2012

Enhance Rule Based Detection for Software Fault Prone Modules

H. Najadat

Izzat M. Alsmadi
Texas A&M University-San Antonio, ialsmadi@tamusa.edu

Enhance Rule Based Detection for Software Fault Prone Modules

Hassan Najadat [1] and Izzat Alsmadi [2]

[1] *Computer Information Systems Department*
Jordan University of Science and Technology, Irbid, Jordan

[2] *Computer Information Systems Department*
Yarmouk University, Irbid, Jordan

najadat@just.edu.jo, ialsmadi@yu.edu.jo

Abstract

Software quality assurance is necessary to increase the level of confidence in the developed software and reduce the overall cost for developing software projects. The problem addressed in this research is the prediction of fault prone modules using data mining techniques. Predicting fault prone modules allows the software managers to allocate more testing and resources to such modules. This can also imply a good investment in better design in future systems to avoid building error prone modules. Software quality models that are based upon data mining from previous projects can identify fault-prone modules in the current similar development project, once similarity between projects is established. In this paper, we applied different data mining rule-based classification techniques on several publicly available datasets of the NASA software repository (e.g. PC1, PC2, etc). The goal was to classify the software modules into either fault prone or not fault prone modules. The paper proposed a modification on the RIDOR algorithm on which the results show that the enhanced RIDOR algorithm is better than other classification techniques in terms of the number of extracted rules and accuracy. The implemented algorithm learns defect prediction using mining static code attributes. Those attributes are then used to present a new defect predictor with high accuracy and low error rate.

Keywords*: Data mining, software quality, software fault tolerance, software mining, fault prone modules, rule extraction*

1. Introduction

Data mining techniques are utilized and applied in the different fields of science. This is due to the huge amount of existed data and the need to convert them into useful knowledge, which leads to better strategic decision support systems. There are many data mining techniques that can be utilized.

Examples of those techniques include: pattern analysis, association rules, sequential patterns, clustering analysis, classification, neural networks, nearest neighbor, and web mining, etc. The majority of these techniques try to emulate the human skill of decision making. For example, the classification approach is the process of predicting a class label for an object based on a set of predefined objects' class labels. Many real life problems such as; credit evaluation, medical diagnosis or expert systems, speech, voice or image recognition are considered as classification problems [12].

Software mining entails the use of data mining techniques to improve and support the management activities of software projects. The term software mining describes a broad class of investigations into the examination of software repositories [1, 11], such as; fault, effort, or cost prediction, software reuse, and detection of incomplete changes. This paper focuses on the classification of software modules into either faulty or correct modules through the use of data mining techniques.

The importance of this classification comes from the fact that it enables software developers and the testing team to focus on those software modules that are classified as faulty. This allows software managers to allocate their testing budget to those modules that are most likely to contain errors based on the results of the software mining algorithms. As a result, this is expected to effectively improve using the resources available for testing. However, these methods can't replace human or experts' judgment, which is complementary to those techniques. Throughout the years, several data classification techniques have been proposed. They include: decision tree induction, Bayesian classification, K-Nearest Neighbor, case-based reasoning, genetic algorithms, neural networks, rough set approach, and Support Vector Machines (SVMs) [12, 15].

The problem addressed in this paper is the prediction of fault prone modules using data mining techniques. Predicting fault prone modules allows the software managers to allocate more testing and resources to such modules. They may also help them improve the design of future modules. In data mining, historical data is studied to make scientific prediction for future ones, based on establishing similarities between the historical data and the future ones. The aim of this paper is to apply different data mining rule-based classification techniques on some publicly available datasets of the NASA software repository in order to classify the software modules into either fault or not fault prone modules. The selected datasets are: PC1, PC2, PC3, PC4, CM1, MW1, KC3, and KC4. Table 1 provides a set of features about the dataset. A detailed description about those datasets is provided in [22].

Table 1. The Selected Datasets

Data-set	Language	% defect	# of modules
PC1	C	6	1108
PC2	C	4	5590
PC3	C	10	1564
PC4	C	12	1458
CM1	C	9	506
MW1	C	7	404
KC3	JAVA	9	459
KC4	JAVA	49	126

On the other hand, a new algorithm is introduced. This algorithm combines the RIDOR and CLIPPER mining algorithms. RIDOR algorithm and the newly proposed algorithm is that enhanced RIDOR algorithm in terms of effectiveness (i.e. generating less number of rules) and accuracy (i.e. improving the results). The implemented algorithm learns defect prediction using mining of static code attributes. Those attributes are used to present a new defect predictor with enhancements to accuracy and error rate.

In order to overcome using static classification methods such as decision trees, and present defect predictor that improves the accuracy and ability of detecting defective modules for decision trees based predictor, and Naïve Bayes based predictor, a new defect predictor is presented that is expected to improve predicted accuracy. Furthermore, the software attributes used in those prior works, are chosen among the static code attributes that can easily be extracted from a source code, which prevents the consideration of human errors or subjectivity.

The rest of the paper is organized as follows: in section 2, the necessary background to the field of using data mining in software fault predication is provided. In section 3, the methodology is described. In section 4, the experimental results are provided, and in section 5, the paper is concluded with summary and possible future work.

2. Related Works

Several papers are presented about using mining for software fault prediction [4,5,8,9,13,19,23,24,25,14]. Some of those papers discussed methods for fault prediction such as size and complexity metrics, multivariate analysis, and multi-co-linearity using Bayesian belief networks [8, 9]. Naïve Bayes is widely used for building classifier due to its simplicity and optimal accuracy that it delivers based on Bayes theorem. When developing a defect predictor, the probability of each class is calculated, given the attributes extracted from a module, using metrics such as Halstead and McCabe ones (i.e. metrics that are relevant to predicting faulty modules). The modules will then be classified according to the selected attributes or metrics as usually not all attributes will be selected in the classification stage. Menzies et al. developed predictors with Naïve Bayes (NB) for fault characteristics [19]. They have discovered more predictive power in combined or hybrid predictors than in the mono metrics. Mono-metrics lack needed information that can be captured by combined metrics. They found that NB are the best faulty models predictors reported so far. NB provides same weights or different weights to the attributes based on their importance or variation impact.

Olivier et al. have used the Ant Colony Optimization (ACO) algorithm, and the Max-Min Ant System to develop the AntMiner+ model that classifies the dataset into either faulty or non-faulty modules [13]. This algorithm shown to achieve a predictive accuracy that is competitive to other methods.

Predictors that are built using the previous techniques, suffer from high possible errors in assigning records to the correct class. As it will be shown in the results section, NB provides high number of incorrectly classified modules. As a result, many algorithms were built to overcome the significant drawbacks of NB. One of these algorithms that demonstrated the accuracy of NB technique is Lazy Bayes Rules (LBR), which reduces the problem of high errors in NB and demonstrates the accuracy of NB. In other words, LBR produces a classification algorithm with very errors. LBR has high computational overheads, due to high computational complexity required at the classification time. As a result, this will reduce its usefulness as an alternative to NB. It is feasible only when the number of records in the training set is small.

A group of researchers conducted manual software reviews to find defective modules. They found that approximately 60 percent of defects can be detected manually [25]. They found that the accuracy of correctly classified instances in defect detection of industrial review methods is relatively small. Reviews and inspections can find over 50% of the defects in an artifact, regardless of the lifecycle phase applied. It was also

found that 'about 90% of the downtime comes from at most about 10% of the defects [25]. In our study, we learn defect predictors using static code attributes defined by McCabe [24] and Halstead [23]. McCabe and Halstead are module based metrics, where a module is the smallest unit of functionality in the system as a whole.

3. Methodology

The results reported in this paper are based on datasets obtained from the NASA public MDP (Modular toolkit for Data Processing) repository. This is a public repository for NASA datasets. NASA datasets are composed of several static code attributes.

Eight datasets are selected from those available in the repository which will be used in our study as shown in Table 1. Each dataset describes the attributes of each project properties such as size, number of modules, and the number of defects.

All data sets are available in the extension ".arff" extension to enable us processing and analyzing data sets via a data mining tool called Weka [17]. Many classifiers are implemented in publicly available machine learning toolkit Weka. Several data pre-processing steps are conducted to improve data sets quality, which will eventually increase performance of the predictor.

3.1. Preliminary Experiment

Before implementing the enhanced RIDOR algorithm experiment, a preliminary study is conducted on three datasets of the selected 8. The first 3 selected datasets are: PC1, PC4, and KC1. Both PC1 and PC4 describe a flight software project used for an earth orbiting satellite written in C. PC1 contains 40 KLOC, PC4 36 KLOC. KC1 size is 43 KLOC written in C++ and is a subsystem of a larger ground control system. The number of attributes for each of the PC1, PC4, and KC1 is 43, 43, and 27 respectively. Table 2 shows summary statistics of the first 3 selected datasets.

Table 2. Dataset Characteristics

Dataset	PC1	PC4	KC1
Size	1107	1458	2107
Faulty Modules Number	77	179	326
% of faulty modules	6.87	12.21	15.42
Metrics Number	40	40	21

All classification methods used can only deal with discrete variables or categorical values. A discretization or digitization or discretization step is implemented using Weka with an entropy-based procedure to turn all continuous variables into discrete ones (through averaging and selecting median representatives).

To make sure that only relevant variables are included in the datasets, an input selection procedure is implemented using an x22-based feature selection algorithm available in Weka. Later on, 12 attributes along with the class attribute were retained for each selected dataset. Table 3 summarizes the retained attributes after the input selection. Through selection and evaluation, those attributes prove to be more relevant than others.

3.2. Enhanced RIDOR Algorithm Analysis and Assessment

In order to make an initial assessment of accuracy and effectiveness of existing classifiers, several classifiers are selected to be applied on the selected MDP datasets. The selected classifiers include: JRIP, NNge, PART, PRISM, RIDOR and AntMiner. JRip is an implementation Cohen's RIPPER [14]. RIPPER creates first a default rule and then recursively develops exceptions to it. Part constructs rules based on partial decision trees. NNge is a nearest neighbor algorithm forming non-nested general exemplars. It is a nearest neighbor algorithm which learns rules based on the hyper rectangles it divides the instance space into [2]. PART creates a rule set by repeatable creating pruned decision trees using J4.8 [18]. PRSIM is an attribute-value pair oriented approach that generates rules from training set directly. It is a covering based method [21]. The purpose of Ant-Miner is to use ants to create rules describing an underlying data set [16].

The RIpple DOwn Rule (RIDOR) is another algorithm that produces a set of rules. It works by generating a default rule, after which exceptions are generated in a tree-like fashion until all training instances are classified correctly according to the rule set. RIDOR is the algorithm that usually gets the best classification scores [10,3].

The result of each classifier is compared with others in terms of accuracy, number of extracted rules, and the time in seconds, needed to build the classifier. The performance of all selected classifiers is summarized in Table 4.

In Table 4, NNge has the highest accuracy, while JRIP comes in seconds and followed by PRISM. Their accuracy values are: 96.64%, 91.86%, and 88.67% respectively. However these algorithms tend to generate large number of rules (NNge with 301 for example), which in turns make the extracted rules incomprehensible and hard to interpret by humans.

Table 3. Attributes Retained After Input Selection

	PC1	PC4	KC1
Branch count	⋆		⋆
Call pairs		⋆	⋆
Condition count	⋆		
Cyclomatic complexity	⋆		⋆
Cyclomatic density		⋆	⋆
Decision complexity			⋆
Decision count			⋆
Decision density			⋆
Design density			⋆
Error density	⋆		
Essential complexity	⋆	⋆	
Halstead content	⋆	⋆	
Halstead difficulty	⋆		
LOC blank	⋆	⋆	⋆
LOC code and comment	⋆		⋆
LOC comments	⋆	⋆	⋆
LOC executable	⋆	⋆	
LOC total	⋆	⋆	
Normalized complexity		⋆	⋆
Num operands		⋆	
Num unique operands		⋆	
Percent comment		⋆	

Table 4. The Performance of the Classifiers

	Accuracy %	No. Of Rules	Time (s)
JRIP	91.86	33	6.56
NNge	96.64	301	0.92
PART	82.17	42	0.28
PRISM	88.67	68	0.22
RIDOR	75.73	16	0.2
AntMiner	75.42	13	576

On the other hand, AntMiner performs best in terms of the number of extracted rules, closely followed by RIDOR. Such methods are more compressible and can be understood easily by humans. When considering the time that is required to build a classifier model for each algorithm, we can see that the AntMiner is the worst in comparison with other algorithms which take very short time to construct their model.

As the results have shown, AntMiner and RIDOR algorithms are the best, of those selected, regarding the effectiveness of the extracted rules. The new proposed enhanced RIDOR algorithm combines the RIDOR and CLIPER algorithms to produce few numbers of rules while at the same time increase the RIDOR accuracy. RIDOR algorithm generates the default rule first. It then generates the exceptions for the default rule with the least weighted error rate. It generates the best exceptions and performs a tree-like expansion of exceptions. The leaf has only default rule with no exceptions. The exceptions are set of rules that predict the class (other than default class). Table 5 and 6 show performance comparison between the enhanced RIDOR algorithms and AntMiner on the PC1 and PC4 datasets respectively.

Table 5. Performance Comparison on PC1 Dataset

	No. of rules	Accuracy %	Time (s)
Enhanced RIDOR	6	85.32	0.6
AntMiner	7	69.79	180

Table 6. Performance Comparison on PC4 Dataset

	No. Of rules	Accuracy %	Time (s)
Enhanced RIDOR	6	67.57	0.2
AntMiner	7	70.15	420

The enhanced RIDOR algorithm tries to benefit of advantages of the two algorithms: CLIPER and RIDOR. CLIPER algorithm was particularly proposed to efficiently mine a closed set of iterative patterns. The efficiency stems from its pruning strategy. We will adapt this strategy and applied it to the list of rules extracted from RIDOR classifier. The procedure to implement this new algorithm was as the following:

1. Extract the rules from the RIDOR, retain the rules of faulty classes, and assign all other rules to the rest.
2. Encode each attribute as symbols and consider it as an event. For example, the Halstead content attribute is referenced to as H and its order intervals as: (H1, H2, H3, H4, H5, etc.).

3. Those attributes will be then compacted or merged to only two: (i.e. H1, H5) based on the values of Halstead Content attribute. Furthermore, the iterative events can also be merged. For example the iterative event: H, B, C, and D can be compacted to (H, D). This may result in classification of non-faulty modules into faulty ones. However, this can be less problematic than the opposite.
4. Repeat previous steps for all predecessors attributes in the rules list.

This procedure is applied to PC1, PC4, KC1 and the modified dataset. The results were compared with those of AntMiner as an alternative algorithm to RIDOR. The results have shown that the enhanced RIDOR algorithm tends to generate fewer numbers of rules with approximately the same accuracy of AntMiner. Tables 7 and 8 summarize the performance of the proposed algorithm with the AntMiner algorithm in terms of number of rules, accuracy, and the time needed to build the classifier.

3.3. Evaluating the Enhanced RIDOR Algorithm

The modified datasets are passed to three learners from Weka. Those are: Naïve Bayse, LBR, and Averaged One- Dependence Estimators (AODE) [20].

The selected techniques are based on those that will weaken NB's attribute independence assumption, achieving the error performance of LBR, without affecting their computational load. Analyzing LBR shows that this computational load is an indication of two factors: model selection, and probability estimation. One way to minimize the computation required for model selection is to perform no model selection, as does NB.

Table 7. Performance Comparison on KC1 Dataset

	No. of rules	Accuracy %	Time (s)
Enhanced RIDOR	9	61	0.2
AntMiner	8	59.9	300

Table 8. Performance Comparison on the Modified Dataset

	No. of rules	Accuracy %	Time (s)
Enhanced RIDOR	2	79.6	0.02
AntMiner	6	78	5

In addition to the desire to minimize computation, a second motivation for avoiding model selection is that selection between alternative models can be expected to increase variance. This is because selection between models allows a learning system to more closely fit the training data. In consequence, changes in the training data will lead to greater changes in the model, which leads to greater variance. In contrast, under approaches such as NB where there is no choice in the form of the model, all that changes when the training data changes is the underlying conditional probability tables which tends to result in relatively gradual changes in the patterns of classification. Model selection avoidance may minimize the variance components of a classifier's error.

Applying the NB product rule, it follows that for any attribute value xi: P(y, x) = P(y, xi)P(x | y, xi)

As this equality holds for every xi, it follows that it also holds for the mean over any group of attribute values. Hence,

$$P(y,x) = \frac{\sum_{i:1\le i\le n \wedge F(xi)\ge m} P(y,xi)P(x \mid y,xi)}{|\{i : 1 \le i \le n \wedge F(xi) \ge m\}|}$$

Where F(xi) is a count of the number of training examples having attribute-value xi and is used to enforce the limit, m that is placed on the support needed in order to accept a conditional probability estimate [20].

In order to obtain the error performance of the LBR, without its computational burden, AODE technique is introduced, that demonstrated the NB accuracy and has the ability of classification with low probability of errors as LBR, without suffering from high computational complexity.

The reasons for using AODE in Our defect prone predictor is that the evaluation of any predictor based on its accuracy, theory and experiments prove that a predictor built with AODE has better accuracy and error performance than NB.

AODE can be used to generate defect-prone predictors. Before a set of attributes are passed to the predictors, a number of attributes can be removed without damaging the performance of the predictor. Several suggested algorithms for attribute subset selection can find what attributes can be removed as a data reduction form. The simplest and fastest subset selection method is to compute information gain for each attribute and then rank attributes from the most to the least informative attribute [7].

Eight sets of static code attributes data is used to learn predictors. Infogain is computed for each attribute in each data set, to select only highest two from the thirty-eight attributes. Two predictors are presented: LBR and AODE.

Each of these models plus the previous NB model are implemented several times using Weka in order to examine their accuracy. Moreover, we implement (M =10) * (N=10) cross-validation, in which the dataset is divided into 10 buckets. The model was trained using 9 buckets and the tenth is used for the dataset. The cross validation is implemented several times and then the average is computed. In each time, one model is implemented, and a cross validation is conducted via Weka. The data is rearranged randomly by choosing a filter provided by Weka (Figure1).

The performance of the learners affects significantly the efficiency of the predictors. The ability of one predictor to detect defective modules depends on learners' accuracy. Strong predictors use high accurate learner, with low performance errors (i.e. incorrect classified records). The accuracy measure is important to assess predictors. Accuracy is defined as the percentages of the correctly classified instances to the percentages of the incorrectly classified instances. There are four possible results for classifying modules. Modules can be faulty and classified as either faulty (i.e. True negative), or not faulty (i.e. False negative). They can also be not faulty and classified as either not faulty (i.e. True positive), or faulty (i.e. False positive).

Various comparative tests are performed to select the best technique. NB, LBR and AODE methods are applied on the tested datasets using WEKA. The results of those runs based on the above measures are shown in Table 9.

Results show that the accuracy of the correctly classified instances is better using LBR and AODE methods relative to NB in terms of accuracy. However, LBR takes much time to run relative to the other modules. AODE shows the best performance with

log-transforms since it has a majority of accuracy of the correctly classified instances and a minority of percentage of incorrectly classified instances.

In AODE, initial runs with iterative InfoGain sub setting illustrated that all datasets could be reduced from having 38 to 3 attributes (except PC1) without impacting the performance. However, iterative sub setting caused selecting seven attributes for PC1. Therefore, for this dataset only, exhaustive sub setting was applied on 27 subsets to discover the three best attributes. Large reductions in the number of attributes were produced without impacting the performance of the learned predictor.

```
M = 10 , N = 10
All = 38  # all the attributes
DATAS=(pc1 pc2 pc3 pc4 cm1 wm1 kc3 kc4) # data set
FILTERS = (none logNums) # filter list
LEARNERS = (NB LBR AODE)  # learner list
For data in DATAS
  For filter in  FILTERS
      Data' =filter (data)
      Rank data' attributes via InfoGain
      For i=1,2,3,All
         Attribute' = the i-th highest ranked attributes
         Data ''  = select attributes' from data'
         Repeat M times
          Randomized order from data''
           Generate N bins from data''
           For I in 1 to N
               Tests     =bin[i]
               Training Data = data'' – tests
               For learner in LEARNERS
                    METHOD =(filter attributes')
                     Predictor =learner(training Data)
             RESULT (METHOD)= apply predictor to tests
```

Figure 1. The Modified Generic Algorithm

Table 9. Accuracy Results from Selected Datasets

Data Sets	Accuracy NB	Accuracy LBR	Accuracy AODE	Selected Attributes
PC1	79.58%	92.68%	93.22%	3,35,37
PC2	98.28%	98.32%	99.37%	5,39
PC3	84.52%	89.19%	90.61%	1,20,37
PC4	82.30%	86.63%	87.45%	1,4,39
CM1	72.28%	89.31%	90.10%	5,35,36
MW1	85.61%	91.56%	92.31%	23,31,35
KC3	71.18%	90.61%	90.61%	16,24,26
KC4	73.60%	74.40%	68.80%	3,13,31
AVG	80.92%	89.09%	89.06%	

4. Conclusion

The goal of fault prone modules' prediction using data mining is to improve the software development process. This enables the software manager to effectively allocate project resources toward those modules that require more effort. This will eventually enable the developers to fix the bugs before delivering the software product to end users.

In this paper a rule-based classification method is suggested for the classification of modules from their fault prone characteristic. A modification is proposed on the RIDOR algorithm or rule to produce smaller number of rules, while at the same time increase its accuracy. Same goal can be evaluated in future applied to other rule-based classification methods such as AntMiner, PRISM, and JRIP.

Several pre-processing techniques were proposed to improve the accuracy using classifiers such as LBR and AODE. Similarly, static code attributes with log filtration is used to improve accuracy and performance.

References

[1] Ahmed E. Hassan, Richard C. Holt. Guest Editors' Introduction: Special Issue on Mining Software Repositories. IEEE computer society **(2005)**

[2] Martin B. Instance-Based learning: Nearest Neighbor With Generalization. Master Thesis, University of Waikato, Hamilton, New Zealand **(1995)**

[3] Witten, I.H. and Frank, E., Data Mining: Practical Machine Learning Tools and Techniques (San Francisco, CA: Morgan Kaufmann), 2nd edition **(2005)**

[4] Compton et al. Ripple Down Rules: Possibilities and Limitations. Proceedings of the Sixth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop. pp.6-1-6-20. Calgary, Canada, University of Calgary **(1991)**

[5] David Lo, SiauCheng Khoo, Chao Liu. Efficient Mining of Iterative Patterns for Software Specification Discovery. SIGKDD **(2007)**

[6] David Martens, Manu De Backer, Raf Haesen, Jan Vanl hienen, Monique Snoeck, Bart Baesens. Classification with Ant Colony Optimization. IEEE, VOL. 11. No.5 **(2007)**

[7] Viktor Jovanoski Nada, Nada Lavrac, Feature Subset Selection in Association Rules Learning Systems, In Proceedings of Slovenian Electrical and Computer Science Conference ERK'99 **(1999)**

[8] Fenton N.E, Ohlsson N, Quantitative Analysis of Faults and Failures in a Complex Software System. IEEE Transactions on Software Engineering **(1999)**

[9] Fenton N.E., Neil M, Critique of Software Defect Prediction Models. IEEE Transactions on Software Engineering **(1999)**

[10] Gaines, B.R., Compton, P.: Induction of Ripple-Down Rules Applied to Modeling Large Databases. J. Intell. Inf. Syst. 5(3), 211–228 **(1995)**

[11] Huzefa Kagdi, Michael L. Collard, Jonathan I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. Journal of Software Maintenance And Evolution: Research And Practice **(2007)**

[12] Jiawi Han, Micheline Kamber. Data Mining Concepts and Techniques. Second edition, Morgan Kaufman Publishers **(2006)**

[13] Oliver Vanderuys, David Martens BartBaesens, Christophe Mues, Manu De Backer, Raf Haesen. Mining Software Repositories for comprehensible Software Fault Prediction Models. The Journal of Systems and Software 81 **(2008)** 823-839.

[14] Cohen W. W. Fast Effective Rule Induction. Proceedings of the 12th International Conference, on Machine Learning, 115-123 **(1995)**

[15] Richard O.Duda, Peter E.Hart, David G.stork. Pattern Classification. Second edition, John Wiley and sons **(2001)**

[16] M.D. Backer, R. Haesen, D. Martens, B. Baesens. A stigmergy based approach to data mining. Proceedings of the 18th Australian Joint Conference **(2005)**

[17] WEKA data mining toolkit. http://www.cs.waikato.ac.nz/~ml/weka.

[18] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for eclipse. In PROMISE'07: International Workshop on ICSE Workshops 2007, pages 9–9 **(2007)** May.

[19] Tim Menzies, Jeremy Greenwald, and Art Frank. Data Mining Static Code Attributes to Learn Defect Predictors". IEEE Transactions on Software Engineering **(2007)** January.

[20] G. Webb, J. Boughton, and Z. Wang. Not So Naive Bayes:Aggregating One-Dependence Estimators. Machine Learning, vol. 58, no. 1, pp. 5-24 **(2005)**

[21] Zhang Xia1, Yin Yixin, Meng Xiuyan, and Zhao Hailong, Text Classification based on Rule Mining by Granule Network Constructing, Fifth International Conference on Fuzzy Systems and Knowledge Discovery **(2008)**

[22] Metric Data Program MDP http://mdp. ivv.nasa.gov

[23] M. Halstead, Elements of Software Science. Elsevier **(1977)**

[24] T. McCabe. A Complexity Measure. IEEE Trans. Software Eng.,vol. 2, no. 4, pp. 308-320 **(1976)** December.

[25] F. Shull, V. B. ad B. Boehm, A. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero, and M. Zelkowitz. What we have learned about fighting defects. Proceedings of 8th International Software Metrics Symposium, Ottawa, Canada **(2002)** pp. 249–258.

Authors

Dr. Hassan Najadat is an assistant professor in computer science. He got his phd from NDSU, Fargo, ND in 2005. He is working in the computer information system department at Jordan University of Science and Technology, Ramtha, Jordan. His research interests are focused on data mining.

Izzat Alsmadi is an assistant professor in software engineering. He got his phd from NDSU, Fargo, ND in 2008. He is working in the computer information system department at Yarmouk University, Irbid, Jordan. His research interests are focused on software testing and metrics.