Software Fault Prediction at Design Phase

Pradeep Singh*, Shrish Verma** and O. P Vyas†

Abstract – Prediction of fault-prone modules continues to attract researcher’s interest due to its significant impact on software development cost. The most important goal of such techniques is to correctly identify the modules where faults are most likely to present in early phases of software development lifecycle. Various software metrics related to modules level fault data have been successfully used for prediction of fault-prone modules. Goal of this research is to predict the faulty modules at design phase using design metrics of modules and faults related to modules. We have analyzed the effect of pre-processing and different machine learning schemes on eleven projects from NASA Metrics Data Program which offers design metrics and its related faults. Using seven machine learning and four preprocessing techniques we confirmed that models built from design metrics are surprisingly good at fault proneness prediction. The result shows that we should choose Naïve Bayes or Voting feature intervals with discretization for different data sets as they outperformed out of 28 schemes. Naïve Bayes and Voting feature intervals has performed AUC > 0.7 on average of eleven projects. Our proposed framework is effective and can predict an acceptable level of fault at design phases.

Keywords: Software metrics, Machine learning, Design metric, Fault prediction

1. Introduction

Software faults are the imperfections in software development process that would cause software to fail to meet the desired expectations. Faults can be introduced, by numerous problems occurring at requirement specification, design, code and testing phase. They can also be caused by external factors, such as environmental disturbances or human actions, either accidental or deliberate. If faults occur in early phases like requirement and are not identified, they can be very difficult to remove in later phases. Software design fault results from the design of algorithms, control, logic, data elements, module interface descriptions, and external software / hardware / user interface descriptions. Coding defects are derived from errors in implementing the code. Coding defects can arrive from a failure to understand programming language constructs, and miscommunication with the designers. Test plans, test cases, test harnesses, and test procedures can also contain defects. U.S. economy estimated $60 billion (approx) cost due to software failure each year, this is equivalent to 0.6 percent of gross domestic product, revealed by recent study commissioned by the Department of Commerce's National Institute of Standards and Technology (NIST). They pointed that improvements in testing can reduce the cost by $20.0 billion, approximately, but it won’t eliminate all software errors [1].

The reason that the cost of correcting a fault increases so rapidly is related to the rigorous process of software fault detection and correction. In early phases of the software processes exist only on the paper, and correcting a fault may require to make the change in the documents. Software bug removal takes around 40% of total development effort [2]. Identifying which module is most likely to be error prone is a critical task. By identifying the faulty module in early phases; software resource allocation for bug removal to those modules can be done in appropriate manner. Software fault prediction can be defined as let \( X_d = \{x_1, x_2, \ldots, x_n\} \) be a set of vector in \( \mathbb{R}^p \) here \( x_i \) represents a software module that assigned the software metric at design time and \( Y = (fp, nfp) \) in \( \mathbb{R}^2 \) denotes its binary class (fault-prone or not fault-prone). Let there is an unknown function \( S \) which transforms \( X_d \) to \( Y \). Our goal is to estimate \( S \) as an efficient combination of preprocessing and learner for fault prediction.

In this paper we developed models for faulty module prediction at design phase by using design time software metrics. For this purpose we have extracted design metrics from 11 projects. These 11 software projects fault data are taken from Promise repository [3]. In this paper, after describing the project sets, we have conducted an extensive empirical study for investigation of the effect of pre-processing and dimensionality reduction on different machine learning techniques for fault identification using the design metrics only; We find that models built from design metrics are useful as they are built in early phases of the development life cycle and these models have efficient predictability using design metric; We
therefore recommend that in future, researchers explore the effects of the attributes from design phases of the development life cycle for software fault prediction. It is also interesting to study that how fault identified at design time can improve the next level predicting capabilities. The remaining details are divided in following sections. Section 2 describes the related work in the area. Section 3 discusses about software project and their corresponding design metrics. Section 4 presents the 28 scheme which includes pre-processing and dimensionality reduction and machine learners. Section 5 provides the results and discusses their implications, and Section 6 concludes with a summary and future work.

### 2. Related Work

Software researchers have visualize from the study that cost incurred to correct faults increase exponentially with the time they remain uncorrected in the system. It is therefore advisable to eliminate as many faults as early as possible during software development [4]. Code reviews, are labor-intensive and one member can inspect only 8 to 20 LOC/minute. Same review process will be repeated for all members of the review team, which can be as large as four or six [7]. One effective approach for early identification of faulty modules is software fault prediction which investigates and learns the characteristics of individual code and design segments to predict the fault-prone modules [4]. Earlier number of studies have been conducted but they have used metrics after coding phase i.e. they used all design metrics of coding and design phase [4, 5, 16]. In this study, our emphasis is on early fault prediction model at design phase for identification of faulty segments. For this purpose, a large number of software design characteristics have been extracted from standard software projects. These design characteristics includes Branch Count, Call Pairs, Condition Count, Cyclomatic Complexity design metrics etc. The detailed list of design metrics used in the study is shown in Table 2.

Koru et. al. investigated change-prone classes with tree based models on K-Office and Mozilla by using class label metric. They found that around majority of change is rooted in a small proportion of (around 20%) classes and tree based models were very useful to identify these change-prone classes [19]. We have also conducted a study to identify the impact of feature reduction method on software fault prediction performance of learners [18].

In 2012, Singh et al. [16] have utilized two different machine learning algorithm: 348 (a decision tree) and Naïve Bayes algorithm on open source project developed in object oriented language They used the object oriented metric and reported the prediction accuracy 98.15% and 95.58% respectively. Shull et al. [6] found that around 60% of bugs can be identified by manual review. Lessmann et al. [5] suggested that to get confidence we need to develop reliable research and comparative study for software prediction models. Hall [20] mentioned that data preprocessing such as the removal of noninformative features or the discretization of continuous attributes may improve the performance of some classifiers. Thus, we stress that the aim of this paper is to evaluate different pre-processing and dimensionality reduction processes for finding effective fault prediction models at design phase.

### 3. Metrics Description

The software projects used in this study are from the NASA Metrics Data Program (MDP) and we have extracted design level metrics for these projects. Eleven software projects used are described in Table 1. Column 2 of Table 1 shows the total numbers of modules in the software projects and column 3 shows the number of modules that have some faults. Column 5 describes the language used in the development of the project. KC1 and KC3 projects of NASA Metrics Data Program (MDP) and we have extracted design level metrics for these projects. Eleven software projects used are described in Table 1. Column 2 of Table 1 shows the total numbers of modules in the software projects and column 3 shows the number of modules that have some faults. Column 5 describes the language used in the development of the project. KC1 and KC3 projects of NASA mission are responsible for the storage management for ground data. MW1 project is a zero gravity experiment related to combustion. MC1 is software for combustion experiment on the space shuttle. MC2 is a video guidance system. JM1 project is a real-time ground system that uses simulations to generate certain predictions for mission’s. PC1, PC2, PC3, PC4 and PC5 are the flight software for earth orbiting satellite. More details of the project development can be found at [21].

We have extracted the design metrics shown in Table 2 from these projects. Project KC3, PC3, PC4, MW1, MC2 and PC2 have 15 total design metrics. MC1 and PC5 have 14 design metrics. KC1, JM1 and PC1 have 4 attributes branch count, cyclomatic complexity, design complexity. All 11 projects having module segment id and its associate fault status that shows that module is fault or not faulty. After removing other attributes, the project data is ready for pre-processing. We have used three different pre-processing schemes. After pre-processing we have applied seven different machine learning algorithm. The details description of each project is given in Table 1. The design metrics shown in Table 2 have been processed by McCabe IQ 7.1 [8].

### Table 1. Dataset used in the study

<table>
<thead>
<tr>
<th>Data</th>
<th>Modules</th>
<th>Faulty modules</th>
<th>Design metric</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>KC3</td>
<td>458</td>
<td>43</td>
<td>15</td>
<td>Java</td>
</tr>
<tr>
<td>PC3</td>
<td>1565</td>
<td>160</td>
<td>15</td>
<td>C</td>
</tr>
<tr>
<td>PC4</td>
<td>1458</td>
<td>178</td>
<td>15</td>
<td>C++</td>
</tr>
<tr>
<td>MW1</td>
<td>403</td>
<td>31</td>
<td>15</td>
<td>C</td>
</tr>
<tr>
<td>MC2</td>
<td>161</td>
<td>52</td>
<td>15</td>
<td>C++</td>
</tr>
<tr>
<td>MC1</td>
<td>9466</td>
<td>68</td>
<td>14</td>
<td>C/C++</td>
</tr>
<tr>
<td>PC2</td>
<td>5589</td>
<td>23</td>
<td>15</td>
<td>C</td>
</tr>
<tr>
<td>PC5</td>
<td>17,186</td>
<td>516</td>
<td>14</td>
<td>C++</td>
</tr>
<tr>
<td>KC1</td>
<td>2407</td>
<td>326</td>
<td>4</td>
<td>C++</td>
</tr>
<tr>
<td>JM1</td>
<td>10,878</td>
<td>2106</td>
<td>4</td>
<td>C</td>
</tr>
<tr>
<td>PC1</td>
<td>1107</td>
<td>77</td>
<td>4</td>
<td>C</td>
</tr>
</tbody>
</table>
Table 2. Design metrics used in this study

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch_Count</td>
<td>Branch count metrics</td>
</tr>
<tr>
<td>Call_Pairs</td>
<td>Number of calls to other functions in a module</td>
</tr>
<tr>
<td>Condition_Count</td>
<td>Number of conditions in a module</td>
</tr>
<tr>
<td>Cyclomatic_Complexity: ( v(g) )</td>
<td>The number of logical independent paths ( v(g) = e - n + 2 )</td>
</tr>
<tr>
<td>Decision_Count</td>
<td>Number of decision points in a given module</td>
</tr>
<tr>
<td>Decision_Density</td>
<td>Condition count / Decision count</td>
</tr>
<tr>
<td>Design_Complexity: ( iv(g) )</td>
<td>The design complexity of a module</td>
</tr>
<tr>
<td>Essential_Complexity: ( ev(g) )</td>
<td>The essential complexity of a module</td>
</tr>
<tr>
<td>Essential_Density</td>
<td>Essential density is calculated as: ( ev(g) - 1 / iv(g) - 1 )</td>
</tr>
<tr>
<td>Maintenance_Severity</td>
<td>Maintenance severity is calculated as: ( ev(g) / iv(g) )</td>
</tr>
<tr>
<td>Multiple_Condition_Count</td>
<td>Number of multiple conditions that exist within a module</td>
</tr>
<tr>
<td>Node_Count</td>
<td>Number of nodes found in a given module</td>
</tr>
<tr>
<td>Design_Density</td>
<td>Design density = ( iv(g) / v(g) )</td>
</tr>
<tr>
<td>Edge_Count</td>
<td>Number of edges found in a given module</td>
</tr>
<tr>
<td>Modified_Condition_Count</td>
<td>The effect of a condition affect a decision outcome by varying that condition only</td>
</tr>
</tbody>
</table>

The design metrics are processed from design phase artifacts. These are design diagrams, data flow diagrams, control flow graphs, and Unified Modelling Language diagrams. The best part of these metrics are that they are available before coding phase. The design metrics mined from design artifact and used in the study are listed in Table 2.

4. Experimental Design

Seven machine learning algorithms used from Weka for empirical data analysis [9]. We use 11 NASA MDP projects, with the design metrics extracted for each project. We have used area under the receiver operating characteristic curve to measure the performance of fault prediction models. This experiment is intended to demonstrate the effectiveness of early fault prediction using different preprocessing and learning scheme using design metrics. We have used various well-known and effective classifiers from different categories for our experiment. For this purpose, 28 different learning schemes have been designed according to the following data preprocessing, and learning algorithms.

- Four data pre-processors
  - None: data unchanged;
  - Discretization: entropy based discretization
  - Binning (equal-width binning.)
  - Principal component analysis

- Seven learning algorithms: Naïve Bayes (NB), Voting feature intervals (VFI), Simple logistic, KNN, J48, OneR and Random forest.

Twenty eight learning schemes: The combination of four data pre-processors and seven learning algorithms yields a total of 28 different learning schemes: for e.g Naïve Bayes learner have four

- NB + None
- NB + PCA
- NB + binning
- NB + entropy based;

PCA is used abundantly in different analysis techniques for extracting relevant information from data set like pattern recognition, neurocomputing and computer graphics. PCA is a generalized dimensionality reduction technique so details are left.

We used the state-of-the-art supervised discretization technique developed by Fayyad and Irani [10] for binary discretization, of continuous attribute into two intervals. This entropy based discretization process is a viable choice as it improves accuracy [17, 20].

Given a set of samples, \( S \), Entropy for \( S \) can be calculated from Eq. (1)

\[
\text{Entropy}(S) = -p_1 \log_2(p_1) - p_2 \log_2(p_2)
\]  

Now suppose if \( S \) is partitioned into two intervals \( S_1 \) and \( S_2 \) using boundary \( T \), the entropy after partition can be calculated as the equation given in Eq. (2)

\[
E(S, T) = \left| \frac{S_1}{|S|} \right| \text{Ent}(S_1) + \left| \frac{S_2}{|S|} \right| \text{Ent}(S_2)
\]

\( T \) is taken from the midpoints of the feature values. \( \text{Ent}(S_1) \) and \( \text{Ent}(S_2) \) are the entropy of \( S_1 \) and \( S_2 \) respectively. The goal is to get the maximum information gain after the split and the gain is calculated as

\[
\text{Gain}(S, T) = \text{Entropy}(S) - E(S, T).
\]

Recursively evaluating all possible splits and then selecting the best split(s) with maximum gain is chosen. A partition point is decided for discretization which minimizes the entropy function. The process is recursively applied to obtain partitions until it minimizes the difference to acceptable level e.g.

\[
\text{Ent}(S) - E(S, T) > \delta
\]

Minimal descriptive length principle is used by [10] to determine stopping criteria for recursive discretization process. Recursive partitioning for attribute \( A \) within a set of value \( S \) stops if the Gain satisfies the following criteria

\[
\text{Gain}(A,T;S) < \frac{\log_2(N - 1)}{N} + \frac{\Delta(A,T;S)}{N}
\]

Here

\[
\Delta(A,T;S) = \log_2(3^t - 2) - \{k \cdot \text{Ent}(S) - k_1 \cdot \text{Ent}(S_1) - k_2 \cdot \text{Ent}(S_2)\}
\]
And $k_i$ is the number of class labels represented in the set $S$.

Next scheme for pre-processing used is simple equal-width binning algorithm which divides the data into $k$ intervals of equal size. The width of intervals is: $i = (max - min)/k$ and the interval boundaries are: $min + i, min + 2i... min + (k-1)i$.

In this experiment, we have used 10-fold cross validation. For this, we took randomly 90 percent of the data as training data and the remainder 10% as test data and this process is repeated 10 times. The experiment calculates 3, 080 AUC curve (11 different data sets * one design metric groups * 7 machine learners * 4 pre-processing for each learner * 10 runs). For each model we have performed the same cross validation to get AUC. Table 3 shows the result for the AUCs from each dataset with pre-processing metric group and machine learner experiment. We compare the performance of each models derived from pre-processing and machine learning on design metrics.

4.1 Machine learners

Out of seven learning algorithms the first one is Naive Bayes classifier (NB). NB has been used extensively in fault-proneness identification, for example in [4]. Naive Bayes implements the probabilistic classifier. It is easy to implement on different types of datasets, also provides good results. Naive Bayes makes the assumption that feature values are statistically independent given the target value of the instance, the probability of observing the conjunction is just the product of the probabilities for the individual attributes [15]. Therefore target value output by the Naive Bayes is

\[
v_{NB} = \arg \max_{v_j \in P} \left( P(v_j) \prod_i P(a_i | v_j) \right)
\]

where $v_{NB}$ denotes the target value output by the naive Bayes classifier. The conditional probabilities $P(a_i | v_j)$ need to be estimated from the training set. OneR classifier is a simple, accurate algorithm that generates one rule for each predictor in the data, and then selects the rule with the smallest total error as its “one rule”. Studies have shown that OneR produces rules only slightly less accurate than state-of-the-art classification algorithms. KNN - K nearest neighbors is a simple algorithm that stores all available cases and predict the numerical target based on a similarity measure which finds the training instance closest in Euclidean distance to the given test instance and predicts the same class as this training instance. First one is selected in case of several instances qualify as the same distance. Simple Logistic builds logistic regression models fitting them using LogitBoost with simple regression functions as base learners and determining how much iteration to perform using cross-validation for automatic attribute selection. Details of simple logistic can be found at [14],

VFI (Voting feature intervals) uses voting for classification by discretizing numeric attributes [9], J48 is a JAVA implementation of Quinlan’s C4.5 algorithm [11]. J48 and Random forest are well known algorithm so details of these algorithms are skipped here. The procedure we have used to evaluate the model is 10 fold cross validation for each dataset with various pre-processing and machine learning combination.

Procedure Evaluation data Learning (data, scheme)
Input: data - the data on which the learner is built, KC1, KC3, PC1, PC3, PC4, MW1, MC2, JM1, MC1, PC2, PC5
Learners- The learning scheme. [Naive Bayes (NB), VFI, Simple logistic, KNN, J48, OneR and Random forest]

Output Result [D_AUC]=D_PREDICATOR with D_TEST
AUC over 10 fold cross validation.
Preprocessing={None, PCA, equal-width binning, entropy based discretisation}
FOR i = 1 to 10
FOR EACH data
FOR EACH preprocessing
P_Data = Preprocess data // after preprocessing of data
Divide the Processed data in 90% D_TRAIN and 10% D_TEST
// construct predictor from D_TRAIN
D_PREDICTOR = Train LEARNER with D_TRAIN
//Evaluate predictors on the test data
[D_AUC] =Test D_PREDICATOR with D_TEST
END
END
END

We have used the area under receiver operating characteristic curve (AUC) to evaluate the effectiveness of the fault prediction as it is most informative indicator for prediction accuracy. If AUC is greater than 0.5 then the learner have capability to identifying fault prone modules [5]. The classifiers performance measures are derived from the confusion matrix. Confusion matrix gives the TP, FP and FN and TN. TP and TN are the correctly identified faulty and not faulty modules. A FP occurs when outcome is predicted yes when it is actually not faulty. A FN occurs when the when the outcome is incorrectly predicted as negative when it is actually positive.

Receiver Operating Characteristic (ROC) curves compares the classification performance by a plotting the TP rate on y axis and FP rate on X axis across all the possible experiments.

An ideal ROC curve has a shape which covers the maximum area. ROC curves output by two machine learner are shown in Fig. 1. The ideal point on the ROC curve would when no positive examples are classified incorrectly and negative examples are classified as negative. Continuous measures ROC curves have a problem when two curves intersect each other. As shown in Fig. 1 the curve C1
intersects C2, determination of which model is dominating requires to obtain the area under each curve. Every model gets different values for area under curve. So AUC is used for the evaluation of model performance. AUC is independent of the decision criterion selected and prior probabilities. The AUC comparison can establish a dominance relationship between classifiers [12]. The bigger the area under curve, the better the model is. As opposed to other measures, the area under the ROC curve (AUC) does not depend on the imbalance of the training set [13]. Thus, the comparison of the AUC of two classifiers is fairer and more informative than comparing their misclassification rates.

5. Results

AUC is measured across the 10 fold cross validation for each learning scheme on each data set. Fig. 2 shows the performance of each model for each data set. It can be seen from Table 3 that VFI and Naïve Bayes with discretization outperformed on the average of 11 data sets with AUC 0.736 and 0.735 respectively. This means we should choose Naïve Bayes or VFI learning scheme after discretization for different data sets. If we see the result of individual data set then for KC1 Naïve Bayes with PCA has performed well with area under curve 0.716. PC1 using random forest performed highest with the AUC 0.621. KC3 with AUC 0.79 and PC3 with AUC 0.708 by using Naïve Bayes with discretization has performed better that the other models. On PC4 software design data Random forest with discretization has outperformed with AUC 0.791. MW1 using simple logistics with PCA achieves AUC 0.766. MC2 Simple logistics with PCA AUC is 0.688 and JM1 with Simple logistics after PCA 0.676. MC1 using Simple logistics with discretization AUC is 0.89 and PC2 achieved AUC 0.757 after applying Naïve Bayes. On PC5 data set Simple logistics with discretization achieved highest AUC 0.96 whereas Naïve Bayes using discretization achieved AUC 0.95. This reveals that different attribute selectors can be suitable for different learning algorithms.

It is evident from the results shown in Table 3 that machine learning algorithm applied after the discretization process improves the performance of classifiers Naïve Bayes, VFI, KNN, Simple Logistics and Random forest. Performance of each scheme is shown in Fig. 2 which clearly indicates that Naïve Bayes is consistently performing well for each dataset.

Out of seven five classifiers have reported improved prediction capability in AUC measurement on the average performance of 11 classifiers when used with discretized data. When PCA is used as dimensionality reduction for pre-processing step only Simple logistics and OneR has shown improvement. Maximum AUC for each data set is shown in bold font.

Binning pre-process step has slightly improved the KNN AUC from 0.583 to 0.586 only. Further Naïve Bayes and VFI with discretization on average of performance have outperformed in all 28 classification strategies. To summarize, since Naïve Bayes and VFI learning scheme dominates when applied after discretization, we should choose Naïve Bayes learning or VFI schemes for different data sets.

By the results it is evident that Naïve bayes and VFI with discretization improve the performance of software fault prediction also entropy based discretization helps in improving the performance of other learning algorithms. By the results it can be seen that principal components degrades the performance of Naive Bayes. Also similar findings are reported by other researchers like Hall et al [20].

6. Conclusion

Development of fault free software in budgeted cost is the major concern of software organizations. Prediction of faulty modules prior to coding improves the knowledge for the faulty modules and can be corrected at less cost with less effort. The goal of prediction of fault prone modules at design time using machine learning and pre-processing of data is to minimize the testing cost and to develop reliable software. Using the information at design time the software managers can effectively make changes in module design

Fig. 1. ROC curve of two experiments

Fig. 2. Performance measure using AUC for 28 Schemes
and allocate project resources toward those modules that are fault prone. While developing the software, fault prediction enables developers to fix the faults at design phase itself. In this research, we evaluated and analyzed techniques to predict the fault at design phase. Extensive experiments were performed to explore the impacts of different elements of a learning scheme for fault prediction. From these results, we conclude that a data pre-processor/dimensionality reduction can play different roles with different learning algorithms for different data sets. These results strongly endorse building fault predictors using VFI and Naive Bayes with discretization. Also discretization process has been shown to improve the prediction capability of other learners. In this experiment, each data set was divided into 10 parts and performed 10-fold cross validation. For each pass, we took 90 percent of the data as training data and the remainder as test data. An extensive study conducted with the results of 3,080 AUC curve. Results from experiments have established the fact that design metrics can be used accurately as software fault indicator in early stage of software development.

References


Software Fault Prediction at Design Phase


Pradeep Singh is with the Department of Computer Science Engineering as Assistant Professor at National Institute of Technology, Raipur. He has completed his M.Tech. from Motilal Nehru National Institute of Technology (MNNIT) Allahabad, India with specialization in Software Engineering.

His current research interests include empirical studies on software quality, software fault prediction models, and computational intelligence. He has more than ten years experience in various government academic institutes. He has published over 12 referred articles and served as reviewer of several journals including Knowledge Based System. He is a Member of IEEE, CSI and the ACM.

Shrish Verma is Professor in the department of Electronics & Telecommunication, National Institute of Technology, Raipur. He has completed his Post graduation in Computer Engineering from Indian Institute of Technology, Kharagpur. He has completed his PhD in Engineering from Pt. Ravi Shankar Shukla University Raipur. His area of interest is Image processing, data mining, Software fault prediction models and Software bug classification. He has published over 20 referred articles and served as reviewer of several journals.

O. P Vyas is Professor at Indian Institute of Information Technology Allahabad. Prof. O. P. Vyas has pursued M.Tech. in Computer Science from IIT Kharagpur and Ph.D. in Computer Networks from IIT Kharagpur in joint collaboration with Technical University of Kaiserslautern (Germany). Before joining IIIT-Allahabad, he worked at Pt. R. S. University Raipur, where he successfully founded the School of Computer Science and was instrumental in introducing Computer Education in area of Chhattisgarh region. Prof. O.P. Vyas has been active researcher and had published more than 80 research papers, three books and completed one Indo-German Project under DST-BMBF. He has received DAAD Fellow (Technical University of Kaiserslautern-Germany) and AOTS Fellow (CICC Japan).