Software Fault Prediction of Unlabeled Program Modules

C. Catal, U. Sevim, and B. Diri, *Member, IAENG*

***Abstract*—Software metrics and fault data belonging to a previous software version are used to build the software fault prediction model for the next release of the software. Until now, different classification algorithms have been used to build this kind of models. However, there are cases when previous fault data are not present; and hence, supervised learning approaches cannot be applied. In this study, we propose a fully automated technique which does not require an expert during the prediction process. In addition, it is not required to identify the number of clusters before the clustering phase, as required by K-means clustering method. Software metrics thresholds are used to remove the expert necessity.**

Our technique first applies X-means clustering method to cluster modules and identifies the best cluster number. After this step, the mean vector of each cluster is checked against the metrics thresholds vector. A cluster is predicted as fault-prone if at least one metric of the mean vector is higher than the threshold value of that metric. In addition to X-means clustering-based method, we made experiments with pure metrics thresholds method, fuzzy clustering, and K-means clustering-based methods. Experiments reveal that unsupervised software fault prediction can be fully automated and effective results can be produced using X-means clustering with software metrics thresholds. Three datasets, collected from Turkish white-goods manufacturer developing embedded controller software, have been used for the validation.

***Index Terms*— Clustering, metrics thresholds, software fault prediction, and X-means clustering.**

I. Introduction

The quality of software components should be tracked continuously during the development of high-assurance systems such as telecommunication infrastructures, medical devices, and avionic systems. Quality assurance group can improve the product quality by allocating necessary budget and human resources to low quality modules identified with different quality estimation models. Recent advances in software quality estimation yield building defect predictors with a mean probability of detection of 71 percent and mean false alarms rates of 25 percent [1]. Software quality estimation is not only interested in reliability, but also the other quality characteristics such as usability, efficiency, maintainability, functionality, and portability. However, some researchers prefer using the term *software quality estimation* for the software fault prediction modeling studies [2]. Software metrics are used as independent variables and fault data are regarded as dependent variable in software fault prediction models.

The aim of building this kind of models is to predict the fault labels (fault-prone or not fault-prone) of the modules for the next release of the software. Some benefits of using fault prediction models are [3]:

- The identification of refactoring candidate modules (fault-prone modules),
- The selection of the best design approach from design alternatives,
- The improvement of software testing process and software quality,
- Reaching a highly dependable system.

A typical software fault prediction process includes two steps, as shown in Figure 1. First, a fault prediction model is built using previous software metrics and fault data belonging to each software module (class or method level). After this training phase, fault labels of program modules can be estimated using this model [4].

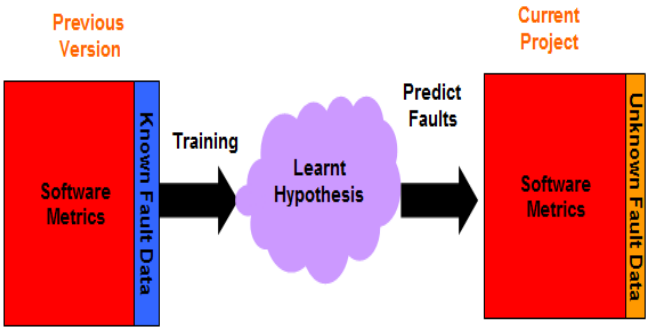

Fig. 1. This shows software fault prediction process [4].

The selection of metrics type is dependent on the programming paradigm used in the project and research targets. Our systematic review, focusing on 74 papers published between year 1990 and 2007, revealed that 60 percent of papers used method-level metrics [5]. Therefore, we applied method-level metrics to build our models in this study. A sample training dataset, including software metrics and known fault data, is shown in Figure 2. All the metrics are separated with commas in this figure and the last column

C. Catal, PhD is with The Scientific and Technological Research Council of TURKEY, Marmara Research Center, Information Technologies Institute, Gebze, Kocaeli, 41470, TURKEY (phone: +90 262 677 26 34; fax: +90 262 646 31 87; e-mail: cagatay.catal@bte.mam.gov.tr).

U. Sevim is with with the Department of Computer Engineering, Bogazici University, Bebek, Istanbul, 34342 TURKEY (e-mail: ugur.sevim@boun.edu.tr).

B. Diri, Ass. Prof. Dr. is with the Department of Computer Engineering, Yildiz Technical University, Yildiz, Istanbul, 34349 TURKEY (e-mail: banu@ce.yildiz.edu.tr).

This project is supported by The Scientific and Technological Research Council of TURKEY (TUBITAK) under Grant 107E213. The findings and opinions in this study belong solely to the authors, and are not necessarily those of the sponsor.

presents whether this module caused fault or not during the testing phase. This last feature (column) consists of *false* and *true* values.

```
15,2,1,2,42,168,0.11,9,18.67,1512,0.06,84,12,0,1,0,9,7,false
25,2,1,2,34,132.83,0.12,8.57,15.5,1138.58,0.04,63.25,13,6,4,0,8,7,true
15,2,1,2,43,191.76,0.15,6.67,28.76,1278.37,0.06,71.02,8,2,2,0,10,12,false
9,2,1,2,20,74.01,0.13,7.88,9.4,582.82,0.02,32.38,4,2,1,0,9,4,true
16,2,1,2,44,196.21,0.15,6.67,29.43,1308.1,0.07,72.67,9,2,2,0,10,12,false
```

Fig. 2. This shows a sample fault prediction dataset.

From machine learning perspective, Figure 1 is a supervised learning approach because the modeling phase uses class labels represented as *known fault data* in the figure. Most of the software fault prediction studies focused on developing fault predictors using previous fault data.

However, there are cases when previous fault data are not available. For example, a software company might start to work on a new project domain or might plan building fault predictors for the first time in their development cycle. In addition, current software version's fault data might not be collected and therefore, there might not exist in any previous fault data for the next release of the software. In these cases, supervised learning approaches can not be developed because of the absence of class labels. Figure 3 depicts this challenging problem and unsupervised learning approaches can be applied in these cases.

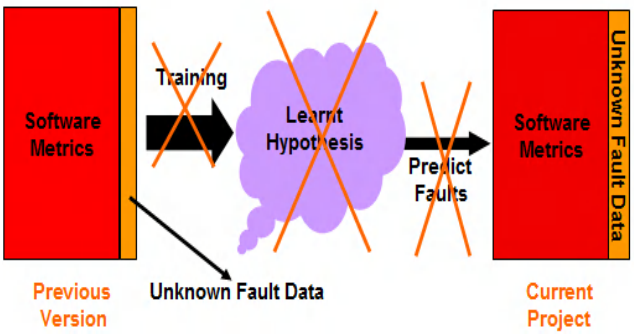

Fig. 3. This shows no fault data problem [4].

There are a few studies that have tried to build a fault prediction model when the fault labels for modules are unavailable. Zhong et al. [6] used K-means and Neural-Gas clustering methods to cluster modules, and then an expert who is 15 years experienced engineer, labeled each cluster as fault-prone or not fault-prone by examining not only the representative of each cluster, but also some statistical data such as global mean, minimum, maximum, median, 75 percentile, and 90 percentile of each metric.

To remove the obligation of an expert assistance, we developed a prediction model in our previous work [7] and validated it on three datasets, collected from Turkish white-goods manufacturer developing embedded controller software. Metrics thresholds were used to embed the expert knowledge into our model and subjective human interaction was eliminated [7]. First, K-means clustering method is applied and the mean vector of each cluster is checked against the metrics thresholds vector. A cluster is predicted as fault-prone if at least one metric of the mean vector is higher than the threshold value of that metric [7]. The main contribution of that study is the usage of metrics thresholds with or without clustering methods and the removing the obligation of an expert assistance.

However, there is one drawback of our previous study [7]. Because K-means clustering method is used in the first stage, K number should be selected heuristically and this number may affect the overall performance of the model. Therefore, we aimed to build a fully automated fault prediction model which can be applied when there is no previous fault data. In this new study, we propose a new fault prediction model which does not require the selection of K number heuristically. Instead, K number is automatically calculated with X-means clustering algorithm. After the identification of K number and the clusters, metrics thresholds are again used as done in our previous study.

The main contribution of this paper is the development of an automated way of assigning fault-proneness labels to the modules and the removing the subjective expert opinion. Subjective human interaction directly affects the quality of the software fault prediction model and adds unnecessary complexity.

We explored our new approach on three datasets, collected from Turkish white-goods manufacturer developing embedded controller software for washing machines, dish washers, and refrigerators. These datasets, AR3, AR4, and AR5 are available at http://promisedata.org . They include 29 metrics, but we used only 6 metrics during modeling because we know only their industrial thresholds. Even though we validated our approach on datasets collected from Turkish company, neither our model nor the metrics thresholds are dependent on this company. In this analysis, a module is a method because procedural programming was used. The results of this study show that the application of X-means clustering method with metrics thresholds provides better performance compared to pure thresholds and fuzzy clustering-based approaches.

6 method-level metrics including the primitive Halstead and McCabe metrics were used for the development of this model. The metrics used in our experiments are lines of code, cyclomatic complexity, unique operator, unique operand, total operand, and total operator. Threshold vector [LoC, CC, UOp, UOpnd, TOp, and TOpnd] was chosen as [65, 10, 25, 40, 125, and 70]. We started the analysis with the metrics thresholds proposed by Integrated Software Metrics, Inc. (ISM). Later, values were calibrated according to our experiments in order to achieve high-performance prediction models. We used same thresholds values as in our previous study [7].

The rest of the paper is organized as follows. Section 2 presents related work and Section 3 introduces clustering methods. Section 4 presents an empirical case study using real-world data from embedded controller software developed in Turkey. Section 5 explains conclusion and future works.

II. RELATED WORK

There are a few software fault prediction studies which do not use prior fault data for modeling. Zhong et al. [6] used K-means and Neural-Gas algorithms to cluster modules and

an expert explored several statistical data within each cluster to label each cluster as fault-prone or not fault-prone. However, this approach is dependent on the capability of the expert who should be specialized in machine learning and software engineering areas. Furthermore, the selection of the cluster number, K, is done heuristically when k-means clustering method is chosen and this process can affect the model's performance drastically. Seliya et al. [8] proposed a constraint-based semi-supervised clustering scheme that uses K-means clustering method as the underlying clustering algorithm for this problem. They showed that this approach works better than their previous unsupervised learning based prediction approach. However, the selection of the cluster number is still a critical issue in this model and their approach uses an expert's domain knowledge to iteratively label clusters as fault-prone or not. Therefore, this model is also dependent on the capability of the expert. Catal et al. [7] proposed a clustering and metrics thresholds based software fault prediction approach and explored it on three datasets. The main contribution of their paper is the usage of metrics thresholds with or without clustering methods and the removing of the obligation of an expert assistance. However, the selection of the cluster number is done heuristically in this clustering based model too. In this study, we use x-means clustering method and our model does not require the selection of cluster number. Instead of an exact cluster number, an interval is provided to the x-means algorithm.

III. CLUSTERING METHODS

A. Clustering

Clustering is an unsupervised learning approach. It locates in *indirect data mining* group and classification area locates in d*irect data mining group*. While classification uses class labels for training, clustering does not use class labels and tries to discover relationships between the features [9]. Clustering methods can be used to group the modules having similar metrics by using similarity measures or distances. After the clustering phase, the mean values of each metric within cluster can be checked against industrial metrics thresholds. If the limits are exceeded, the cluster can be labeled as fault-prone. Cluster analysis has four basic steps [10]:

- *Feature Selection:* We used 6 method-level metrics including the primitive Halstead and McCabe metrics because we know the thresholds of these metrics.
- *Clustering Algorithm Selection:* X-means clustering algorithm was selected because it does not require the selection of cluster number, K, prior to execution of the algorithms.
- *Cluster Validation:* Any clustering algorithm can generate several clusters, but they may not reflect the existence of the patterns locating in the dataset. Therefore, evaluation parameters are required to judge the effectiveness of the algorithm. After the clustering phase, the mean vector of each cluster is checked against the metrics thresholds vector. Evaluation parameters are used after this phase and they evaluate the overall performance of our approach. False positive rate (fpr), false negative rate (fnr), and the error values were calculated by using confusion matrix during our experiments.
- *Results Interpretation:* We compared our model's performance with pure thresholds based approach. Because the performance of our two-phase model improves, we suggest this approach. The overall interpretation was realized with this comparison.

The classification of clustering algorithms is not easy, but a categorization was created by Berkhin [11] and we provide this list as follows:

- Hierarchical Methods
  - Agglomerative Algorithms
  - Divisive Algorithms
- Partitioning Methods
  - Relocation Algorithms
  - Probabilistic Clustering
  - K-medoids Methods
  - K-means Methods
  - Density-Based Algorithms
    - Connectivity Clustering
    - Density Functions Clustering
- Grid-Based Methods
- Methods using Co-occurrence of Categorical Data
- Constraint-based Clustering
- Clustering Algorithms used in Machine Learning
  - Gradient Descent and Neural Networks
  - Evolutionary Methods
- Scalable Clustering Algorithms
- Algorithms for High Dimensional Data
  - Subspace Clustering
  - Projection Techniques
  - Co-clustering Techniques

These groups may overlap and other researchers may create different categorizations. Another categorization is shown as follows [9]:

- Fuzzy clustering
- Hard clustering
  - Partitional
    - K-means and derivatives
    - Locality-sensitive hashing
    - Graph-theoretic methods
  - Hierarchical
    - Divisive
    - Agglomerative
      - Graph methods
      - Geometric methods

X-means is under *K-means and derivatives* group.

B. Clustering Algorithms Used in Experiments

K-means: One of the simplest clustering algorithms is K-means clustering method. The pseudo code of this algorithm is given as follows [9]:

"Require: Dataset D, number of clusters k, Dimension d:

{ C_i is the *i*th cluster }

{ 1. Initialization Phase}
1: (C_1, C_2, …, C_k} = Initial partition of D.
{ 2. Iteration Phase}
2: **repeat**
3: d_{ij} = distance between case i and cluster j;
4: n_i = argmin d_{ij};
5: Assign case i to cluster n_i;
6: Recompute the cluster means of any changed clusters;
7: **until** no further changes of cluster membership occur
8: Output results" [9].

In the initialization phase, clusters are initialized with random instances and in the iteration phase, instances are assigned to clusters according to the distances, computed between the centroid of the cluster and the instance. This iteration phase goes on until no changes occur in the clusters.

X-means: One drawback of k-means algorithm is the selection of the number of clusters, k, as an input parameter. Pelleg and Moore [12] developed an algorithm to solve this problem and used the Bayesian Information Criterion (BIC) or the Akaike Information Criterion (AIC) measure for optimization [9]. Rather than choosing the specific number of clusters, k, x-means needs k_{min} and k_{max} values. The algorithm starts with k_{min} value and adds centroids if needed. The BIC or Schwarz criterion is applied to split some centroids into two and hence new centroids are created [9]. Final centroid set is the one that has the best score.

Given n objects in a dataset D = {x_1, x_2, ..., x_n} in a d-dimensional space and a set of alternative models M_j = {C_1, C_2, …, C_k}, scoring of these alternative models, identified with different k values, is done by using the posterior probabilities P(M_j | D) [9]. The Schwarz criterion is shown in Equation 1.

$$BIC(M_j) = \hat{I}_j(D) - \frac{p_j}{2}\log n \tag{1}$$

$\hat{I}_j(D)$ is the loglikelihood of the *j*th model and M_j's number of parameters are represented with p_j. The largest score reflects the true model and it is selected as the final model [9]. The maximum likelihood estimate of variance is calculated using the Equation 2 under the identical spherical Gaussian distribution and μ_i is the centroid which is closest to the object x_i [9].

$$\hat{\sigma}^2 = \frac{1}{n-k}\sum_{i=1}^{n}(x_i - \mu_i)^2 \tag{2}$$

The point probabilities are calculated using the Equation 3 [9].

$$\hat{P}(x_i) = \frac{|C_i|}{n}\frac{1}{\sqrt{2\pi}\,\hat{\sigma}^d}\exp\left(-\frac{1}{2\hat{\sigma}^2}\|x_i - \mu_i\|^2\right) \tag{3}$$

The loglikelihood of the data is calculated using the Equation 4. "The Schwarz criterion is used in X-means globally to choose the best model it encounters and locally to guide all centroid splits." [9].

$$l(D) = \prod_{i=1}^{n} P(x_i)$$

$$= \sum_{i=1}^{n}\left(\log\frac{1}{\sqrt{2\pi}\hat{\sigma}^d} - \frac{1}{2\hat{\sigma}^2}\|x_i - \mu_i\|^2 + \log\frac{|C_i|}{n}\right) \tag{4}$$

Fuzzy C-means: Fuzzy c-means clustering method was developed by Bezdek [13]. Each instance can belong to every cluster with a different membership grades between 0 and 1 for this algorithm. A dissimilarity function, shown in Equation 5, is minimized and centroids which minimize this function are identified. The general steps of this algorithm are shown as follows [14], [15]:

I. Initialize the membership function randomly according to the Equation 5.
II. Calculate centroids according to the Equation 7.
III. Calculate dissimilarity value according to the Equation 6. Stop, if the improvement compared to previous iteration is below a threshold level.
IV. Calculate a new u according to the Equation 8. Go to step 2.

$$\sum_{i=1}^{c} u_{ij} = 1, \forall j = 1,...,n \tag{5}$$

$$J(U, c_1, c_2, ..., c_c) = \sum_{i=1}^{c} J_i = \sum_{i=1}^{c}\sum_{j=1}^{n} {u_{ij}}^m {d_{ij}}^2 \tag{6}$$

$$c_i = \frac{\sum_{j=1}^{n} {u_{ij}}^m x_j}{\sum_{j=1}^{n} {u_{ij}}^m} \tag{7}$$

$$u_{ij} = \frac{1}{\sum_{k=1}^{c}\left(\frac{d_{ij}}{d_{kj}}\right)^{2/(m-1)}} \tag{8}$$

c_i is *i*th cluster's centroid, *u* is between 0 and 1, d_{ij} is the Euclidean distance between centroid and the data point, m is a weighting exponent which is between 1 and ∞ [14]. Because the first step of the algorithm uses random assignments, it may not converge to an optimal solution and the performance is dependent on the initial centroids [14]. One approach to solve this problem is using a defined procedure to identify initial centroids such as calculating the means of all data points [14]. We used Fuzzy C-means clustering implementation locating in MATLAB. However, X-means implementation was accessed from WEKA open source machine learning tool. K-means implementation exists in both MATLAB and WEKA tool, but we used MATLAB implementation because we had developed some MATLAB programs to evaluate the overall performance of these algorithms. Evaluation parameters will be introduced in the next chapter.

IV. EMPIRICAL CASE STUDY

A. Evaluation Parameters

Up to now, different evaluation parameters were used for imbalanced datasets, specifically for software quality classification problem. Some of these parameters are shown as follows:

- Area under ROC Curve (AUC) [16]

- PD (probability of detection), PF (probability of false alarm), balance [1]
- G-mean1, G-mean2, F-measure [17]
- Sensitivity, specificity, J-coefficient [18]
- Correctness, completeness [19]
- FPR (false positive rate), FNR (false negative rate), error [20]

In this study, we used FPR, FNR and error parameters to evaluate the models we developed. Error is the percentage of mislabeled modules, false positive rate (FPR) is the percentage of not faulty modules labeled as fault-prone by the model, and false negative rate (FNR) is the percentage of faulty modules labeled as not fault-prone [6]. Confusion matrix used to calculate evaluation parameters is shown in Table 1. Equations 8, 9, and 10 calculate FPR, FNR and error values respectively.

Table 1. Confusion matrix

Predicted Labels	**Actual Labels**	
	YES	NO
YES	True-Positive **(TP)**	False-Positive **(FP)**
NO	False-Negative **(FN)**	True-Negative **(TN)**

$$FPR = \frac{FP}{FP + TN} \tag{8}$$

$$FNR = \frac{FN}{FN + TP} \tag{9}$$

$$Error = \frac{FN + FP}{TP + FP + FN + TN} \tag{10}$$

All of these evaluation parameters must be minimized, but there is a trade-off between FPR and FNR values. FNR value is much more critical than FPR value because high FNR value means that a large amount of fault-prone modules can not be detected prior to the system testing or operation.

B. Results and Analysis

We used four different types of unsupervised software fault predictors and three of them are based on clustering methods. Because *k*-means clustering and fuzzy C-means clustering methods require the selection of number of clusters, we first used X-means clustering method in three datasets and calculated the *k* values for each of these datasets. As explained in previous chapter, X-means algorithm requires an interval to calculate the best *k* value. We chose the minimum *k* value as 2 and maximum *k* value as the number of data points in that dataset. X-means algorithm identified k value as 2 for AR5 and calculated k value as 3 for AR3 and AR4 datasets. Experimental results are shown in Table 2.

In order to evaluate the performance of our fully automated approach which is based on X-means clustering method, we compared it with our metrics thresholds based approach [7]. Table 2 shows that FPR values decreased for AR3 (from 43,63 to 34,55) and AR5 datasets (from 32,14 to 14,29) when X-means based approach is used. Even though FPR value increased for AR4 dataset, its FNR value decreased from 20 to 5. As explained in Evaluation Parameters section, FNR value is much more critical for our models.

While our pure metrics thresholds based approach (**Threshold**) detects fault-prone modules according to the metrics thresholds, X-means based approach first calculates the best k value, divides data points into k clusters and then the mean vector of each cluster is checked against the metrics thresholds vector. Same approach is used for fuzzy c-means and k-means based fault predictors. Therefore, our clustering based approaches have two stages and second step of them is similar to pure metrics thresholds based approach.

According to our pure metrics thresholds based approach, a module is predicted as fault-prone if at least one metric of the module is higher than the specified value of that metric. According to our clustering based approaches, a cluster is predicted as fault-prone if at least one metric of the mean vector is higher than the specified threshold value of that metric. Datasets include class labels, but we ignored this column for modeling because our purpose was to develop models for software fault prediction without priori fault data. Class labels were used to calculate the evaluation parameters.

Table 2. Experimental results on three datasets

Data	**Prm.**	**Threshold**	**X-means**	**Fuzzy c**	**K-means**
AR3	FPR	43,63	34,55	12,73	34,27
	FNR	25	25	25	25
	Error	41,27	33,33	14,29	33,09
	# cluster	N/A	3	3	3
AR5	FPR	32,14	14,29	14,29	14,28
	FNR	12,5	12,5	12,5	12,5
	Error	27,77	13,89	13,89	13,88
	# cluster	N/A	2	2	2
AR4	FPR	35	44,83	4,6	4,6
	FNR	20	5	45	45
	Error	32,71	37,38	12,15	12,14
	# cluster	N/A	3	3	3

Because fuzzy c-means and k-means clustering based approaches did not improve the performance compared to x-means based approaches when the same k values are used, we suggest using x-means based prediction model.

If previous fault data exist for projects, normally supervised learning algorithms such as Naïve Bayes and Random Forests can be applied. However, our research focus was to build fault prediction models that can be used when the fault labels for modules are unavailable. Experiments reveal that unsupervised software fault prediction can be fully automated and effective results can be produced using X-means clustering with software metrics thresholds.

C. External Validity

In order to generalize the results of an empirical study outside the experimental setting, threats to the external validity should be discussed. Datasets used in this study were collected from an industrial environment in Turkey; systems were developed by professional developer groups, and these systems are real industry projects. These features satisfy the requirements explained in Khoshgoftaar et al.'s study [21]. However, development practices and project domain of this

Turkish software company may be different than the other software companies that plan using this prediction model. Software development model of this company is not process-oriented as in NASA and projects were developed by centrally-controlled top-down management teams. Therefore, open source projects are different than these kinds of projects used here.

Another important point for our models is the effect of noisy instances. Because we calculate the mean vector of each cluster, noisy instances can change the mean vector drastically and hence the performance of our models may be affected negatively. Because projects used in these experiments are middle-sized ones and the dataset collection process is done carefully, we assume that there is no noisy instances. However, the prediction of a dataset consisting of many noisy instances may not provide acceptable results.

V. CONCLUSION AND FUTURE WORK

This study proposed an unsupervised software fault prediction approach. Experiments revealed that unsupervised software fault prediction can be fully automated and effective results can be produced by using X-means clustering with software metrics thresholds. The main contribution of this paper is the development of an automated way of assigning fault-proneness labels to the modules and the removing the subjective expert opinion. There is no heuristic step in our model as needed in k-means clustering based fault prediction approaches.

We studied on three public datasets which locate in PROMISE repository. Results are promising and our model can be used when there is no priori fault data. Our metrics thresholds vector was created by using the thresholds proposed by Integrated Software Metrics, Inc. (ISM) and these threshold values were calibrated in our previous study [7]. Even though ISM focused on NASA datasets to calculate these threshold values, we could apply similar threshold values for our datasets, collected from Turkish white-goods manufacturer developing embedded controller software. Our models are not dependent on this vector and each company can identify its threshold vector with different approaches.

Future work will consider evaluating our model for datasets which have noisy instances such as JM1 dataset in PROMISE repository. A pre-processing step is necessary to remove noisy instances before our prediction model is applied or we need to develop a new unsupervised fault prediction model which is insensitive to noisy instances.

REFERENCES

[1] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors", *IEEE Transactions on Software Engineering,* vol. 32, no.1, 2007, pp. 2-13.

[2] N. Seliya, T. M. Khoshgoftaar, "Software quality estimation with limited fault data: a semi-supervised learning perspective", *Software Quality Journal*, vol. 15, no. 3, 2007, pp. 327-344.

[3] C. Catal, B. Diri, "Investigating the effect of dataset size, metrics set, and feature selection techniques on software fault prediction problem", *Information Sciences*, vol. 179, no. 8, pp. 1040-1058, 2009.

[4] N. Seliya, "Software quality analysis with limited prior knowledge of faults", Graduate Seminar, Wayne State University, Department of Computer Science, 2006, Webpage: www.cs.wayne.edu/graduateseminars/gradsem_f06/Slides/seliya_wsu_talk.ppt

[5] C. Catal, B. Diri, "A systematic review of software fault predictions studies", *Expert Systems with Applications*, vol. 36, no.4, pp. 7346-7354, 2009.

[6] S. Zhong, T. M. Khoshgoftaar, and N. Seliya, "Unsupervised learning for expert-based software quality estimation", *Proc. of the 8th Intl. Symp. On High Assurance Systems Eng.*, Tampa, FL, 2004, pp. 149-155.

[7] C. Catal, U. Sevim, B. Diri, "Clustering and metrics thresholds based software fault prediction of unlabeled program modules", *6th Int'l. Conference on Information Technology: New Generations*, IEEE Computer Society, Las Vegas, Nevada, 2009.

[8] N. Seliya, 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*, vol. 37, no. 2, 2007, pp. 201-211.

[9] G. Gan, C. Ma, J. Wu, "Data clustering: theory, algorithms, and applications", Society for Industrial and Applied Mathematics, Philadelphia, 2007.

[10] R. Xu, D. Wunsch, "Survey of clustering algorithms", *IEEE Transactions on Neural Networks*, vol. 16, no. 3, 2005, pp. 645-678.

[11] P. Berkhin, "Survey of clustering data mining techniques", *Technical Report,* Accrue Software, San Jose, California, 2002, www.ee.ucr.edu/~barth/EE242/clustering_survey.pdf

[12] D. Pelleg, A. Moore, "X-means: extending k-means with efficient estimation of the number of clusters", *Proceedings of the 17th International Conference on Machine Learning*, pp. 727-734, 2000, Stanford University, Stanford, CA, USA.

[13] J. C. Bezdek, "Pattern recognition with fuzzy objective function algorithms", Plenum Press, New York, 1981.

[14] S. Albayrak, F. Amasyalı, "Fuzzy c-means clustering on medical diagnostic systems", *International 12. Turkish Symposium on Artificial Intelligence and Neural Networks*, Turkey, 2003.

[15] J. S. R. Jang, C. T. Sun, E. Mizutani, "Neuro-fuzzy and soft computing", Prentice Hall, pp. 426-427, 1997.

[16] J. Van Hulse, T. M. Khoshgoftaar, A. Napolitano, "Experimental perspectives on learning from imbalanced data", *24th Int'l. Conference on Machine Learning*, Corvalis, Oregon, pp. 935-942, 2007.

[17] Y. Ma, L. Guo, B. Cukic, "A statistical framework for the prediction of fault-proneness", Advances in Machine Learning Application in Software Engineering, Idea Group Inc., pp. 237-265, 2006.

[18] K. El-Emam, W. Melo, J. C. Machado, "The prediction of faulty classes using object-oriented design metrics", *Journal of Systems and Software*, vol. 56, no. 1, pp. 63-75, 2001.

[19] Y. Zhou, H. Leung, "Empirical analysis of object-oriented design metrics for predicting high and low severity faults", *IEEE Transactions on Software Eng.*, vol. 32, no. 10, pp. 771-789, 2006.

[20] S. Zhong, T. M. Khoshgoftaar, and N. Seliya, "Analyzing software measurement data with clustering techniques", *IEEE Intelligent Systems*, vol. 19, no. 2, pp. 20-27, 2004.

[21] T. M. Khoshgoftaar, N. Seliya, N. Sundaresh, "An empirical study of predicting software faults with case-based reasoning", *Software Quality Journal*, vol. 14, no. 2, pp. 85-111, 2006.