* Journal of Software: Evolution and Process, 2015. **27**(5): p. 373-400.
7. Al Dallal, J., *Incorporating transitive relations in low-level design-based class cohesion measurement.* Software: Practice and Experience, 2013. **43**(6): p. 685-704.
8. Al Dallal, J., *The impact of accounting for special methods in the measurement of object-oriented class cohesion on refactoring and fault prediction activities.* Journal of Systems and Software, 2012. **85**(5): p. 1042-1057.
9. Al Dallal, J., *Fault prediction and the discriminative powers of connectivity-based object-oriented class cohesion metrics.* Information and Software Technology, 2012. **54**(4): p. 396-416.
10. Al Dallal, J. and L.C. Briand, *A precise method-method interaction-based cohesion metric for object-oriented classes.* ACM Transactions on Software Engineering and Methodology (TOSEM), 2012. **21**(2): p. 8.
11. Abaei, G. and A. Selamat, *Increasing the Accuracy of Software Fault Prediction using Majority Ranking Fuzzy Clustering.* International Journal of Software Innovation (IJSI), 2014. **2**(4): p. 60-71.
12. Abaei, G. and A. Selamat. *Software fault prediction based on improved fuzzy clustering*. in *Distributed Computing and Artificial Intelligence, 11th International Conference*. 2014. Springer.
13. Bishnu, P.S. and V. Bhattacherjee, *Software fault prediction using quad tree-based k-means clustering algorithm.* IEEE Transactions on knowledge and data engineering, 2012. **24**(6): p. 1146-1150.
14. Zhong, S., T.M. Khoshgoftaar, and N. Seliya, *Analyzing software measurement data with clustering techniques.* IEEE Intelligent Systems, 2004. **19**(2): p. 20-27.
15. Catal, C., U. Sevim, and B. Diri. *Clustering and metrics thresholds based software fault prediction of unlabeled program modules*. in *Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on*. 2009. IEEE.
16. Catal, C., U. Sevim, and B. Diri, *Metrics-driven software quality prediction without prior fault data*, in *Electronic Engineering and Computing Technology*. 2010, Springer. p. 189-199.
17. Abaei, G., Z. Rezaei, and A. Selamat. *Fault prediction by utilizing self-organizing Map and Threshold*. in *Control System, Computing and Engineering (ICCSCE), 2013 IEEE International Conference on*. 2013. IEEE.
18. ISM *Integrated Software Metrics, Inc. (ISM), in, INNOVA Commercialization Group*. 2013.
19. Rahman, F., D. Posnett, and P. Devanbu. *Recalling the imprecision of cross-project defect prediction*. in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. 2012. ACM.
20. Zimmermann, T., et al. *Cross-project defect prediction: a large scale experiment on data vs. domain vs. process*. in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. 2009. ACM.

21. Karr, C.L. *Design of an adaptive fuzzy logic controller using genetic algorithm*. in *Proc. of the 4th Int. Conf. on Genetic Algorithm*. 1991.
22. Ng, K.C. and Y. Li. *Design of sophisticated fuzzy logic controllers using genetic algorithms*. in *Fuzzy Systems, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the Third IEEE Conference on*. 1994. IEEE.
23. Rao, A.V.S. and D.K. Pratihar, *Fuzzy logic-based expert system to predict the results of finite element analysis.* Knowledge-Based Systems, 2007. **20**(1): p. 37-50.
24. Vundavilli, P.R., et al., *Fuzzy logic-based expert system for prediction of depth of cut in abrasive water jet machining process.* Knowledge-Based Systems, 2012. **27**: p. 456-464.
25. Thwin, M.M.T. and T.-S. Quah, *Application of neural networks for software quality prediction using object-oriented metrics.* Journal of systems and software, 2005. **76**(2): p. 147-156.
26. Menzies, T., J. Greenwald, and A. Frank, *Data mining static code attributes to learn defect predictors.* IEEE transactions on software engineering, 2007. **33**(1): p. 2-13.
27. Mauša, G., T.G. Grbac, and B.D. Bašić. *Multivariate logistic regression prediction of fault-proneness in software modules*. in *MIPRO, 2012 Proceedings of the 35th International Convention*. 2012. IEEE.
28. El Emam, K., et al., *Comparing case-based reasoning classifiers for predicting high risk software components.* Journal of Systems and Software, 2001. **55**(3): p. 301-320.
29. Catal, C. and B. Diri. *Software fault prediction with object-oriented metrics based artificial immune recognition system*. in *International Conference on Product Focused Software Process Improvement*. 2007. Springer.
30. Catal, C. and B. Diri. *Software defect prediction using artificial immune recognition system*. in *Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering*. 2007. ACTA Press.
31. Catal, C. and B. Diri. *A fault prediction model with limited fault data to improve test process*. in *International Conference on Product Focused Software Process Improvement*. 2008. Springer.
32. Zhong, S., T.M. Khoshgoftaar, and N. Seliya. *Unsupervised Learning for Expert-Based Software Quality Estimation*. in *HASE*. 2004. Citeseer.
33. Zhang, Z.-W., X.-Y. Jing, and T.-J. Wang, *Label propagation based semi-supervised learning for software defect prediction.* Automated Software Engineering, 2017. **24**(1): p. 47-69.
34. Li, Z., et al., *Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction.* Automated Software Engineering, 2017: p. 1-45.
35. Jiang, Y., M. Li, and Z.-H. Zhou, *Software Defect Detection with R ocus.* Journal of Computer Science and Technology, 2011. **26**(2): p. 328-342.
36. Lu, H., B. Cukic, and M. Culp. *An iterative semi-supervised approach to software fault prediction*. in *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*. 2011. ACM.
37. Li, M., et al., *Sample-based software defect prediction with active and semi-supervised learning.* Automated Software Engineering, 2012. **19**(2): p. 201-230.
38. Lu, H., E. Kocaguneli, and B. Cukic. *Defect prediction between software versions with active learning and dimensionality reduction*. in *Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on*. 2014. IEEE.
39. Erturk, E. and E.A. Sezer, *Software fault prediction using Mamdani type fuzzy inference system.* International Journal of Data Analysis Techniques and Strategies, 2016. **8**(1): p. 14-28.
40. Vir, R. and P. Mann, *A hybrid approach for the prediction of fault proneness in object oriented design using fuzzy logic.* journal of academia and industrial research, 2013: p. 661-666.
41. Yadav, H.B. and D.K. Yadav, *A fuzzy logic based approach for phase-wise software defects prediction using software metrics.* Information and Software Technology, 2015. **63**: p. 44-57.
42. Zadeh, L.A., *Fuzzy sets*, in *Fuzzy Sets, Fuzzy Logic, And Fuzzy Systems: Selected Papers by Lotfi A Zadeh*. 1996, World Scientific. p. 394-432.
43. Adams, E.S. and D.A. Farber, *Beyond the formalism debate: expert reasoning, fuzzy logic, and complex statutes.* Vand. L. Rev., 1999. **52**: p. 1241.
44. Gaines, B.R., *Foundations of fuzzy reasoning.* International Journal of Man-Machine Studies, 1976. **8**(6): p. 623-668.

45. Nozaki, K., H. Ishibuchi, and H. Tanaka, *A simple but powerful heuristic method for generating fuzzy rules from numerical data.* Fuzzy sets and systems, 1997. **86**(3): p. 251-270.
46. Ramík, J., *Soft computing: overview and recent developments in fuzzy optimization.* Ostravská univerzita, Listopad, 2001: p. 33-42.
47. Liu, Z.-Q. and S. Miyamoto, *Soft computing and human-centered machines*. 2012: Springer Science & Business Media.
48. Medasani, S., J. Kim, and R. Krishnapuram, *An overview of membership function generation techniques for pattern recognition.* International Journal of approximate reasoning, 1998. **19**(3-4): p. 391-417.
49. Soyer, A., Ö. Kabak, and U. Asan, *A fuzzy approach to value and culture assessment and an application.* International Journal of Approximate Reasoning, 2007. **44**(2): p. 182-196.
50. Bilgiç, T. and I.B. Türkşen, *Measurement of membership functions: theoretical and empirical work*, in *Fundamentals of fuzzy sets*. 2000, Springer. p. 195-227.
51. Tzvieli, A., *Possibility theory: an approach to computerized processing of uncertainty*, 1990, Wiley Online Library.
52. Dubois, D. and H. Prade, *The three semantics of fuzzy sets.* Fuzzy sets and systems, 1997. **90**(2): p. 141-150.
53. Wilamowski, B., *Neural networks and fuzzy systems.* The Microelectronic Handbook, 2002.
54. Klir, G.J. and B. Yuan, *Fuzzy sets and fuzzy logic: theory and applications.* Possibility Theory versus Probab. Theory, 1996. **32**(2).
55. Mamdani, E. and S. Assilian, *An experiment in linguistic synthesis with a fuzzy logic controller.* International journal of human-computer studies, 1999. **51**(2): p. 135-147.
56. Mamdani, E.H. and S. Assilian, *An experiment in linguistic synthesis with a fuzzy logic controller.* International journal of man-machine studies, 1975. **7**(1): p. 1-13.
57. Takagi, T. and M. Sugeno, *Fuzzy identification of systems and its applications to modeling and control*, in *Readings in Fuzzy Sets for Intelligent Systems*. 1993, Elsevier. p. 387-403.
58. Holland, J.H., *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. 1992: MIT press.
59. Kumar, K.S., R.B. Misra, and N.K. Goyal, *Development of fuzzy software operational profile.* International Journal of Reliability, Quality and Safety Engineering, 2008. **15**(06): p. 581-597.
60. McCabe, T.J., *A complexity measure.* IEEE Transactions on software Engineering, 1976(4): p. 308-320.
61. PROMISE *Promise Software Engineering Repository*. 2012.
62. Matthews, B.W., *Comparison of the predicted and observed secondary structure of T4 phage lysozyme.* Biochimica et Biophysica Acta (BBA)-Protein Structure, 1975. **405**(2): p. 442-451.
63. Hall, T., et al., *A systematic literature review on fault prediction performance in software engineering.* IEEE Transactions on Software Engineering, 2012. **38**(6): p. 1276-1304.
64. Abaei, G. and A. Selamat, *A survey on software fault detection based on different prediction approaches.* Vietnam Journal of Computer Science, 2014. **1**(2): p. 79-95.
65. Brockwell, P.J. and R.A. Davis, *Introduction to time series and forecasting*. 2016: springer.

Graphical Abstract

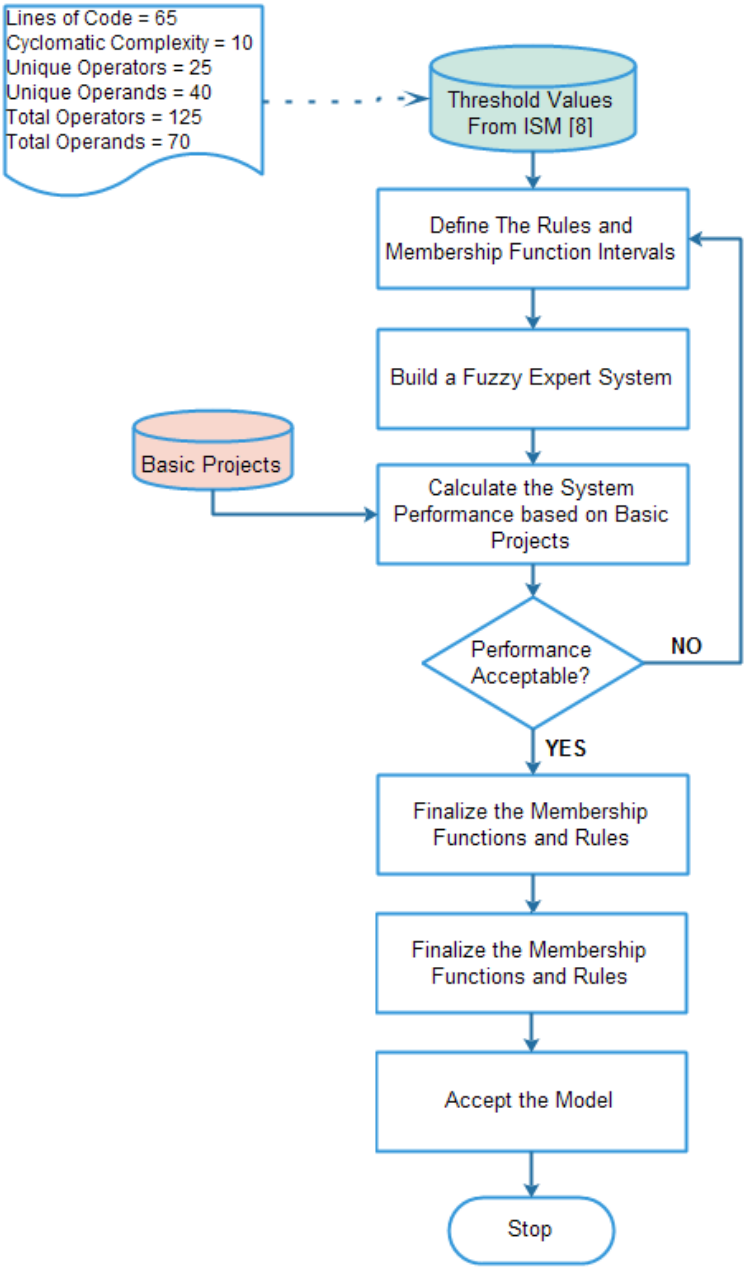


Fig. 1. Process steps in Method 1 (Fuzzy logic system)