A K-Means Based Clustering Approach for Finding Faulty Modules in Open Source Software Systems

Parvinder S. Sandhu, Jagdeep Singh, Vikas Gupta, Mandeep Kaur, Sonia Manhas, Ramandeep Sidhu

***Abstract*—**Prediction of fault-prone modules provides one way to support software quality engineering. Clustering is used to determine the intrinsic grouping in a set of unlabeled data. Among various clustering techniques available in literature K-Means clustering approach is most widely being used. This paper introduces K-Means based Clustering approach for software finding the fault proneness of the Object-Oriented systems. The contribution of this paper is that it has used Metric values of JEdit open source software for generation of the rules for the categorization of software modules in the categories of Faulty and non faulty modules and thereafter empirically validation is performed. The results are measured in terms of accuracy of prediction, probability of Detection and Probability of False Alarms.

***Keywords*—**K-Means, Software Fault, Classification, Object Oriented Metrics.

I. Introduction

FAULTS in software systems continue to be a major problem. Many systems are delivered to users with excessive faults. This is despite a huge amount of development effort going into fault reduction in terms of quality control and testing. It has long been recognized that seeking out fault-prone parts of the system and targeting those parts for increased quality control and testing is an effective approach to fault reduction. Fault-proneness of a software module is the probability that the module contains faults. A correlation exists between the fault-proneness of the software and the measurable attributes of the code (i.e. the static metrics) and of the testing (i.e. the dynamic metrics). Prediction of fault-prone modules provides one way to support software quality engineering through improved scheduling and project control. Quality of software is increasingly important and testing related issues are becoming crucial for software. Methodologies and techniques for predicting the testing effort, monitoring process costs, and measuring results can help in increasing efficiency of software testing. Being able to measure the fault-proneness of software can be a key step towards steering the software testing and improving the effectiveness of the whole process. In the past, several metrics for measuring software complexity and testing thoroughness have been proposed. Static metrics, e.g., the McCabe's cyclomatic number or the Halstead's Software Science, statically computed on the source code and tried to quantify software complexity. Despite this it is difficult to identify a reliable approach to identifying fault-prone software components.

Clustering is used to determine the intrinsic grouping in a set of unlabeled data. It is the process of organizing objects into groups whose members are similar in some way. Among various clustering techniques available in literature K-Means clustering approach is most widely being used. K-Means is an unsupervised clustering technique used to classify data in to K clusters. It is partitional clustering approach, each cluster is associated with a centroid (center point), each point is assigned to the cluster with the closest centroid, Number of clusters, K, must be specified.Hence, in this study, a K-Means Based Clustering Approach is used for finding faulty Modules in Open Source Software Systems. In order to perform the analysis we validate the performance of the K-Means based clustering method for dataset derived from open source software JEdit [1]. We investigate the accuracy of the fault proneness predictions using object oriented design using metrics suite given by Chidamber and Kemerer [2] and used in [3] for fault prediction. In the literature [3]-[17] various types of Fault-Proneness Estimation Models are discussed.

The paper is organized as follows: section II explains about the methodology followed and section III the result of the study. Finally conclusions of the research are presented in section IV.

II. Methodology Followed

The following are the steps used for the predicton of fault prone modules:

First of all, find the structural code and design attributes of software systems. Thereafter, select the suitable metric values as representation of statement. Next step is to analyze, refine metrics and normalize the metric values. We used JEdit open source software in this study [18]. JEdit is a programmer's text editor developed using Java language. JEdit combines the functionality of Window, Unix, and MacOS text editors. It

Dr. Parvinder S. Sandhu is working as Professor with the Rayat & Bahra Institute Of Engineering & Bio-Technology, Mohali-Sahauran14004. E-Mail: parvinder.sandhu@gmail.com.

Vikas Gupta is working as Asstt. Prof. with the Rayat Institute Of Engineering & Information Technology, Rail Majra, Ropar, Punjab, India.

Jagdeep Singh & Ramandeep Singh Sidhu are doing M.Tech. CSE from RIEIT, Rail Majra, Punjab.

Mandeep Kaur & Sonia Manhas are working with SSCET, Badhani, Punjab, India.

was released as free software and the source code is available on [19]. JEdit includes 274 classes. The number of developers involved in this project was 144. The project was started in 1999. The number of bugs was computed using SVC repositories. The release point for the project was identified in 2002. The log data from that point to 2007 was collected. The header files in C++ were excluded in data collection. The word bug or fixed was counted. Details on bug collection process can be found in [20]. The following are the metrics used in the classification process:

- Coupling between Objects(CBO)
- Lack of Cohesion (LCOM)
- Number of Children (NOC)
- Depth of inheritance (DOI)
- Weighted Methods per Class(WMC)
- Response for a class (RFC)
- Number of Public Methods(NPM)
- Lines of Code (LOC)

The data collection is performed and subsequently the it is tried to use Correlation-based Feature Subset Selection and Chi-squared Ranking Filter is applied to find the attributes that are important for the prediction. Correlation-based Feature Subset Selection Evaluates the worth of a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them [21]. Subsets of features that are highly correlated with the class while having low intercorrelation are preferred. Chi-squared Ranking Filter evaluates the worth of an attribute by computing the value of the chi-squared statistic with respect to the class.

Thereafter, the reduced number of attributes are givcen as input to the K-means clustering algorithm. As Clustering is a technique that divides data in to two or more clusters depending upon some criteria. As, in this study data is divided in to two clusters depending upon that whether they are fault free or fault prone. In the K-means technique Euclidean distance as well as Manhattan distance measures are experimented. If the Manhattan distance is used, then centroids are computed as the component-wise median rather than mean.

To predict the results, we have used confusion matrix. The confusion matrix has four categories: True positives (TP) are the modules correctly classified as faulty modules. False positives (FP) refer to fault-free modules incorrectly labeled as faulty. True negatives (TN) are the fault-free modules correctly labeled as such. False negatives (FN) refer to faulty modules incorrectly classified as fault-free modules.

The following set of evaluation measures are being used to find the results:

- *Probability of Detection (PD)*, also called recall or specificity, is defined as the probability of correct classification of a module that contains a fault.

  PD = TP / (TP + FN) (1)
- *Probability of False Alarms (PF)* is defined as the ratio of false positives to all non defect modules.

  PF = FP / (FP + TN) (2)

Basically, PD should be maximum and PF should be minimum.

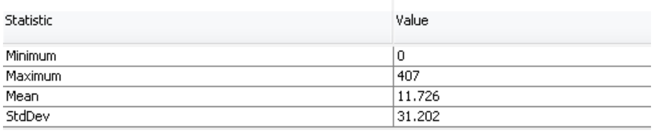
TABLE I
A CONFUSION MATRIX OF PREDICTION OUTCOMES

Predicted		Real Data	
		Fault	**No fault**
	Fault	**TP**	**FP**
	No Fault	**FN**	**TN**

III. RESULT AND DISCUSSION

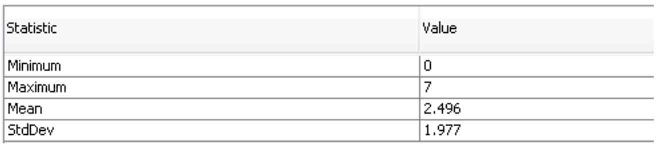
The data is collected from [1] and the statistics of the metric data of the WMC, DIT, NOC, CBO, RFC, LCOM, NPM, LOC metrics is tabulated in Table II, III, IV, V, VI, VII, VIII and IX metrics respectively. The details of the number of Faulty and Non-Faulty Modules present in the dataset is shown in Table X.

TABLE II
STATICS OF THE WMC METRIC VALUES IN JEDIT DATA



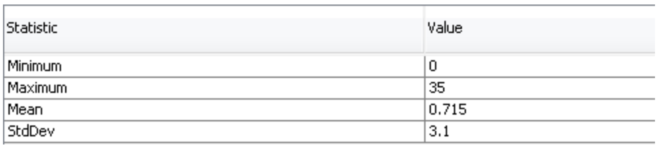
Statistic	Value
Minimum	0
Maximum	407
Mean	11.726
StdDev	31.202

TABLE III
STATICS OF THE DIT METRIC VALUES IN JEDIT DATA



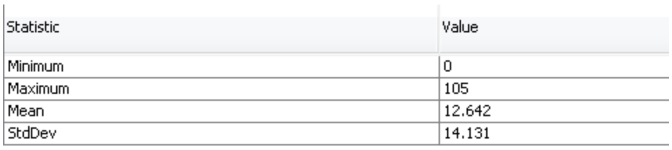
Statistic	Value
Minimum	0
Maximum	7
Mean	2.496
StdDev	1.977

TABLE IV
STATICS OF THE NOC METRIC VALUES IN JEDIT DATA



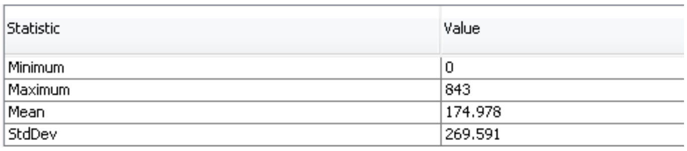
Statistic	Value
Minimum	0
Maximum	35
Mean	0.715
StdDev	3.1

TABLE V
STATICS OF THE CBO METRIC VALUES IN JEDIT DATA



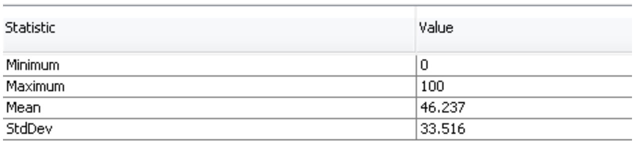
Statistic	Value
Minimum	0
Maximum	105
Mean	12.642
StdDev	14.131

TABLE VI
STATICS OF THE RFC METRIC VALUES IN JEDIT DATA



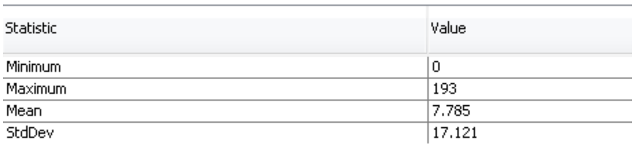
Statistic	Value
Minimum	0
Maximum	843
Mean	174.978
StdDev	269.591

TABLE VII
STATICS OF THE LCOM METRIC VALUES IN JEDIT DATA



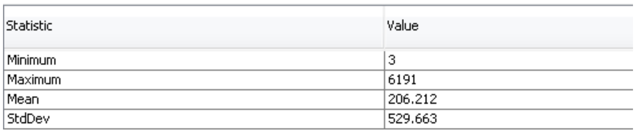
Statistic	Value
Minimum	0
Maximum	100
Mean	46.237
StdDev	33.516

TABLE VIII
STATICS OF THE NPM METRIC VALUES IN JEDIT DATA



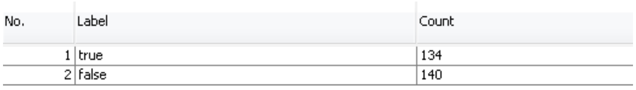
Statistic	Value
Minimum	0
Maximum	193
Mean	7.785
StdDev	17.121

TABLE IX
STATICS OF THE LOC METRIC VALUES IN JEDIT DATA



Statistic	Value
Minimum	3
Maximum	6191
Mean	206.212
StdDev	529.663

TABLE X
STATICS OF THE BUGS PRESENT VALUES IN JEDIT DATA



No.	Label	Count
1	true	134
2	false	140

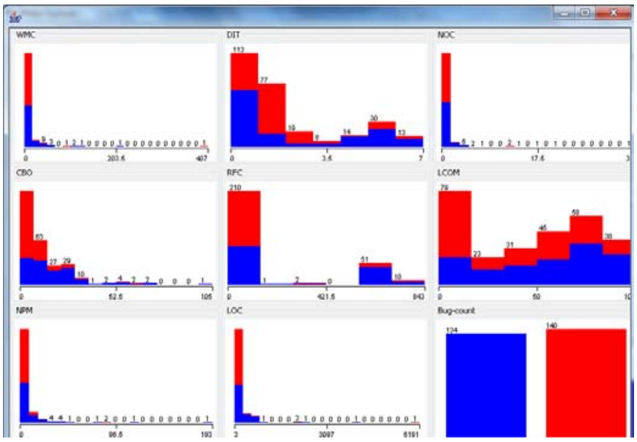


Fig. 1 Graphical Representation of the Statistics of the Metric Data of the Attributes of Dataset

First, the dataset is evaluated using Correlation-based Feature Subset Selection using BestFirst Search. The parameters are:

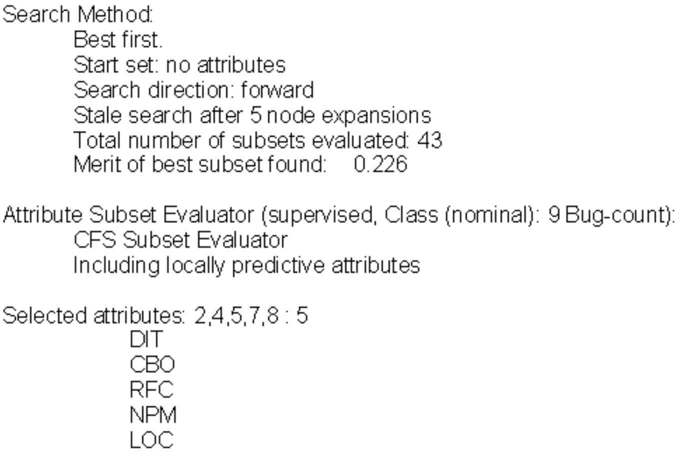
- locallyPredictive -- Identify locally predictive attributes. Iteratively adds attributes with the highest correlation with the class as long as there is not already an attribute in the subset that has a higher correlation with the attribute in question The default value is True is used in the experiment.
- missingSeparate -- Treat missing as a separate value. Otherwise, counts for missing values are distributed across other values in proportion to their frequency. The default value False is used in the experiment.

BestFirst Searches the space of attribute subsets by greedy hillclimbing augmented with a backtracking facility. Setting the number of consecutive non-improving nodes allowed controls the level of backtracking done. Best first may start with the empty set of attributes and search forward, or start with the full set of attributes and search backward, or start at any point and search in both directions (by considering all possible single attribute additions and deletions at a given point).

The parameters used are:

- direction -- Set the direction of the search. The default value 'forward' is used.
- lookupCacheSize -- Set the maximum size of the lookup cache of evaluated subsets. This is expressed as a multiplier of the number of attributes in the data set. It is set to 1.
- searchTermination -- Set the amount of backtracking. It is set to 5.

The figure 2 shows the results after applying Correlation-based Feature Subset Selection using BestFirst Search. It has proposed the use of DIT, CBO, RFC, NPM and LOC metric as significant metrics for the prediction.



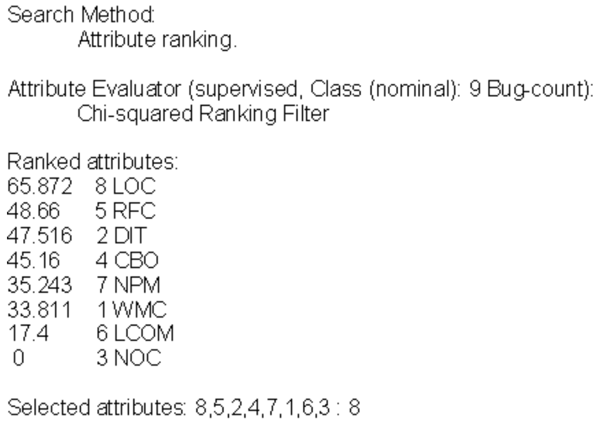
```
Search Method:
        Best first.
        Start set: no attributes
        Search direction: forward
        Stale search after 5 node expansions
        Total number of subsets evaluated: 43
        Merit of best subset found:    0.226

Attribute Subset Evaluator (supervised, Class (nominal): 9 Bug-count):
        CFS Subset Evaluator
        Including locally predictive attributes

Selected attributes: 2,4,5,7,8 : 5
              DIT
              CBO
              RFC
              NPM
              LOC
```

Fig. 3 Snapshot of the Output of Correlation-based Feature Subset Selection using BestFirst Search

In case of Chi-squared Ranking Filter selection Ranks attributes by their individual evaluations. Use in conjunction with attribute evaluators (ReliefF, GainRatio, Entropy etc). The following parameters are used:

- binarizeNumericAttributes -- Just binarize numeric attributes instead of properly discretizing them. The false value of this parameter is used.
- missingMerge -- Distribute counts for missing values. Counts are distributed across other values in proportion to their frequency. Otherwise, missing is treated as a separate value. The false value of this parameter is used.
- generateRanking -- A constant option. Ranker is only capable of generating attribute rankings. The true value of this parameter is used.
- numToSelect -- Specify the number of attributes to retain. The default value (-1) indicates that all attributes are to be retained. Use either this option or a threshold to reduce the attribute set.
- startSet -- Specify a set of attributes to ignore. When generating the ranking, Ranker will not evaluate the attributes in this list. This is specified as a comma seperated list off attribute indexes starting at 1. It can include ranges. Eg. 1,2,5-9,17.
- threshold -- Set threshold by which attributes can be discarded. Default value results in no attributes being discarded. Use either this option or numToSelect to reduce the attribute set. The Default value -1.7976931348623157E308 is used in the experimentation.

Snapshot of the Output of Chi-squared Ranking Filter is shown in figure 3. It shows that rank of the five attributes recommended by the CFS is better than other attributes. So, we selected the 5 attributes as selected by CFS algorithm

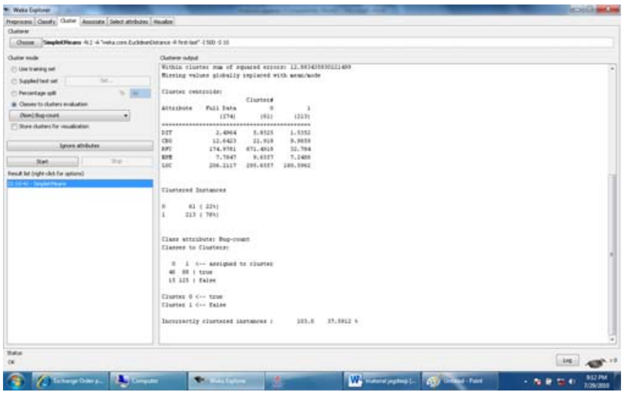


```
Search Method:
        Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 9 Bug-count):
        Chi-squared Ranking Filter

Ranked attributes:
65.872   8 LOC
48.66    5 RFC
47.516   2 DIT
45.16    4 CBO
35.243   7 NPM
33.811   1 WMC
17.4     6 LCOM
 0       3 NOC

Selected attributes: 8,5,2,4,7,1,6,3 : 8
```

Fig. 3 Snapshot of the Output of Chi-squared Ranking Filter

Thereafter, K-Means algorithm is applied on the reduced dataset. First of all the Euclidean distance is used and results are shown in figure 4 and values of the cluster centriods are tabulated in table XI. Second, ManhattanDistance is used and cluster centriods are tabulated in table XII. Manhattan distance (or Taxicab geometry) is the distance between two points is the sum of the (absolute) differences of their coordinates [22]. In both the cases the following parameters are used:

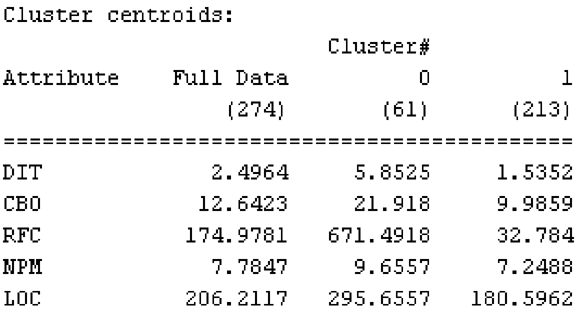
- distanceFunction -- The distance function to use for instances comparison. First set to EuclideanDistance and therafter set to ManhattanDistance.
- dontReplaceMissingValues -- Replace missing values globally with mean/mode. Default value False is used.
- maxIterations -- set maximum number of iterations. It is set to 500.
- numClusters -- set number of clusters. It is set to 2.
- preserveInstancesOrder -- Preserve order of instances. It is set to False.
- seed -- The random number seed to be used. It is set to 10.



```
Within cluster sum of squared errors: 12.883435930221489
Missing values globally replaced with mean/mode

Cluster centroids:
                         Cluster#
Attribute    Full Data          0           1
                 (274)       (61)       (213)
=============================================
DIT             2.4964     5.8525      1.5352
CBO            12.6423     21.918      9.9859
RFC           174.9781   671.4918      32.784
NPM             7.7847     9.6557      7.2488
LOC           206.2117   295.6557    180.5962


Clustered Instances

0       61 ( 22%)
1      213 ( 78%)


Class attribute: Bug-count
Classes to Clusters:

  0   1  <-- assigned to cluster
 46  88 | true
 15 125 | false

Cluster 0 <-- true
Cluster 1 <-- false

Incorrectly clustered instances :      103.0     37.5912 %
```

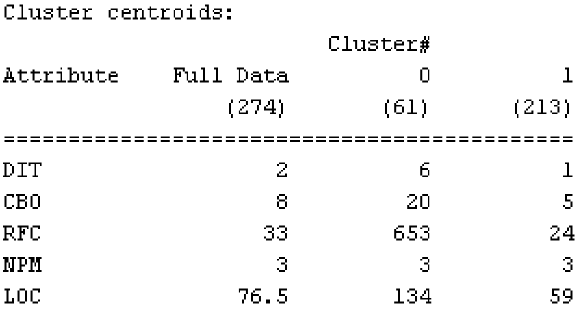
Fig. 4 Snapshot of the Output Kmeans algorithm using Euclidean Distance

TABLE XI
K-MENAS CLUSTER CENTRIODS USING EUCLIDEAN DISTANCE



```
Cluster centroids:
                           Cluster#
Attribute     Full Data           0           1
                  (274)        (61)       (213)
===============================================
DIT              2.4964      5.8525      1.5352
CBO             12.6423      21.918      9.9859
RFC            174.9781    671.4918      32.784
NPM              7.7847      9.6557      7.2488
LOC            206.2117    295.6557    180.5962
```

TABLE XII
K-MENAS CLUSTER CENTRIODS USING MANHATTAN DISTANCE



```
Cluster centroids:
                          Cluster#
Attribute    Full Data          0          1
                 (274)       (61)      (213)
============================================
DIT                  2          6          1
CBO                  8         20          5
RFC                 33        653         24
NPM                  3          3          3
LOC               76.5        134         59
```

The 61 (means 22%) Clustered Instances belongs to cluster 0 and 213 (means 78%) instances belongs to cluster 1. In both the cases the same confusion matrix is recorded as shown in table XIII.

TABLE XIII
RECORDED CONFUSION MATRIX OF PREDICTION OUTCOMES

	Real data		
Predicted		*Fault*	*No Fault*
	Fault	*46*	*88*
	No Fault	*15*	*125*

The Accuracy of classification, Probability of detection (PD) and Probability of False Alarms (PF) values are 62.4% 0.754 and 0.413 respectively.

IV. CONCLUSION

This paper empirically evaluates performance of Kmeans based Clustering technique in predicting fault-prone classes using open source software. The proposed KMeans based classification technique shows 62.4 percent accuracy. It also shows high value of Probability of detection (PD) i.e. 0.754 and low value of Probability of False Alarms (PF) i.e. 0.413.

This study confirms that construction of Kmeans based model is feasible, adaptable to Object Oriented systems and useful in predicting faulty prone classes. It is therefore concluded that model is implemented using Kmeans based technique for classification of the software components into faulty/fault-free systems is found satisfactory. The contributions of the study can be summarized as follows: First open source software systems analyzed. These systems are developed with different development methods than proprietary software. In previous studies mostly proprietary software were analyzed. Second, we examine K-Means clustering method to predict the faulty classes with better accuracy.

The future work can be extended in following directions:

- Most important attribute can be found for fault prediction and this work can be extended to further programming languages.
- More algorithms can be evaluated and then we can find the best algorithm. We plan to replicate our study to predict model based on hybrid genetic algorithms or soft computing techniques.

REFERENCES

[1] http://promisedata.org/repository/
[2] S. Chidamber, and C. Kemerer, "A metrics suite for object-oriented design", IEEE Transactions on Software Engineering, 20(6), 1994, pp.476-493.
[3] Arvinder Kaur and Ruchika Malhotra, “Application of Random Forest in Predicting Fault-Prone Classes”, 2008 International Conference on Advanced Computer Theory and Engineering ICACTE 2008, Pukhet, Dec. 2008, pp. 37-43.
[4] Lanubile F., Lonigro A., and Visaggio G. (1995) “Comparing Models for Identifying Fault-Prone Software Components”, Proceedings of Seventh International Conference on Software Engineering and Knowledge Engineering, June 1995, pp. 12-19.
[5] Saida Benlarbi,Khaled El Emam, Nishith Geol (1999), “Issues in Validating Object-Oriented Metrics for Early Risk Prediction”, by Cistel Technology 210 Colonnade Road Suite 204 Nepean, Ontario Canada K2E 7L5.
[6] Runeson, Claes Wohlin and Magnus C. Ohlsson (2001), “A Proposal for Comparison of Models for Identification of Fault-Proneness”, Dept. of Communication Systems, Lund University, Profes 2001, LNLS 2188, pp. 341-355.
[7] Mahaweerawat, A. (2004), “Fault-Prediction in object oriented software’s using neural network techniques”, Advanced Virtual and Intelligent Computing Center (AVIC), Department of Mathematics, Faculty of Science, Chulalongkorn University, Bangkok, Thailand, pp. 1-8.
[8] Bellini, P. (2005), “Comparing Fault-Proneness Estimation Models”, 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05), vol. 0, 2005, pp. 205-214.
[9] Ma, Y., Guo, L. (2006), “A Statistical Framework for the Prediction of Fault-Proneness”, West Virginia University, Morgantown.
[10] Eric Rotenberg (1999), “AR-SMT: A Microarchitectural Approach to Fault Tolerance in Microprocessors”, Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing, June 15-18, pp. 84-90.
[11] L. Briand, J. Wilst, H. Lounis, "Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs", Empirical Software Engineering: An International Journal, 6(1), 2001, pp.11-58.
[12] T. Gyimothy, R. Ferenc, 1. Siket, "Empirical validation of object-oriented metrics on open Trans. Software Engineering, 31 (10), 2005, pp. 897 —910.
[13] Z. Yuming, and L. Hareton, "Empirical analysis of Object-Oriented Design Metrics for predicting high severity faults", IEEE Transactions on Software Engineering, 32(10), 2006, pp.771-784.
[14] G. Pai, "Empirical analysis of Software Fault Content and Fault Proneness Using Bayesian Methods", IEEE Transactions on software Engineering, 33(10), 2007, pp.675-686.
[15] K.K Aggarwal, Y. Singh, A. Kaur, R. Malhotra, "Empirical Analysis for Investigating the Effect of Object-Oriented Metrics on Fault Proneness: A Replicated Case Study", Published online in Software Process Improvement and Practice, Wiley, 2008.
[16] K.K Aggarwal, Y. Singh, A. Kaur, R. Malhotra, "Investigating the Effect of Coupling Metrics on Fault Proneness in Object-Oriented Systems", Software Quality Professional, 8(4), 2006, pp.4-16.
[17] T.M. Khoshgaftaar, E.D. Allen, J.P. Hudepohl, S.J. Aud, Application of neural networks to software quality modeling of a very large telecommunications system, IEEE Transactions on Neural Networks, 8(4), 1997, pp. 902-909.
[18] Promise. http://promisedata.org/repository/.
[19] Website sourceforge: www.sourceforge.net/projects/jedit
[20] S. Watanabe, H. Kaiya, K. Kaijiri, Adapting a Fault Prediction Model to Allow Inter Language Reuse, PROMISE'08, May 12-13, Leipzig, Germany, 2008.
[21] M. A. Hall (1998). Correlation-based Feature Subset Selection for Machine Learning. Hamilton, New Zealand.
[22] Wikipedia. Taxicab geometry. URL http://en.wikipedia.org/wiki/Taxicab_geometry.