CHAPTER 7

NEURO-FUZZY MODEL FOR QUALITY ASSESSMENT
NEURO-FUZZY MODEL FOR QUALITY ASSESSMENT

7.1 FUZZY LOGIC FOR QUALITY ASSESSMENT

The neural network based defect prediction model predicts the number of defects that are likely to occur in the different phases of the software development life cycle. The ultimate requirement is to assess the quality of the software in terms of risk involved in using that software for the final application. For this, predicting the number of defects in a product alone is not sufficient. From the number of defects predicted a method has to be evolved to derive the quality of the software delivered. It is a well known fact that a defect in the requirements or design is more probable and more severe and hence more difficult to remove. So from experience it is established that if the defects in the requirements phase are more, the quality of the code that is delivered will be poor. Similarly if the errors in the design phase are more, the quality of the realized software will not be very good. This assessment is based on human reasoning developed from experience. Natural choice for quality assessment was fuzzy logic because it is based on approximate reasoning the way human beings process information. Initially the development of a fuzzy inference system was considered for assessing the quality of the software. Though present day computers are not oriented towards processing of fuzzy knowledge and common sense reasoning, they can be programmed to process fuzzy information using fuzzy logic. Fuzzy logic enables us to deal with problems that are too complex or too ill defined to be susceptible to solution by conventional means. Fuzzy systems have found a number of practical applications in identification, control, prediction and diagnosing. Traditionally to develop a fuzzy system, human experts often carry out the generation of IF-THEN rules by expressing their knowledge.
7.2 Fuzzy Inference Systems

Fuzzy Inference system (FIS) are popular computing frameworks based on the concept of fuzzy set theory which have been applied with success in many fields like control, decision support, system identification. Their success is mainly due to their closeness to human perception and reasoning as well as their intuitive handling and simplicity, which are important factors for acceptance and usability of the systems. In general a FIS consists of four modules as shown in Figure 7.2.1

- **Fuzzification Module** transforms the system inputs which are crisp numbers into fuzzy sets. This is done by applying a fuzzification function.
- **Knowledge Base** stores the IF-THEN rules provided by experts.
Inference Engine simulates the human reasoning process by making fuzzy inference on the inputs and IF-THEN rules.

Defuzzification Module transforms the fuzzy set obtained by the inference engine into a crisp value.

7.3 Fuzzy Model Used

Fuzzy inference methods are classified as direct methods and indirect methods. Direct methods such as Mamdani and Sugeno are the two most commonly used (these two methods only differ in how they obtain the outputs). Indirect methods are more complex. We have chosen Mamdani method.

Mamdani rule base is a crisp model of a system, i.e. it takes crisp inputs and gives crisp outputs. It works with the use of user-defined fuzzy rules on user-defined fuzzy variables. The idea behind using a Mamdani rule base to model crisp system behavior is that, the rules for many systems can be easily described by humans in terms of fuzzy variables. Thus we can model a complex non-linear system, with common-sense rules on fuzzy variables.

Operation of the Mamdani rule base can be broken down into four parts

a. Mapping each of the crisp inputs into a fuzzy variable (fuzzification).

b. Determining the output of each rule given its fuzzy attendance.

c. Determining the aggregate output(s) of all the fuzzy rules.

d. Mapping a fuzzy output(s) to crisp output(s) (defuzzification).
7.3.1 Mapping of Inputs

Since the Mamdani rule base models a crisp system, it has crisp inputs and outputs. The rules are given in terms of fuzzy variables. The membership of each fuzzy input variable is evaluated for the given crisp input based on experience, and the resulting value is used in evaluating the rules. In the process of fuzzification of membership function, their range is classified as “small”, “moderate”, “large” and “very large”. The range for small, moderate, large and very large is given in Table 7.3.1.1

<table>
<thead>
<tr>
<th>Error Range</th>
<th>No. of Req Defects</th>
<th>No. of Design Defects</th>
<th>No. of Code Defects</th>
<th>No. of Testing Defects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>&lt;7</td>
<td>&lt;12</td>
<td>&lt;15</td>
<td>&lt;5</td>
</tr>
<tr>
<td>Moderate</td>
<td>5-15</td>
<td>8-20</td>
<td>9-20</td>
<td>3-10</td>
</tr>
<tr>
<td>Large</td>
<td>12-24</td>
<td>16-30</td>
<td>17-33</td>
<td>7-16</td>
</tr>
<tr>
<td>Very Large</td>
<td>&gt;20</td>
<td>&gt;25</td>
<td>&gt;30</td>
<td>&gt;14</td>
</tr>
</tbody>
</table>

A clear definition of the defect membership function in any of the phases was not available. A triangular membership function was assumed in all the 4 phases namely-requirements, design, coding and testing phases.

7.3.2 Fuzzy Rules

Fuzzy rules (about 30) have been given for the fuzzification of input membership function. Input membership functions are combined together using “AND” operator. Some typical rules are tabulated in tables 7.3.2.1 and 7.3.2.2
Neuro-Fuzzy Model for Quality Assessment

Table 7.3.2.1 Fuzzy Rules (typical)

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Design</th>
<th>Code</th>
<th>Test</th>
<th>Software Quality Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>Small</td>
<td>Small</td>
<td>Small</td>
<td>Excellent</td>
</tr>
<tr>
<td>Small</td>
<td>Small</td>
<td>Small</td>
<td>Moderate</td>
<td>Excellent</td>
</tr>
<tr>
<td>Small</td>
<td>Small</td>
<td>Moderate</td>
<td>Large</td>
<td>Good</td>
</tr>
<tr>
<td>Small</td>
<td>Small</td>
<td>Large</td>
<td>Large</td>
<td>Good</td>
</tr>
<tr>
<td>Small</td>
<td>Moderate</td>
<td>Small</td>
<td>Very Large</td>
<td>Fair</td>
</tr>
<tr>
<td>Moderate</td>
<td>Small</td>
<td>Small</td>
<td>Very Large</td>
<td>Fair</td>
</tr>
<tr>
<td>Small</td>
<td>Large</td>
<td>Large</td>
<td>Very Large</td>
<td>Poor</td>
</tr>
<tr>
<td>Very Large</td>
<td>Very Large</td>
<td>Large</td>
<td>Small</td>
<td>Poor</td>
</tr>
</tbody>
</table>

Table 7.3.2.2 Fuzzy Rules (Snapshot)

1. If (RequirementDefect is Small) and (DesignDefect is Small) and (CodeDefect is Small) and (TestDefect is Small) then (SoftwareQ is Excellent) (1)
2. If (RequirementDefect is Small) and (DesignDefect is Small) and (CodeDefect is Moderate) and (TestDefect is Small) then (SoftwareQ is Excellent) (1)
3. If (RequirementDefect is Small) and (DesignDefect is Small) and (CodeDefect is Large) and (TestDefect is Small) then (SoftwareQ is Good) (1)
4. If (RequirementDefect is Small) and (DesignDefect is Small) and (CodeDefect is VeryLarge) and (TestDefect is Small) then (SoftwareQ is Good) (1)
5. If (RequirementDefect is Small) and (DesignDefect is Moderate) and (CodeDefect is Small) and (TestDefect is Small) then (SoftwareQ is Excellent) (1)
6. If (RequirementDefect is Small) and (DesignDefect is Moderate) and (CodeDefect is Moderate) and (TestDefect is Small) then (SoftwareQ is Good) (1)
7. If (RequirementDefect is Small) and (DesignDefect is Moderate) and (CodeDefect is Large) and (TestDefect is Small) then (SoftwareQ is Fair) (1)
8. If (RequirementDefect is Small) and (DesignDefect is Moderate) and (CodeDefect is VeryLarge) and (TestDefect is Small) then (SoftwareQ is Fair) (1)
9. If (RequirementDefect is Small) and (DesignDefect is Large) and (CodeDefect is Small) and (TestDefect is Small) then (SoftwareQ is Good) (1)
10. If (RequirementDefect is Small) and (DesignDefect is Large) and (CodeDefect is Moderate) and (TestDefect is Small) then (SoftwareQ is Good) (1)
11. If (RequirementDefect is Small) and (DesignDefect is Large) and (CodeDefect is VeryLarge) and (TestDefect is Small) then (SoftwareQ is Fair) (1)
12. If (RequirementDefect is Small) and (DesignDefect is VeryLarge) and (CodeDefect is VeryLarge) and (TestDefect is Small) then (SoftwareQ is Poor) (1)
13. If (RequirementDefect is Small) and (DesignDefect is VeryLarge) and (CodeDefect is Large) and (TestDefect is Small) then (SoftwareQ is Fair) (1)
14. If (RequirementDefect is Small) and (DesignDefect is VeryLarge) and (CodeDefect is VeryLarge) and (TestDefect is Small) then (SoftwareQ is Good) (1)
15. If (RequirementDefect is Small) and (DesignDefect is VeryLarge) and (CodeDefect is Large) and (TestDefect is Small) then (SoftwareQ is Fair) (1)
16. If (RequirementDefect is Small) and (DesignDefect is VeryLarge) and (CodeDefect is VeryLarge) and (TestDefect is Small) then (SoftwareQ is Poor) (1)
17. If (RequirementDefect is Small) and (DesignDefect is Small) and (CodeDefect is Small) and (TestDefect is Moderate) then (SoftwareQ is Good) (1)

7.3.3 Evaluating the Rules

Using the membership values of defects, determined during fuzzification, the rules are evaluated according to compositional rule of inference. The result is an output fuzzy set that is some clipped version on the user-specified output fuzzy set. Range of this output fuzzy set depends on the combination of membership functions.
After evaluating the rules, we have fuzzy output defined for each of the rules in the rule base. We then need to combine these fuzzy outputs into a single fuzzy output. Mamdani defines that the output of the rule base should be the maximum of the outputs of each rule. Another point to consider is that some of the rules might be more important than the other rules in determining the system behavior. But here while modeling this system we have not assigned any type of weight to any rule.

7.3.4 De-Fuzzification

Fuzzy logic is a rule-based system written in the form of if-then rules. These rules are stored in the knowledge base of the system. The input to the fuzzy system is a scalar value that is fuzzified. The set of rules is applied to the fuzzified input. The output of each rule is fuzzy. These fuzzy outputs need to be converted into a scalar output quantity so that the nature of the action to be performed can be determined by the system. The process of converting the fuzzy output is called defuzzification. Here for the purpose of defuzzification, “centroid” method is used.

7.3.5 Quality Index Computation

The Quality Index is calculated by using the defects and the fuzzy rules as shown in Figure 7.3.5.1. The inputs in the form of number of defects in each phases are fed into the FIS and the quality index is computed as output. From the quality index computed it is possible to make an assessment about the quality of the software.
Following are the defuzzified output of software quality index. It is given in the form of some typical crisp values which are tabulated in order to get a better understanding of how the quality is affected by the number of errors in each phase of the development life cycle. Table 7.3.5.1 provides the defuzzified crisp output, when output is fuzzified using Gaussian distribution.

Table 7.3.5.1 Crisp output using Gaussian distribution

<table>
<thead>
<tr>
<th>Defects in SRS</th>
<th>Defects in SDD</th>
<th>Defects in Code</th>
<th>Defects in Testing</th>
<th>Software Quality Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0.901</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>9</td>
<td>2</td>
<td>0.899</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>0.82</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>10</td>
<td>4</td>
<td>0.82</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>14</td>
<td>4</td>
<td>0.724</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>12</td>
<td>5</td>
<td>0.618</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td>20</td>
<td>5</td>
<td>0.549</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>22</td>
<td>6</td>
<td>0.549</td>
</tr>
</tbody>
</table>
The software quality is assessed based on the quality index following the criteria given below:

- If the Quality Index is less than 0.5, the software quality is poor
- If the Quality Index is between 0.4 and 0.7, the software quality is fair
- If the Quality Index is between 0.6 and 0.85, the software quality is good
- If the Quality Index is between 0.8 and 1, the software quality is excellent.

### 7.4 Limitation of Fuzzy Inference System

With the neural network predicting the number of defects in each phase, using a fuzzy inference system employing fuzzy IF-THEN rules can model the qualitative aspects of human knowledge and reasoning processes without employing precise quantitative analysis. However, some of the limitations of this approach are:

1. No standard methods exist for transforming human knowledge or experience into the rule base and data base of a fuzzy inference system.
2. There is a need for effective methods for tuning the membership functions so as to minimise the output error measure.

Generally, fuzzy systems work well when experience or introspection can be used to articulate the fuzzy IF-THEN rules underlying the system behaviour and/or the fuzzy sets representing the characteristic of each variable.

In the case of flight software, all the combination of input conditions could not be captured as fuzzy rules. Hence the possibility of using neural network techniques to
generate the fuzzy rules by means of an Adaptive Neuro-Fuzzy Inference System to assess the quality of flight software was explored.

### 7.5 Adaptive Neuro-Fuzzy Inference System

Fuzzy logic and neural networks are complementary technologies. Neural networks extract information from systems to be learned, while fuzzy logic techniques most often use verbal and linguistic information from experts [90]. The integration of neural networks and fuzzy logic into a neuro-fuzzy model reaps the benefits of both neural networks and fuzzy logic systems. The neural network provides the connectionist structure (fault tolerance and distribute representation properties) and learning abilities to the fuzzy logic systems, and the fuzzy logic systems provide the neural networks with a structural framework with the high level fuzzy IF-THEN rule thinking and reasoning. These integrated systems can learn and adapt. In short neuro-fuzzy computing is one which has neural networks that recognize patterns and adapt themselves to cope with changing environments and fuzzy inference systems that incorporate human knowledge and perform inferencing and decision making.

Thus for solving the problem of quality assessment of software from the number of defects from the different phases of software development, a neuro-fuzzy model was adopted. In this model, the neuro-fuzzy system provides the fuzzy system with the kind of automatic tuning methods typical of neural networks but without altering their functionality (e.g. fuzzification, defuzzification, inference engine, and fuzzy logic base). In neuro-fuzzy systems, neural networks are used in augmenting numerical processing
of fuzzy sets, such as membership function elicitation, and realization of mappings between fuzzy sets that is utilised as fuzzy rules. The neural network can utilize a finite number of instances to build a membership function that might represent the fuzzy concept in the quality assessment in a valid manner over a wide range of possibilities. Hence a neural network with a single hidden layer can be trained with a standard back propagation algorithm to represent the unknown membership functions. If we could adopt a method which is like a fuzzy inference system using a back propagation to minimize the error, the performance of the model realised will be better than a FIS system. Adaptive Neuro–Fuzzy Inference System (ANFIS) is a fuzzy inference system implemented in the framework of adaptive networks. Using an input/output data set, the ANFIS method constructs a fuzzy inference system whose membership function parameters are tuned using a back propagation gradient descent and a least square type of method. Thus a neuro- fuzzy model will help in solving the two main problems faced in fuzzy reasoning- lack of a definite method for determining the membership functions and the lack of a learning function for self tuning inference rules.

7.6 ADAPTIVE NETWORKS

An adaptive network is a network structure whose overall input-output behaviour is determined by a collection of modifiable parameters. Specifically the configuration of an adaptive network is composed of a set of nodes connected by directed links, where each node performs a static node function on its incoming signals to generate a single node output and each link specifies the direction of signal flow from one node to another. Each node represents a process unit and the links between nodes specify the
causal relationship between the connected nodes. Usually a node function is a parameterised function with modifiable parameters; by changing these parameters, we change the node function as well as the overall behaviour of the adaptive network.

There are two types of adaptive network- feedforward and recurrent networks. In our study feedforward adaptive network is used. Conceptually a feedforward adaptive network is actually a static mapping between its input and output spaces. This mapping may be either a simple linear relationship or a highly non-linear one depending on the network structure and functionality of each node. We have to construct a network to achieve a desired non-linear mapping that is regulated by a data set consisting of desired input and output pairs of a target system to be modeled. This data set is called the training data set and the procedure we follow in adjusting the parameters to improve the network performance is referred as the learning rules or adaptation algorithms. The difference between the desired output and the network output is called the error measure. The learning rule specifies how these parameters should be changed to minimize a prescribed error measure [63].

### 7.7 ANFIS Architecture and Learning Algorithm

Neural networks and adaptive fuzzy inference systems have the ability to construct models using only target system sample data. ANFIS is a class of adaptive networks that is functionally equivalent to fuzzy inference systems. FIS structure is a network type structure similar to that of a neural network, which maps inputs through input membership functions and associated parameters, and then through output membership functions and associated parameters to outputs.
ANFIS applies two techniques in updating parameters. For premise parameters that define membership functions, ANFIS employs gradient descent to fine-tune them. For consequent parameters that define the coefficients of each output equations, ANFIS uses the least-squares method to identify them. This approach is thus called hybrid learning method since it combines the gradient descent method and the least squares method.

ANFIS modeling process starts by obtaining a data set (input-output data pairs) and dividing it into training and checking data sets. The training data set is used to find the initial premise parameters for the membership functions by equally spacing each of the membership functions. A threshold value for the error between the actual and desired output is determined. The consequent parameters are found using the least-squares method. Then an error for each data pair is found. If this error is larger than the threshold value, update the premise parameters using the gradient decent method. The process is terminated when the error becomes less than the threshold value. Then the checking data set is used to compare the model with actual system. A lower threshold value is used if the model does not represent the system.

In principle, if the size of the available input-output data set is large, then fine tuning of the membership functions is recommended (or even necessary), since human-determined membership functions are seldom optimal in terms of reproducing desired outputs. However, if the data set is too small, then it probably does not contain enough
information about the target system. In this situation, the human-determined membership functions represent important information that might not be reflected in the data set; therefore, the membership functions should be kept fixed throughout the learning process.

If the membership functions are fixed and only the consequent part is adjusted, Sugeno ANFIS can be viewed as a functional-link network [132], where the “enhanced representations” of the input variables are obtained via the membership functions. By updating the membership functions we are actually tuning this enhanced representation for better performance. The initial membership functions and rules for the fuzzy inference system can be designed by employing human expertise about the target system to be modeled. Then ANFIS can refine the fuzzy IF-THEN rules and membership functions to describe the input/output behaviour of a complex system, possible to intuitively setup reasonable membership functions and then employ the ANNs training process to generate a set of fuzzy IF-THEN rules that approximate a desired data set.

7.8 SOFTWARE QUALITY ASSESSMENT USING ANFIS

The data available for assessing the quality of the flight software are the number of defects detected during requirements, design, coding and testing phases. Though a Fuzzy Expert System was attempted to assess the quality of the software based on the defects, it becomes only a local description of the system under consideration.
In case of complicated processes it is difficult for human experts to test all the input-output data to find necessary rules for fuzzy system. To solve this problem and simplify the generation of IF-THEN rules, neural networks have been employed. The combination of neural network and fuzzy system allows an improved quality assessment of the software resident in intelligent system. Thus neuro fuzzy system combines the learning capabilities of neural networks with the linguistic rule interpretation of fuzzy inference system. The synthesis of neuro fuzzy inference systems includes the generation of knowledge base rules that have If-THEN form. In this thesis the assessment of the software quality is made by knowing the present measurement and recent historical data.

### 7.9 Quality Assessment Methodology

#### 7.9.1 Training and Testing the Neuro-Fuzzy Model

There were four inputs identified- number of defects detected in Requirement Phase, Design Phase, Coding phase and Testing phase. The output used for training was the Quality Index obtained from experience. A single hidden layer network was used. The training error was 0.045. From the training data the membership functions were generated for all the four phases. The typical membership function generated for Requirements phase is given in Figure.7.9.1. Similar membership functions are generated for other phases too by the Neuro-Fuzzy model.
A typical rule in a Sugeno fuzzy model has the form

If Input 1 =x and Input 2=y, then Output is z=ax+by+c

The output level $z_i$ of each rule is weighted by the firing strength $w_i$ of the rule. For example, for an AND rule with Input 1 = x and Input 2 = y, the firing strength is

$$w_i = \text{AndMethod} \left(F_1(x), F_2(y)\right)$$

Where $F_{1,2}(.)$ are the membership functions for Inputs 1 and 2.

The final output of the system is the weighted average of all rule outputs, computed as

$$\text{Final Output} = \frac{\sum_{i=1}^{N} w_i z_i}{\sum_{i=1}^{N} w_i}$$

Where $N$ is the number of rules.

When the Quality assessment problem was modelled using ANFIS architecture, 81 rules were generated. The Quality Index is calculated for different input test data. A typical sample is given in Figure 7.9.1.2
Figure 7.9.1.2. Typical Quality Index computed by Neuro-Fuzzy model

From the figure we can see that for Requirements Defect=1, Design Defect=3, Code defect=1, Test defect=1, the Quality Index computed by the model is 0.878. The firing strengths of each the rules for each input is \( w_1=0.1117, w_2=0.2446, w_3=0.2501, w_4=0.274 \) and the firing strength of output is \( w_0=0.2535 \)

The Defuzzified output of Requirements Defect and Design Defect Vs Quality Index is given in Figure 7.9.1.3. From the 3D graph, we can clearly see that when the number of errors in Requirements and Design phase are high, the quality index decreases.

Figure 7.9.1.3. Requirements Defect & Design Defect Vs Quality Index
The Defuzzified output of Requirement defect and Coding defect Vs Quality Index obtained from the Neuro-Fuzzy model is given in Figure 7.9.1.4. Here we can clearly see that as the number of errors in Requirements and Coding phase decrease, the quality index increases.

![Figure 7.9.1.4. Requirements Defect & Code Defect Vs Quality Index](image)

The total error count from output of all BPNs is taken and fed into a quality index calculation model as shown in Figure 7.9.1.5.

The number of defects predicted by the Defect prediction models in different phases is fed into the Neuro-fuzzy model and a quality index is obtained. The average test error in Neuro-Fuzzy model was 0.035153.
The four BPN models will predict the defects in different phases if the environment conditions are correctly captured in the form of input factors identified and fed into the respective models. From the defects predicted it is possible to get a quality index from the neuro-fuzzy quality assessment model.

### 7.9.2 Comparison of Fuzzy Logic and ANFIS

For academic interest the quality index given by the FIS and Neuro-Fuzzy model were compared using the same set of test data. The Quality Index obtained from Fuzzy Logic and Neuro Fuzzy Model were plotted for same test data. The outputs were plotted against total number of defects. From Figure 7.9.2.1 we can see that ANFIS output is more smoother than Fuzzy Logic for the input data sets.
Figure 7.9.2.1 Quality Index from Fuzzy Logic and ANFIS

7.9.3 Effect of perturbing the inputs on Quality Index Computed by Neuro-Fuzzy Model

The input data fed into the Neuro-Fuzzy model was perturbed to study how the Quality Index varied. It was found that within ± 10% variation of inputs, the outputs were within the bounds specified as excellent, good, fair and poor. The input data set were chosen such that the defects in the data set 1 were less and progressively increasing in the data set 9. As expected the Quality Index progressively reduces with the number of defects increasing in the data set as seen in Figure 7.9.3.1.

In the Figure 7.9.3.1 we can also see the effect of perturbing all the defects detected in various phases by ± 10%. It is seen there is a upper bound and a lower bound band within which the Quality Index lies.
Next the number of requirements defect alone was perturbed to ±10% in a data set holding the defects from other phases same as the data set given for testing the ANFIS model i.e., requirements defects alone was varied by +10% and then by -10% in all the datasets, while the design, coding and testing defects retained the earlier value in those data sets. From Fig 7.9.3.2, we can see when the requirements defect was varied by +10%, the quality index comes below the nominal and in the dataset 8 and dataset 9, where the number of defects is more, there is a decrease of 0.1 in the quality index.
Figure 7.9.3.2 Quality Index from Neuro-Fuzzy model when requirements defect alone is perturbed by +10%

Figure 7.9.3.3 Quality Index from Neuro-Fuzzy model when requirements defect alone is perturbed by -10%
When the requirements defect in the datasets were varied by -10%, the quality index is slightly above the nominal curve as seen in Figure 7.9.3.3. The impact is small because the requirements defects in set chosen for testing fall in the low domain. As next phase, the number of design defects alone was perturbed by ±10%, holding the defects from other phases same as the data used for testing, including the number of requirements defects which is restored to the test data set value.

Examine Figure 7.9.3.4 shows that varying the design defect by +10% has significant effect on the Quality Index computed by ANFIS. It varies by 0.1 in many of the data sets.
Figure 7.9.3.5 Quality Index from Neuro-Fuzzy model when design defect alone is perturbed by -10%

Figure 7.9.3.5 also shows that when design defect is perturbed by -10%, the Quality Index increases especially in the boundary limits defined as moderate defects. Similarly the number of code defects alone were perturbed by +10% and -10%, and the effect on Quality Index was studied and the results are plotted in Figure 7.9.3.6 and 7.9.3.7.

From Figure 7.9.3.6 and 7.9.3.7, we observe that varying the code defects by +10% and then by -10% had not much significant impact. Only when the code defects were large, the variation had 0.05 change in the Quality Index. Varying on positive side reduced the quality index and perturbing on negative side, increased the Quality Index as expected.
Figure 7.9.3.6 Quality Index from Neuro-Fuzzy model when code defect alone is perturbed by +10%.

Figure 7.9.3.7 Quality Index from Neuro-Fuzzy model when code defect alone is perturbed by -10%.
Similarly the number of testing phase defects alone were perturbed by +10% and -10%, and the effect on Quality Index was studied.

**Figure 7.9.3.8** Quality Index from Neuro-Fuzzy model when test defect alone is perturbed by +10%

![Graph showing Quality Index from Neuro-Fuzzy model when test defect alone is perturbed by +10%](image)

**Figure 7.9.3.9** Quality Index from Neuro-Fuzzy model when test defect alone is perturbed by -10%

![Graph showing Quality Index from Neuro-Fuzzy model when test defect alone is perturbed by -10%](image)
From figure 7.9.3.8 and 7.9.3.9 we can see that noticeable changes in Quality Index occurred only when the number of defects in the testing phase were large as in data set 8 and data set 9. But the change is of the order of 0.02 in the Quality Index.

The quality index when one parameter alone was varied was observed to lie within the bounds obtained for all parameter variation by ±10% given in Figure 7.9.3.1. This is very clear from the Figures 7.9.3.2 to 7.9.3.9 which show that when the number of requirements defects alone is varied, or number of design defects alone is varied, or number of code defects alone is varied and number of test defects alone varied by ±10%, makes the Quality Index lie within the bound obtained when all parameters are varied simultaneously by ±10%.

From this we can infer that variation in the defects in any of the phases gave a quality index within the range defined by the bounds obtained for ±10% variation. Also from studying the effect of perturbation of inputs to the neuro-fuzzy model, as expected variation in the number of requirements defects and design defects have a greater impact on the Quality index than variation in the coding defects or testing defects. This is in line with the theory that a defect in requirements or design has a significant impact on the cost and schedule. A defect in the code or test had lesser impact in the quality index and thus on the cost and schedule.
7.9.4 Categorization of software based on Quality Index.

Based on the crisp software quality index obtained, which is an indication as to whether the software is excellent, good, fair or poor, the software can be classified under following category:

- **A < Quality Index < B; Quality is “Excellent”. Using the software has no risk.**
- **C < Quality Index < D; Quality is “Good”. Using the software has low risk.**
- **E < Quality Index < F; Quality is “Fair” Using the software has moderate risk.**
- **G < Quality Index < H; Quality is “Poor”. Using the software has high risk.**

For this research the values of limits are chosen as follows:

- **A = 0.8, B = 1.0, C = 0.6, D = 0.85, E = 0.40, F = 0.7, G = 0, H = 0.5,**

The software quality index computed is compared against defined thresholds and assessed as high risk, moderate risk, low risk and no risk. In fixing the lower bound and upper bound, the number of errors that can be tolerated for the specific application has to be considered. It is fixed based on data obtained from the specific application area.

From the quality index an initial assessment on the quality of the software is obtained. If it is within the required value, no further refinement on the quality attributes is required. If it is not within the expected range, managers can do a study of the input factors that had gone into the defect prediction model and from the analysis of the input factors, it is possible to identify which area – complexity of problem, designer’s expertise or process
adopted, has to be refined or improved. For quality/test engineers, an identification of the areas where inspection or testing has to be tightened will be possible from the analysis of the defects – whether SRS or SDD reviews has to be strengthened or efforts have to be concentrated on code inspection or testing.