CHAPTER 5

FAULT PREDICTION OF OBJECT ORIENTED DESIGN USING A HYBRID ANFIS PREDICTION MODEL

5.1 INTRODUCTION

If under certain environment and situation defects in the application or product get executed then the system will produce the wrong results causing a failure. Not all defects result in failures, some may stay inactive in the code and we may never notice them. Example: Defects in dead code will never result in failures. Thus the software fault prediction becomes the most concerned task which needs to be done for providing the efficient outcome without fault. In this research work, software fault prediction in the Object oriented programming language is concentrated to provide the efficient software outcome.

5.2 BACKGROUND STUDY

According to Laprie et al.(1992) , a system failure occurs when the delivered service no longer complies with the specifications, the latter being an agreed description of the system's expected function and/or service". This definition applies to both hardware and software system failures. Faults or bugs in a hardware or a software component cause errors. An error is defined by Laprie et al.(1992) as that part of the system which is liable to lead to subsequent failure, and an error affecting the service is an indication that a failure occurs or has occurred. If the system comprises of multiple components, errors can lead to a component failure. As various components in
the system interact, failure of one component might introduce one or more faults in another. Figure 5.1 shows this cyclic behavior.

**Figure 5.1  Fault behavior**

It is not just defects that give rise to failure. Failures can also be caused because of the other reasons also like:

Because of the environmental conditions as well like a radiation burst, a strong magnetic field, electronic field or pollution could cause faults in hardware or firmware. Those faults might prevent or change the execution of software.

Failures may also arise because of human error in interacting with the software, perhaps a wrong input value being entered or an output being misinterpreted.

Finally failures may also be caused by someone deliberately trying to cause a failure in the system.

### 5.2.1 Difference Between Error, Defect And Failure In Software Testing

Error: The mistakes made by programmer are known as an ‘Error’. This could happen because of the following reasons:

- Because of some confusion in understanding the functionality of the software
- Because of some miscalculation of the values
- Because of misinterpretation of any value, etc.
Defect: The bugs introduced by programmer inside the code are known as a defect. This can happen because of some program mistakes.

Failure: If under certain circumstances these defects get executed by the tester during the testing then it results into the failure which is known as software failure. Few points that is important to know:

When tester is executing a test he/she may observe some difference in the behavior of the feature or functionality, but this not because of the failure. This may happen because of the wrong test data entered, tester may not be aware of the feature or functionality or because of the bad environment. Because of these reasons incidents are reported. They are known as incident report. The condition or situation which requires further analysis or clarification is known as incident. To deal with the incidents the programmer need to to the analysis that whether this incident has occurred because of the failure or not.

It’s not necessary that defects or bugs introduced in the product are only by the software. To understand it further let’s take an example. A bug or defect can also be introduced by a business analyst. Defects present in the specifications like requirements specification and design specifications can be detected during the reviews. When the defect or bug is caught during the review cannot result into failure because the software has not yet been executed.

These defects or bugs are reported not to blame the developers or any people but to judge the quality of the product. The quality of product is of utmost importance. To gain the confidence of the customers it’s very important to deliver the quality product on time.
5.2.2 Software Fault Prediction

Software fault prediction approaches use previous software metrics and fault data to predict fault-prone modules for the next release of software. If an error is reported during system tests or from field tests, that module’s fault data is marked as 1, otherwise 0. For prediction modeling, software metrics are used as independent variables and fault data is used as the dependent variable. Therefore, we need a version control system such as Subversion to store source code, a change management system such as Bugzilla to record faults, and a tool to collect product metrics from version control system. Parameters of the prediction model are computed by using previous software metrics and fault data. Software fault prediction models have been studied since 1990s until now and fault-prone modules can be identified prior to system tests by using these models. According to recent studies, the probability of detection (PD) (71%) of fault prediction models may be higher than PD of software reviews (60%) if a robust model is built. Benefits of software fault prediction are listed as follows (Catal & Diri, 2009a):

- Reaching a highly dependable system,
- Improving test process by focusing on fault-prone modules,
- Selection of best design from design alternatives using object oriented metrics,
- Identifying refactoring candidates that are predicted as fault prone,
- Improving quality by improving test process

By definition, fault is a structural defect that may eventually lead to deterioration of the systems. Software testing is one of the most critical and costly phases in software development. Defect predictors have been effective secondary tools to help test terms to locate potential defects accurately (A. Porter & R. Selby, 1990). Software defect prediction is the task of classifying
software modules into fault-prone (fp) and non-fault-prone (nfp) ones by means of metric based classification (Briandet al.2002), (Menzies T.& Greenwald J,2007).

Use of software metrics to predict software faults was initiated by Porter and Selby in 1990 (A. Porter& R. Selby,1990), (S. Lessmannet al.2008). Since then, there has been an extensive interest on metric based fault prediction (M. Fagan,1986), (B. Turhanet al.2009). Interestingly, Turhan et al.(S. M. Fakhrahmad,2008) shows that we can build classifiers based on software metrics from a Turkish refrigerator manufacturer and predict faults in NASA software modules of a space shuttle. It has been shown that defect predictors which employ data mining module-based features have over 70% prediction accuracy (F. Shullet al.2002). In comparison to the 60% detection rate of manual software review (based on a panel in IEEE Metrics 2002) (C Catal & B Diri,2009) and considering the ability of an expert reviewer who can merely inspect 8 to 20 LOC/minute, it becomes apparent why automatic defect predictors based on design and code attributes are such an active research area.

Catal and Diri (Catal & Diri,2009) performed a research on studies in the field of software defect prediction. They focused on metrics, methods and datasets used in software defect prediction area. According to their studies, the percentage of using public datasets has greatly increased. Moreover, the use of machine learning algorithms has gradually increased since 2005. They have also investigated the statistics related to published articles in this field. According to their research, published articles in the field of software defect prediction started growing from 1990. Most of the publications are from 2000 onwards. Also according to statistics, in over 60% of researches, Method level metrics has been used. In addition, statistical and machine learning methods have the highest use in this area. Menzies et al.(T.
Menzies et al. (2010) concluded that using data mining techniques can not lead to more accurate detection systems. The goal should be changed therefore they proposed to enhance training of each detection system for a specific use.

One of the challenges in fault detection systems is the metrics through which faults can be detected. Two kinds of metrics are used in these systems: Code level and Design level metrics. However, obtaining design metrics is challenging (e.g., complexity metrics) and not always straightforward. It requires availability of design phase artifacts and design diagrams such as DFDs, control flow graphs, Formal Description Language (FDL) graphs and UML diagrams. In our previous research (M. Chapman et al. 2004) we presented a set of metrics with higher accuracy.

During the past decade, several classification systems have been proposed, which perform predictive modeling efforts for detection of modules that are likely to contain faults. The evaluation of such systems has almost been carried out using a set of datasets available from NASA MDP repository (J. Demšar, 2006). A comparative study like (A. Porter & R. Selby, 1990) provided a baseline and dataset for this research. Each module is described by a set of code-level and design-level attributes. All discovered faults of the system are also registered in each dataset, together with the number of modules containing the fault.

Based on a set of experiments on NASA MDP datasets, Lessman et al. (Lessman et al. 2004) concluded that there is no statistically significant difference between predictive performances of dissimilar classifier. He made his comparisons among classifiers based on state-of-the-art statistical methods designed for comparing different classifiers over different datasets (F. Shull et al. 2002). We will extend this research by studying 37 various classifiers and comparing their performances on five other datasets. By evaluating the
classifiers based on their performance measure, (Accuracy (ACC) and area under curve (AUC) ), we determine the best classification performance.

During the past years, many researchers have attempted to evaluate different methods and several defect prediction systems have been proposed. The results of these systems are given in terms of classification accuracy, precision, performance, etc. However, these factors do not really show the goodness of the model. This paper extends our preceding results that presented a set of low cost metrics for fault detection systems (M. Chapman et al. 2004). In our previous work, by using AD-Miner algorithm (Y. Jiang et al. 2008), we proposed a set of metrics which demonstrate higher performance and accuracy for fault prediction systems. In this work we are going to deploy the accuracy by finding a classifier which performs better than others in fault detection systems. Since our intention is to reduce cost and provide higher performance in fault detection systems, in this paper we will peruse classifiers to find a classifier which obtains higher accuracy in fault prediction systems.

5.3 FAULT PREDICTION USING ANFIS MODEL

Object-Oriented metrics play a crucial role in predicting faults. In literature, prediction models are mostly developed using statistical models. The proposed work aims to establish the relationship between object oriented metrics and fault proneness at class level. Fault has been taken as a function of WMC, NOC, DIT, RFC, CBO and LCOM of the CK metric suite. Neural networks (NN) are an ideal choice for problem definitions involving prediction and classification. The neuro-fuzzy based system approach learns the rules and membership functions from data. It is proven in many proposed techniques that neuro-fuzzy give the better results as compare to standalone FIS or ANN because it uses the power of rules decision of FIS and adaptive nature of ANN in a single system together. In addition the results of this work
show that conceptual relations between classes could also be used as a good metric for prediction of fault proneness. Figure 5.2 depicts a neuro fuzzy modeling for fault prediction.

![Figure 5.2 Fault Prediction using ANFIS model](image)

Adaptive neuro-fuzzy inference system (ANFIS) constructs a fuzzy inference system (FIS) for the given input/output parameters whose membership function parameters are adjusted using either a back propagation algorithm alone or in combination with a least squares type of method. This adjustment allows the Fuzzy systems to learn from the data they are modeling. ANFIS is applicable to the modeling situations where it is difficult to discern what the membership functions should look like simply from looking at data. A 3 layer ANFIS structure is shown below in figure 5.3.
Six CK metrics have been used as the input nodes and prediction rate as the achieved output. The proposed hybrid ANFIS along with the Levinson- Marquardt updation has been used to design a classifier to for detecting fault proneness. The ANN consists of three layers namely input layer, hidden and output layer. An activation function is used in the input layer and an S or sigmoidal function used in the next two layers. The neural network output is specified as

\[ Y_1 = f(W, X) \]

Where X represents the input vector and W represents the weight function. The objective is to reduce the mean squared error by updating the weights. The weights are updated using Levinson Marquardt method (L-M method) during the learning phase. The update equation is given by

**Figure 5.3 Scheme of two layers of ANFIS**
\[ W_{k+1} = W_k - (J_k^T J_k + \mu I)^{-1} J_k e_k \]

Where \( J \) is the Jacobian matrix and \( \mu \) is the combination coefficient. The circular nodes represent nodes that are fixed whereas the square nodes are nodes that have parameters to be learnt. One of the neuro-fuzzy advantages is that it uses a hybrid learning procedure for estimation of the premise and consequent parameters. The algorithm could be summarized as

Step 1: Data collection from repository

Step 2: Normalization and categorization of data set

Step 3: Model design and training phase

Step 4: Weight updation

Assign an initial weight to each metric \( W_i = 0 \)

Determine pair of fault and fault free classes from training set

\( X = \{x_1, x_2, \ldots\}; \ Y = \{y_1, y_2, \ldots\} \) Where X = faulty class and Y = fault free class

Update weight using \( W_{k+1} = W_k - (J_k^T J_k + \mu I)^{-1} J_k e_k \)

Step 5: Validation of results and classification based on threshold

In this process by keeping fixed the premise parameters, it estimates them in a forward pass and then in a backward pass by keeping fixed the consequent parameters the process would be continued.
Figure 5.4  Fuzzy controller for fault detection (WMC, NOC, and DIT)

The grade estimation could be made more accurate by utilizing both neural network and fuzzy logic. The membership function \( \mu_A(X_i) \) corresponds to the input \( x = (x_1, x_2, \ldots x_k) \) and the rule set is given as follows

Rule1: if \( x \) is A1 and \( y \) is B2,

Then \( f_1 = p_1x + q_1y + 1 \)

Rule2: if \( x \) is A2 and \( y \) is B2,

Then \( f_2 = p_2x + q_2y + 2 \)

The implication and rule consequences are given as

\[
f = \frac{w_1f_1 + w_2f_2}{w_1 + w_2}
\]

The performance of the model with respect to the predictions made on the test data set would make the network to perform more accurate that both the other methods. Back-propagation is the most commonly used learning algorithm in order to train multilayer feed forward networks. A training input data set is applied, weights of neurons are fixed and a set of
outputs are produced as response. During the backward pass an error is generated as the difference between the networks actual and desired output. The error is taken through feedback and weights are adjusted to minimize the error through successive iterations. Depending upon the input values of the metric, some rules out of the total 27 rules from the knowledge base gets fired. The Mamdani inference engine is used to determine the degree of membership of firing.

5.4 EXPERIMENTAL RESULTS

The datasets used in this study come from the NASA Metrics Data Program (MDP) data repository named as C3. The C3 software is written in Java programming Language and six CK metrics are taken as input, and output is the fault prediction accuracy rate required for developing the software. The network is trained using Gradient descent method and Levenberg Marquardt method. Since the proposed work deals with computation of C3, the conceptual similarities between the methods in the class are shown in Table 5.1.

<table>
<thead>
<tr>
<th></th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>1</td>
<td>0.23</td>
<td>0.71</td>
<td>0.36</td>
<td>0.41</td>
</tr>
<tr>
<td>M2</td>
<td></td>
<td>1</td>
<td>0.30</td>
<td>0.85</td>
<td>0.57</td>
</tr>
<tr>
<td>M3</td>
<td></td>
<td></td>
<td>1</td>
<td>0.39</td>
<td>0.22</td>
</tr>
<tr>
<td>M4</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0.88</td>
</tr>
<tr>
<td>M5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Similarity of methods in classes represents the conceptual cohesion of classes which measures the degree to which the methods in a class are conceptually related. This is used to determine whether a class represents a
single concept. A class is viewed as a set of methods \{M1, M2, M3, M4, \ldots M_k\}. The methods from the class are connected by set of weighted edges and the weight of the edge is defined as the conceptual similarity measure (CSM). The CSM has been extended to compute the average conceptual similarity of methods of a class (ACSM(c)) determined as

\[
ACSM (c) = \frac{1}{N} \sum_{i=1}^{N} CSM (M_i, M_k)
\]

where a ACSM(c) = 0.5 is considered for pairs of different methods. C3 is an average measure where some methods are closely related while some are not and the average is around 0.5. The above table is used as an indicator to depict the closeness of the methods in classes. To begin with the proportion of the total variation in the dependant variable y based on the regression model is computed. Precision, Correctness and Completeness are used to evaluate the logistic regression model developed above. Precision here is used to evaluate how well the model classifies faulty and non-faulty classes. 965 numbers of classes were used out of which 775 contain zero faults and WMC was found to have the highest number of classes containing the same name. The results of the univariate logistic regression indicate that the model based on C3 is better than any other model except that of LCOM. The data set is normalized using Min–Max normalization and segregated into three categories as training set, validation set and testing set. The Min–Max normalization is defined as

\[
n = \frac{n - \text{min(att)}}{\text{max(att)} - \text{min(att)}}
\]

Where att indicate the attribute value. The output data set is defined as the target data set. After the training process is completed, the model is reapplied to classify the test data. The output values so obtained ranging between 0 and 1 which are indecisive for data classification. A fault is used as a dependent
variable and each of the CK metric is an independent variable. It is intended to develop a function between fault of a class and CK metrics suite. Fault is a function of WMC, NOC, DIT, RFC, CBO and LCOM. Logistic regression is the commonly used statistical technique. Logistic Regression is used as the predictor of the outcome as a function of input variables. It is basically used for construction a predictor for fault proneness of classes. Most common evaluation parameters are sensitivity, specificity, precision, region of convergence. Sensitivity is a measure of how many classes are predicted to be fault prone to the actual number of faulty classes while specificity is the measure of predicted non faultiness to actual number of non-faulty classes. Table 5.2 summarizes the performance measures determined for a set of classes.

**Table 5.2  Statistics of Metrics under consideration**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Performance Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
</tr>
<tr>
<td>WMC (Wighted methods/ Class)</td>
<td>9.34</td>
</tr>
<tr>
<td>NOC (Number of children)</td>
<td>0.197</td>
</tr>
<tr>
<td>DIT (Depth of inheritance)</td>
<td>1.544</td>
</tr>
<tr>
<td>RFC (Response for a class)</td>
<td>22.021</td>
</tr>
<tr>
<td>CBO (Coupling between objects)</td>
<td>7.652</td>
</tr>
<tr>
<td>LCOM (Lack of cohesion in methods)</td>
<td>55.09</td>
</tr>
</tbody>
</table>
The above values denote percentile measures with which correlation between the observed class and other classes could be established. The correlation measure is used to analyse the dependency of the one metric with the other classes. Correlation normally takes a range of 0 – 1 where a value nearer to 1 indicates a good correlation or dependence while the values towards 0 indicate a poor dependence. Table 5.3 illustrates the evaluation metrics for an ANN based and ANFIS based technique.

Table 5.3 Evaluation results of proposed work

<table>
<thead>
<tr>
<th>Evaluation measure</th>
<th>Prediction model</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ANN model</td>
<td>Proposed model</td>
<td></td>
</tr>
<tr>
<td>Sensitivity</td>
<td>67.18</td>
<td>67.91</td>
<td></td>
</tr>
<tr>
<td>Specificity</td>
<td>62.09</td>
<td>63.56</td>
<td></td>
</tr>
<tr>
<td>Precision</td>
<td>62.99</td>
<td>63.79</td>
<td></td>
</tr>
</tbody>
</table>

Region of convergence is a graphical analysis between sensitivity and specificity.

Figure 5.5 Region of convergence (Area under curve = 0.71)
The area under the convergence curve arrives at 0.71 indicating a good prediction accuracy. The above determined results indicate that the classes WMC, CBO, RFC metrics are good predictors of fault proneness and can be used in machine learning based prediction techniques. The proposed approach is able to predict faultiness of a class with more than 90% accuracy.

![Evaluation of parameters](image)

**Figure 5.6  Evaluation of precision and correctness of fault models**

Figures 5.5 and 5.6 illustrate the measured precision, correctness and completeness of various fault models and it could be seen that out of all fault metrics WMC and CBO have a significant effect on fault prediction of classes. Figure 5.7 shows the variation of mean square error with respect to number of epochs required for training the network.
Figure 5.7  Convergence of MSE with iteration number

To predict the best model, we used six machine learning techniques that measured the accuracy in terms of sensitivity, specificity, precision, and AUC (area under the curve). The cut-off point was also selected such that a balance is maintained between the numbers of classes predicted as fault and not faults prone. The ROC curve was used to calculate the cut-off point.
Fault prediction using statistical and machine learning methods were carried out by coding in MATLAB environment. Statistical methods such as linear regression and logistic regression were applied. Also machine learning techniques such as hybrid artificial neural network using gradient descent and Levenberg Marquardt methods were applied for fault prediction analysis. It can be concluded from the statistical regression analysis that out of six CK metrics, WMC appears to be more useful in predicting faults. It is to be noted that the results analyzed from the literature survey involve hybrid classifiers such as neural network – genetic algorithm, neural network – particle swarm optimization have a good classification rate while the proposed work has an advantage of a fast convergence by utilization of the levinson–marquardt updation of coefficients. More similar type of studies can be carried out on different datasets to give generalized results across different organizations.
Figure 5.9  Delay Analysis

Figure 5.9 show that the delay analysis and it is defined as the time taken to transmit the data to the required destination. This delay has to analyze by varying the simulation time from 25 to 200 seconds. Compared to the existing ANN model, proposed NAFIS model achieves lesser delay.
Figure 5.10  Efficiency for various fault models

Figure 5.10 shows that the efficiency for various fault model models and it is defined as the ratio of number of received data to the average number of forwarded data. Her WMC, NOC and DIT fault modes achieves less than 75% of efficiency. CBO and RFC models has achieved between 75% and 85% of efficiency. But the proposed ANFIS fault prediction model has increases the efficiency of 95% compared to the ANN model.

5.5  SUMMARY

It could be seen from the results obtained that the object oriented metrics were validated and the purpose of validation to prove the associativity of the metrics to the fault proneness have been met. However, WMC and ACSM cannot be taken to be standalone parameters for predicting system faults since validation with industrial systems are required to draw stronger conclusions. The severity of faults is not taken into account.