CHAPTER 4

FRAMEWORK IMPLEMENTATION

4.1 BACKGROUND

Development of reliable software is challenging as system engineers have to deal with a large number of conflicting requirements such as cost, time, reliability, safety, maintainability, and many more. These days, most of the software development tasks are performed in labor-intensive way (Pandey and Goyal, 2013). This may introduce various faults across the development, causing failures in the near future. The impact of these failures ranges from marginal to catastrophic consequences. Therefore, there is a growing need to ensure the reliability of these software systems as early as possible.

IEEE defines software reliability as “the probability of a software system or component to perform its intended function under the specified operating conditions over the specified period of time”. In another way, this can also be defined as “the probability of failure-free software operation for a specified period of time in a specified environment” (Kumar, 2009). Software reliability is generally accepted as the key factor of software quality since it quantifies software failures, which make the system inoperative or risky.
4.2 PROPOSED RELIABILITY MODEL

Software reliability has most of its roots in the requirements and design (Lyu, 2007) and can be improved by inspection and review of these steps. Generally, software reliability can be estimated or predicted using various available software reliability models. These models use failure data collected during testing, and reliability can be estimated or predicted from a fitted model. This becomes too late and sometimes infeasible for taking corrective actions (Mohanta et. al, 2011). The solution to this problem is to predict the software reliability in the early stage of development process, that is, before the coding starts. This early reliability information can help in project management in reducing the development cost by reducing the amount of the rework.

Figure 4.1 Early Stage Reliability Prediction Model (ESRPM)
Moreover, since the failure data are not available during the early phases of software life cycle, the information such as product-level software metrics, expert opinions, and similar or earlier project data can be used for characterizing the factors affecting software reliability. Early fault prediction provides an opportunity for the early identification of software quality, cost overrun, and optimal development strategies (Rizvi et al., 2016a; Yadav and Yadav, 2015).

This chapter has focused on the identification of reliability-relevant software metrics for early fault prediction. For this, a comprehensive model has been proposed to gather reliability-relevant software metrics from the early phase of software development life cycle, processing it, and integrating it with the fuzzy logic system to predict the design level reliability before coding.

The model is referred as Early Stage Reliability Prediction Model (ESRPM). The model is based on the assumption that the reliability and quality of a software system are adversely affected by the weaknesses of requirements and design constructs. Therefore the model focuses on these two most significant phases of SDLC, for reliability prediction. The model architecture is shown in Fig. 4.1.

4.3 IMPLEMENTING IDENTIFICATION PHASE

As discussed in the previous chapter that the objective of the identification phase is to identify the factors that are related directly or indirectly to the problem as well as its solution. There is no doubt, that quantified reliability will not have significant value if its underlying factors are not identified appropriately.
4.3.1 Identify Reliability Factors

In the current reliability framework the researcher has followed the methodology suggested by ISO 9126 and Dromey (1995), that is in order to study any higher level quality attribute, it should be decomposed into lower level attributes. Therefore in order to quantify the reliability, its lower level attributes have been searched in the literature. After a thorough review of the state-of-the-art, researcher has come up with the following studies highlighting a variety of factors impacting the software reliability positively as well as negatively.

Figure 4.2 Reliability Factors

McCall’s (1977) had suggested that Software Consistency, Accuracy and Fault Tolerance effects the software reliability. While, Recoverability, Maturity and Fault Tolerance are suggested by Dromey as reliability factors (Dromey, 1995). In

4.3.2 Select one or more key Factors

This is the step in the identification phase of the framework, that provides flexibility to the researcher or industry professional. Because quantifying the software reliability by considering all the underlying factors simultaneously is impractical. Therefore the framework suggests concentrating on a few depending on the perspective of the study. Individually reviewing all the twelve factors identified in the previous section, it could be easily inferred that complexity is such a factor that may affect these factors directly or indirectly (Cardoso, 2006; Fiondella et. al, 2013; Koh, 2008; Linda and Brennan, 2006; Yadav and Khan, 2009a). Therefore, the researcher has concentrated on complexity as a key attribute for the reliability of the software application.

4.3.3 Identify Requirements Level Metrics

A number of software reliability prediction models using software metrics have been proposed in last two decades. Almost all existing reliability or defect prediction models has considered a considerable number of software metrics such as traditional software metrics, object oriented software metrics, process metrics.
However, predicting software reliability by taking all the software metrics (traditional, object oriented and process metrics) have following drawbacks: computationally complex, more expensive processing cost, there are many less important software metrics, correlations among the software metrics, increase time complexity. Therefore right selection of software metric could improve the prediction accuracy. However, it is essential to consider the metrics which are most important from reliability point of view. Therefore the researcher has gathered following software requirements metrics from various available sources (He, 2015; Li and Smidts, 2003; Martin, 2000; Radjenovic, 2013).

Table 4.1 Requirements Stage Metrics

<table>
<thead>
<tr>
<th>S.No</th>
<th>Requirements Metrics</th>
<th>S.No</th>
<th>Requirements Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RS (Requirements Stability)</td>
<td>7</td>
<td>PM (Process Maturity)</td>
</tr>
<tr>
<td>2</td>
<td>RSDR (Regularity of Specification and Docu. Reviews)</td>
<td>8</td>
<td>RIW (Review Inspection and Walkthrough)</td>
</tr>
<tr>
<td>3</td>
<td>RFD (Requirement Defect Density)</td>
<td>9</td>
<td>RCR (Requirement Change Request)</td>
</tr>
<tr>
<td>4</td>
<td>Scale of New Functionality Implemented</td>
<td>10</td>
<td>ERT (Experience of Requirement Team)</td>
</tr>
<tr>
<td>5</td>
<td>RC (Complexity of New Functionality)</td>
<td>11</td>
<td>RM (Requirements Management)</td>
</tr>
<tr>
<td>6</td>
<td>QDI (Quality of Documentation Inspected)</td>
<td>12</td>
<td>DSM (Development Staff Motivation)</td>
</tr>
</tbody>
</table>
4.3.4 Identify Design Stage Metrics

As the framework concentrates on four object-oriented design constructs therefore the researcher has gathered following object-oriented design metrics from various available sources (Andersson and Vestergren, 2004; Bansiya, J. and Devis, C., 1997; Bansiya and Devis, 2002; Birkmeier, 2010; Breesam, 2007; Dallal J. A., 2010; Gray, 2008; Yadav and Khan, 2012b; Yadav and Khan, 2011; Yong and Qingxin, 2008).

Table 4.2 Object-Oriented Design Metrics

<table>
<thead>
<tr>
<th>S.No</th>
<th>Object-Oriented Design Metrics</th>
<th>S.No</th>
<th>Object-Oriented Design Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LCOM (Lack of Cohesion in Methods)</td>
<td>8</td>
<td>MPC (Message Pass Coupling)</td>
</tr>
<tr>
<td>2</td>
<td>IMc (Inheritance Metric Complexity Perspective)</td>
<td>9</td>
<td>DIT (Depth of Inheritance):</td>
</tr>
<tr>
<td>3</td>
<td>NOC (Number of Children)</td>
<td>10</td>
<td>EMc (Encapsulation Metric Complexity Perspective)</td>
</tr>
<tr>
<td>4</td>
<td>CoMc (Cohesion Metric Complexity Perspective)</td>
<td>11</td>
<td>CMc (Coupling Metric Complexity Perspective)</td>
</tr>
<tr>
<td>5</td>
<td>WMC (Weighted Method per Class)</td>
<td>12</td>
<td>CBO (Coupling Between Objects)</td>
</tr>
<tr>
<td>6</td>
<td>Response for a Class (RFC)</td>
<td>13</td>
<td>DAC (Data Abstraction Coupling)</td>
</tr>
<tr>
<td>7</td>
<td>AHF (Attribute Hiding Factor)</td>
<td>14</td>
<td>AIF (Attribute Inheritance Factor)</td>
</tr>
</tbody>
</table>
4.4 IMPLEMENTING ASSOCIATION PHASE

As discussed in the previous chapter that the aim of this phase in the proposed framework is to align all the components together by justifying their role in the framework. Further this phase rationalizes the association among the various artifact identified in the previous phase.

4.4.1 Correlate Requirements and Design Metrics with Key Factor

In this step of the framework the researcher has shortlisted eight metrics out of twenty six (Requirements (12) and Design (14)) metrics identified in the previous identification phase. Out of these eight metrics four belongs to requirement phase (RS, RIW, RC and RFD) and four belongs to design phase (IMc, CMc, EMc and CoMc). Now this step will ensure how they are correlated with the selected key factor (Complexity) ?

The four requirements metrics are RS, RIW, RC and RFD. RS (Requirements Stability) is inversely proportional to requirement change request and measures the stability of the requirements, provides insight into system’s reliability and quality. Requirement stability is a metric used to organize, control, and track changes to the originally specified requirements. RS provides an indication of the completeness, stability, and understanding of the requirements. Higher frequency of change requests inculcates the complexity in the requirements specifications. So it could be easily inferred that
More Change Requests => More Complex Requirements => Less Reliable
Low Requirements Stability => High Complexity => Less Reliability

Similarly RIW (Review, Inspections and Walkthrough) is an effective means for uncovering requirements errors and improving software quality. If the RIWs are more, there is a lower probability that the software will contain faults, means have higher reliability of the product.

High RIW => Detect/Remove More Defects => SRS Complexity will Reduce => Reliability Improves

Third metric RFD (Requirements Fault Density) measures the fraction of faulty requirements specification documents. Requirement fault density provides an indicator of the software quality of developing software during requirement analysis phase.

High Fault Density => High Requirements Complexity => Low Reliability

Fourth Metric RC (Complexity of New Functionality) notice or rate the complexity of the newly added functions in the SRS. Therefore it is quiet easy to correlate this requirement metric with the key factor i.e. complexity.

High value of RC => Make the SRS Complex => Lower Reliability

Therefore in summarized form it can be shown how the four requirements metrics are associated with the key factor complexity.
\[ RS \alpha \ Reliability \]
\[ RC \alpha \ 1/Reliability \]
\[ RFD \alpha \ 1/Reliability \]
\[ RIW \alpha \ Reliability \]

After justifying the relationship of requirements metrics with complexity, the relationships of complexity with the identified design metrics are as follows.

\[ IMc \alpha \ 1/Reliability \]
\[ CMc \alpha \ 1/Reliability \]
\[ CoMc \alpha \ Reliability \]
\[ EMc \alpha \ Reliability \]

4.4.2 Correlate key Factor with Reliability

It has been noticed during the literature survey that high complexity in requirements specification as well as design give rise to the occurrences of requirements and design errors (Fiondella et. al, 2013; Koh, 2008; Linda and Brennan, 2006). Subsequently these faults negatively impacts (if not taken care of) on the operational life of the application and push them towards unreliability. As it could be noticed, that the complex softwares have always been developed by experienced and reputed software organizations, because they have professional that can handle the trade-off between complexity and reliability. Growing complexity of the software leads to more chance of errors, faults and subsequent failures. This increases the unreliability of the software. Nobody can negate this hypothesis by saying any of the following statement.
“to make the software more reliable make it as complex as possible”

“if the software is less complex it will be less reliable ”

“if the software is more complex it will be more reliable”

On the basis of above it can be concluded that software complexity is not directly proportional to the software reliability, but it is inversely proportional.

4.4.3 Finalize the Metrics Set

After rationalizing the association of identified requirements and design metrics with the key factor complexity in the above two sections, the final step of this phase is to freeze the following metric set for the quantification phase of the framework.

Finalize Metric Set { Requirement Phase metrics (RS, RIW, RC and RFD) and Design Phase metrics (IMc, CMc, EMc and CoMc)}

4.5 IMPLEMENTING QUANTIFICATION PHASE

It is the most critical phase of the framework, because the actual development of the reliability prediction model takes place in this phase itself. The model is implemented in MATLAB utilizing fuzzy logic toolbox. The basic steps of the model are selection of reliability-relevant software metrics (input/ output variables), development of fuzzy profile of these input/output variables, developing the fuzzy rule base and reliability fault prediction at the end of requirements and design phase using fuzzy inference system (FIS).
4.5.1 Select Input and Output Variables

As already discussed in the identification phase that out of total eight metrics four (RS, RIW, RC, RFD) have been selected for the requirements phase and rest four (EMc, CoMc, CMc, and IMc) for the design phase. These metrics (shown in Table 4.3) are considered as input variables for the fuzzy based reliability prediction model and can be applied to the requirement and design phases. Apart from that, two output variables RLR and DLR are also taken as the output for the model. RLR and DLR represent the level of reliability at the end of requirements and design phases, respectively.

Table 4.3 Input and Output Variables

<table>
<thead>
<tr>
<th>Phase</th>
<th>Input Variable</th>
<th>Output Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>RS, RIW, RC, RFD</td>
<td>RLR</td>
</tr>
<tr>
<td>Design</td>
<td>RLR, EMc, CoMc, CMc, IMc</td>
<td>DLR</td>
</tr>
</tbody>
</table>

*Requirement Stability (RS):*

Requirement stability is inversely proportional to requirement change request. There are many way of requirement gathering. The main problem with requirement gathering is that stakeholders (user, customer, developer, and project manager) are not very clear to their requirements. Requirement change may happen at any time during a software project development (Li and Smidts, 2003). Requirements changes are of two types: controlled and uncontrolled. Controlled requirements changes may take place to provide enhancement of the features of the software system or changing customer needs, and other factors. These requirements changes may be
essential for the adaptation of a system to the changes that occur either in hardware or software. Uncontrolled requirements changes may lead to adverse effect on cost, quality and reliability, schedule of the project under development. Studies have exposed that more than half the errors that occur during software development due to imprecisely defined requirements.

**Requirement Fault Density (RFD):**

This metrics measures the fraction of faulty requirements specification documents. Requirement fault density provides an indicator of the software quality of developing software during requirement analysis phase. It is side effect of requirement engineering. Requirement fault density may range from simple fault to complex fault that may impact a large segment of the description (Li and Smidts, 2003).

**Review, Inspection and Walkthrough (RIW):**

This metrics purify the software product and can be applied at various points during software project development. Software developer and the customer both are actively participate for review of software requirements specification (SRS) document. The review is conducted at various levels. The goal of review process is to ensure that the SRS is feasible, complete, consistent and accurate. From quality point of view, it is very important metric (Li and Smidts, 2003).
**Requirements Complexity (RC):**

At the requirement phase, various requirements are collected and expressed into a common language understandable to both the project team and the stakeholders. The main problem with requirement gathering is that stakeholders are not very clear to their requirements and may have variety of requirements with conflicting goals (Li and Smidts, 2003). Therefore, a requirement complexity measure is used that will take the natural language requirements as input and associate a value of requirement complexity such as low, medium, or high. If the requirement complexity is more, there is a greater probability that the software will contain faults. RC value is measured qualitatively by conducting questionnaires with various domain experts, software professionals, and researchers.

**Inheritance Metrics (IMc):**

Inheritance metric (complexity perspective) provides overall complexity of a design hierarchy through inherited methods and attributes and is estimated by taking the average of (IMc-Class) Inheritance metric complexity perspective of every class. (Yadav and Khan, 2009b; 2012a)

\[ IMc = \text{Avg}\{ IMc_{Class} \} \]

Where IMc-Class=(I_{m&a}_{Class})/( T_{m&a}_{Class})

\( I_{m&a}_{Class} \) is total number of inherited methods and attributes in a class from its base class. And \( T_{m&a}_{Class} \) is total number of methods and attributes in a class.
**Encapsulation Metric (EMc):**

Encapsulation metric (complexity perspective) provides overall complexity of a design hierarchy through encapsulated methods and attributes and is estimated by taking the average of (EMc_Class) Encapsulation metric complexity perspective of every class, (Yadav and Khan, 2012b).

\[
EMc = \text{Avg} \{ EMc\_Class \}
\]

Where \( EMc\_Class = \frac{E_{m&a\_Class}}{T_{m&a\_Class}} \)

- \( E_{m&a\_Class} \) is total number of encapsulated methods and attributes in a class.
- \( T_{m&a\_Class} \) is total number of methods and attributes in a class.

**Cohesion Metric (CoMc):**

Cohesion metric complexity perspective is defined as the average of (CoMc_Class) Cohesion metric complexity perspective per class. (Birkmeier, 2010; Yadav and Khan, 2011; Yadav and Khan, 2012a)

\[
CoMc = \text{Avg} \{ CoMc\_Class \}
\]

Where \( CoMc\_Class = \text{Avg} \{ LCc\_Class \text{ and } HCc\_Class \} \)

- \( LCc\_Class = \frac{\text{DUDM}}{\text{NDIC}} \)
- \( HCc\_Class = \sum \frac{\text{DUDM, IDUM}}{\text{NDIC}} \)

DUDM is the number of directly used data members by methods in a class.

IDUM is the number of indirectly used data members by methods in a class.

NDIC is the number of directly/Indirectly used data members by methods in a class.
4.5.2 Develop Fuzzy Profiles

This is the first step in incorporating human knowledge into engineering systems in a systematic and efficient manner. The data, which may be useful for selecting appropriate linguistic variable, are generally available in one or more forms of (Kumar and Mishra, 2008):

- Expert’s opinion.
- Software requirements.
- User’s expectations.
- Record of existing field data from previous release or similar system.
- Marketing professional’s opinion.
- Data obtained from system logs, etc.

Input/output variables selected at the previous steps are fuzzy in nature and are characterized by membership function. Membership function can be generated either with the help of domain expert or real data. There are no standard guidelines or rules that can be used for the appropriate membership function construction technique. Developing a membership function with help of domain expert knowledge is one of the basic steps in the design of a problem which is to be solved by fuzzy set theory. In this research, membership functions of all the input and output metrics are defined with the help of domain experts. Membership function can have a variety of shapes like polygonal, trapezoidal, triangular, and so on (Ross, 2010). In this research triangular membership functions are considered for fuzzy profile development of various identified input/output variables. Triangular membership functions (TMFs) are widely used for calculating and interpreting reliability data because they are simple and easy to
understand (Yadav et al. 2003). Also, they have the advantage of simplicity and are commonly used in reliability analysis (Ross 2005). Its shape provides convenient representation of domain expert knowledge and it also simplifies the computation.

Table 4.4 Fuzzy Profiles for Requirements Stage Measures

<table>
<thead>
<tr>
<th>Value</th>
<th>RC (0-1)</th>
<th>RS (0-1)</th>
<th>RFD (0-1)</th>
<th>RIW (0-1)</th>
<th>RLR (0-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very low</td>
<td>(0;0;0.35)</td>
<td>(0;0;0.35)</td>
<td>(0;0;0.4)</td>
<td>(0;0;0.4)</td>
<td>(0.25;0.4;0.55)</td>
</tr>
<tr>
<td>Low</td>
<td>(0.25;0.4;0.65)</td>
<td>(0.25;0.45;0.75)</td>
<td>(0.2;0.4;0.7)</td>
<td>(0.2;0.4;0.6)</td>
<td>(0.45;0.6;0.85)</td>
</tr>
<tr>
<td>Medium</td>
<td>(0.5;1;1)</td>
<td>(0.6;1;1)</td>
<td>(0.5;1;1)</td>
<td>(0.4;1;1)</td>
<td>(0.65;0.8;0.95)</td>
</tr>
<tr>
<td>High</td>
<td>(0.85;1;1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Very high</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.5 Fuzzy Profiles for Design Stage Measures

<table>
<thead>
<tr>
<th>Value</th>
<th>RLR (0-1)</th>
<th>IMc (0-1)</th>
<th>EMc (0-1)</th>
<th>CMc (0-1)</th>
<th>CoMc (0-1)</th>
<th>DLR (0-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very low</td>
<td>(0;0;0.35)</td>
<td>(0;0;0.35)</td>
<td>(0;0;0.35)</td>
<td>(0;0;0.4)</td>
<td>(0;0;0.4)</td>
<td>(0;0;0.3)</td>
</tr>
<tr>
<td>Low</td>
<td>(0.25;0.4;0.55)</td>
<td>(0;0;0.4)</td>
<td>(0;0;0.4)</td>
<td>(0;0;0.4)</td>
<td>(0;0;0.4)</td>
<td>(0.2;0.35;0.5)</td>
</tr>
<tr>
<td>Medium</td>
<td>(0.45;0.6;0.85)</td>
<td>(0.3;0.5;0.7)</td>
<td>(0.25;0.45;0.75)</td>
<td>(0.25;0.5;0.75)</td>
<td>(0.3;0.5;0.75)</td>
<td>(0.4;0.55;0.7)</td>
</tr>
<tr>
<td>High</td>
<td>(0.65;0.8;0.95)</td>
<td>(0.65;1;1)</td>
<td>(0.65;1;1)</td>
<td>(0.65;1;1)</td>
<td>(0.65;1;1)</td>
<td>(0.6;0.75;0.9)</td>
</tr>
<tr>
<td>Very high</td>
<td>(0.85;1;1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(0.8;1;1)</td>
</tr>
</tbody>
</table>

Table 4.4 and 4.5 lists the selected input/output variables along with their fuzzy range as well as profile. Fuzzy membership functions are generated utilizing the linguistic categories such as very low (VL), low (L), medium (M), high (H), and very high (VH), identified by a human expert to express his/her assessment. Fuzzy profiles of all the various considered input/output variables are shown in Figs. 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 4.10, 4.11 and 4.12 for visualization purpose.
Figure: 4.3 Membership Functions for Requirements Stability

Figure: 4.4 Membership Functions for Review Inspection and Walkthrough
Figure: 4.5 Membership Functions for Requirements Complexity

Figure: 4.6 Membership Functions for Requirements Fault Density
Figure: 4.7 Membership Functions for Requirements Level Reliability

Figure: 4.8 Membership Functions for Encapsulation
Figure: 4.9 Membership Functions for Cohesion

Figure: 4.10 Membership Functions for Inheritance
Figure: 4.11 Membership Functions for Coupling

Figure: 4.12 Membership Functions for Design Level Reliability
4.5.3 Develop Fuzzy Rule Base

In this step fuzzy rules are defined in the form of IF-THEN conditional statement. IF part of the rule is known as antecedent, and THEN part is consequent (Ross, 2010; Zadeh, 1989). The fuzzy rule base can be designed from different sources such as domain experts, historical data analysis, and knowledge engineering from existing literature (Li and Smidts, 2003; Zhang and Pham, 2000). In this research the fuzzy rules that are required for the prediction of the reliability are defined with the help of domain expert.

In our model, in the requirements phase there are four input metrics. Each input metric has three linguistic states i.e., low (L), medium (M) and high (H). Therefore, total number of rules is 81. Similarly in design phase the number of input variables are five four has three linguistic states (i.e., low (L), medium (M) and high (H)), while one input variable has five states (i.e., Very Low (VL), Low (L), Medium (M) High (H) and Very High (VH)), therefore total number of rules is 405.

The rules are generated from the software engineering point of view and all of them take the form of “If A then B.” Considering all the selected Input/output variables simultaneously may result into a large number of rules. Therefore, to reduce the number of rules the researcher has developed two set of rules corresponding to the requirements and design phase. Tables 4.6 and 4.7 and the figure 4.13 and 4.14 present the fuzzy rules (for the requirements and design phase) used in the Fuzzy Inference System for reliability prediction.
### Table 4.6 Fuzzy Rules for Requirement Phase

<table>
<thead>
<tr>
<th>Rule</th>
<th>RS</th>
<th>RIW</th>
<th>RC</th>
<th>RFD</th>
<th>RLR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LOW</td>
<td>LOW</td>
<td>LOW</td>
<td>LOW</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>2</td>
<td>LOW</td>
<td>LOW</td>
<td>LOW</td>
<td>MEDIUM</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>3</td>
<td>LOW</td>
<td>LOW</td>
<td>LOW</td>
<td>HIGH</td>
<td>LOW</td>
</tr>
<tr>
<td>4</td>
<td>LOW</td>
<td>LOW</td>
<td>MEDIUM</td>
<td>LOW</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>5</td>
<td>LOW</td>
<td>LOW</td>
<td>MEDIUM</td>
<td>MEDIUM</td>
<td>LOW</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>35</td>
<td>MEDIUM</td>
<td>LOW</td>
<td>HIGH</td>
<td>MEDIUM</td>
<td>LOW</td>
</tr>
<tr>
<td>36</td>
<td>MEDIUM</td>
<td>LOW</td>
<td>HIGH</td>
<td>HIGH</td>
<td>VERY LOW</td>
</tr>
<tr>
<td>37</td>
<td>MEDIUM</td>
<td>MEDIUM</td>
<td>LOW</td>
<td>LOW</td>
<td>HIGH</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>77</td>
<td>HIGH</td>
<td>HIGH</td>
<td>MEDIUM</td>
<td>MEDIUM</td>
<td>HIGH</td>
</tr>
<tr>
<td>78</td>
<td>HIGH</td>
<td>HIGH</td>
<td>MEDIUM</td>
<td>HIGH</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>79</td>
<td>HIGH</td>
<td>HIGH</td>
<td>HIGH</td>
<td>LOW</td>
<td>HIGH</td>
</tr>
<tr>
<td>80</td>
<td>HIGH</td>
<td>HIGH</td>
<td>HIGH</td>
<td>MEDIUM</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>81</td>
<td>HIGH</td>
<td>HIGH</td>
<td>HIGH</td>
<td>HIGH</td>
<td>MEDIUM</td>
</tr>
</tbody>
</table>

### Table 4.7 Fuzzy Rules for Design Phase

<table>
<thead>
<tr>
<th>Rule</th>
<th>RLR</th>
<th>EMc</th>
<th>CoMc</th>
<th>IMc</th>
<th>CMc</th>
<th>DLR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VERY LOW</td>
<td>LOW</td>
<td>LOW</td>
<td>LOW</td>
<td>LOW</td>
<td>LOW</td>
</tr>
<tr>
<td>2</td>
<td>VERY LOW</td>
<td>LOW</td>
<td>LOW</td>
<td>LOW</td>
<td>MEDIUM</td>
<td>VERY LOW</td>
</tr>
<tr>
<td>3</td>
<td>VERY LOW</td>
<td>LOW</td>
<td>LOW</td>
<td>LOW</td>
<td>HIGH</td>
<td>VERY LOW</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>99</td>
<td>LOW</td>
<td>LOW</td>
<td>MEDIUM</td>
<td>HIGH</td>
<td>HIGH</td>
<td>VERY LOW</td>
</tr>
<tr>
<td>100</td>
<td>LOW</td>
<td>LOW</td>
<td>HIGH</td>
<td>LOW</td>
<td>LOW</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>101</td>
<td>LOW</td>
<td>LOW</td>
<td>HIGH</td>
<td>LOW</td>
<td>MEDIUM</td>
<td>LOW</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
### 4.5.4 Perform Fuzzification

In this phase, fuzzy inference engine evaluates and combines the result of each fuzzy rule. It maps all the inputs to an output. This process of mapping inputs onto output is known as fuzzy inference process or fuzzy reasoning (Bowles and Pelaez 1995; Zadeh 1989). Basis for this mapping is the number of fuzzy IF-THEN rules, each of which describes the local behavior of the mapping. The two main activities for information processing are as follows: combining input from all the “if” part fuzzy rules and aggregation of “then” part to produce the final output. The Mamdani fuzzy inference system (Mamdani, 1977) is considered here for all the information processing.
Figure: 4.13 Fuzzy Rules for Requirements Phase

Figure: 4.14 Fuzzy Rules for Design Phase
4.5.5 Perform Defuzzification

Defuzzification is the process of deriving a crisp value from a fuzzy set using any defuzzification methods such as centroid, bisector, middle of maximum, largest of maximum, and smallest of maximum (Ross, 2010). Centroid method (also known as center of area, center of gravity) is used in the present research for finding the crisp value, representing the requirements and design level reliability the end of requirements and design phase respectively. This method is the most prevalent and physically appealing of all defuzzification methods (Ross, 2010).

4.6 IMPLEMENTING CORROBORATION PHASE

Although the developed reliability prediction model has been thoroughly corroborated theoretically as well as empirically in the next chapter, even though in order to analyze the fault prediction consistency and influence of various software metrics on early fault prediction some analysis has been presented.

Table 4.8 Reliability Prediction at Requirements Stage

<table>
<thead>
<tr>
<th></th>
<th>RS</th>
<th>RIW</th>
<th>RC</th>
<th>RFD</th>
<th>RLR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Case</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.953</td>
</tr>
<tr>
<td>Average Case</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.665</td>
</tr>
<tr>
<td>Worst Case</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.113</td>
</tr>
</tbody>
</table>

Table 4.9 Reliability Prediction at Design Stage

<table>
<thead>
<tr>
<th></th>
<th>RLR</th>
<th>EMc</th>
<th>CoMc</th>
<th>IMc</th>
<th>CMc</th>
<th>DLR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst Case</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.096</td>
</tr>
<tr>
<td>Average Case</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.55</td>
</tr>
<tr>
<td>Best Case</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.937</td>
</tr>
</tbody>
</table>
Table 4.8 and 4.9, and the figures 4.15, 4.16, 4.17, 4.18, 4.19 and 4.20 presents the values of RLR (Requirements Level Reliability) and DLR (Design Level Reliability) by the proposed model for the best, average and worst-case input values of different input metrics. These values of RLR and DLR signifying the lower and upper bounds of prediction range at the requirements and design phase respectively.

It can be easily noticed that the value of the RLR is 0.113 in the worst case, because the values of corresponding requirements level measure are at their extremes. The RLR at the end of requirements phase range from 0.113 to 0.953, while the range for DLR is 0.096 to 0.937, which is quiet satisfactory.

Figure 4.15 Worst Case at Requirements Phase
Figure 4.16: Average Case at Requirements Phase

Figure 4.17: Best Case at Requirements Phase
Figure 4.18: Worst Case at Design Phase

Figure 4.19: Average Case at Design Phase
The model also helps to determine the influence of a particular software metrics on the software reliability. Once the impact of the particular software metric on reliability has been identified, the better and more cost effectively it can be controlled to improve the overall reliability and quality of the product.

4.7 IMPLEMENTING ANALYSIS PHASE

After implementing the quantification phase successfully this is the next critical phase of the framework. The following sub sections analyses the different quantitative input as well as output values and inferred the suggestive measures along with the guidelines for improving the software reliability.
4.7.1 Analyze Quantified Reliability and Metrics

After ensuring that the developed model is running successfully, in this phase various artifacts involved in the reliability prediction needs to be further analyzed to know more about them. The following sub sections will perform this task for requirements and design phase separately.

Analyzing the Requirements Metrics:

Observing the quantitative change in the Requirements Level Reliability, on the basis of the quantitative variation in the values of requirement metrics following observations are noticed:

Individual Variation

As the value of RS moved towards 1 the value of RLR also increases.
As the value of RS moved towards 0 the value of RLR also decreases.
As the value of RIW moved towards 1 the value of RLR also increases.
As the value of RIW moved towards 0 the value of RLR also decreases.
As the value of RC moved towards 0 the value of RLR increases.
As the value of RC moved towards 1 the value of RLR decreases.
As the value of RFD moved towards 1 the value of RLR decreases.
As the value of RFD moved towards 0 the value of RLR increases.

Combinational Variation

As the value of RC and RFD is moved towards 0 the value of RLR moves towards 1.
As the value of RC and RFD moved towards 1 the value of RLR moves towards 0.

As the value of RS and RIW moved towards 0 the value of RLR also moves towards 0.

As the value of RS and RIW moved towards 1 the value of RLR also moves towards 1.

As the value of RC and RIW moved towards 1 or 0 the value of RLR neither increases nor decreases.

As the value of RS and RFD moved towards 1 or 0 the value of RLR neither increases nor decreases.

As the value of RC and RS moved towards 1 or 0 the value of RLR neither increases nor decreases.

Some of the combinations are discussed here, although all these combinations will be discussed more comprehensibly in the next chapter under the sensitivity analysis section. On the basis of above empirical observations following conclusion may be drawn.

“The higher the value of RS the more reliable the requirements will be”

“The higher the value of RC the less reliable the requirements will be”

“The higher the value of RIW the more reliable the requirements will be”

“The higher the value of RFD the less reliable the requirements will be”
Analyzing the Design Metrics:

Observing the quantitative change in the Design Level Reliability, on the basis of the quantitative variation in the values of Design metrics following observations are noticed:

**Individual Variation**

As the value of RLR moved towards 1 the value of DLR also increases.
As the value of RLR moved towards 0 the value of DLR also decreases.
As the value of EMc moved towards 1 the value of DLR also increases.
As the value of EMc moved towards 0 the value of DLR also decreases.
As the value of CoMc moved towards 1 the value of DLR also increases.
As the value of CoMc moved towards 0 the value of DLR also decreases.
As the value of IMc moved towards 0 the value of DLR increases.
As the value of IMc moved towards 1 the value of DLR decreases.
As the value of CMc moved towards 1 the value of DLR decreases.
As the value of CMc moved towards 0 the value of DLR increases.

**Combinational Variation**

As the value of EMc and CoMc is moved towards 0 the value of DLR moves towards 0.
As the value of EMc and CoMc moved towards 1 the value of DLR moves towards 1.
As the value of IMc and CMc moved towards 0 the value of DLR also moves towards 1.

As the value of IMc and CMc moved towards 1 the value of DLR also moves towards 0.

As the value of EMc and CMc moved towards 1 or 0 the value of DLR neither increases nor decreases.

As the value of CMc and CoMc moved towards 1 or 0 the value of DLR neither increases nor decreases.

As the value of EMc and IMc moved towards 1 or 0 the value of DLR neither increases nor decreases.

Some of the combinations are discussed here, although all these combinations will be discussed more comprehensively in the next chapter under the sensitivity analysis section. On the basis of above empirical observations following conclusion may be drawn.

“The higher the value of RLR the more reliable the design will be”

“The higher the value of CMc the less reliable the design will be”

“The higher the value of EMc the more reliable the design will be”

“The higher the value of IMc the less reliable the design will be”

“The higher the value of CoMc the more reliable the design will be”
4.7.2 Perform Contextual Interpretation and Develop Suggestive Measures

On the basis of the analysis being performed in the previous step, the next task is to frame different suggestive measures. These measures will be used as reliability improvement guidelines. These guidelines will assist to regulate the values of the requirements and design metrics, and improve the reliability of the developing software before the coding starts. Therefore following recommendations are developed for the personnel involved in the requirements and the design phase.

a) Keep the requirements change requests as low as possible.
b) Perform more and more review inspections and walkthrough.
c) Control the complexity of the new functional requirements.
d) Modules having complexity needs to handle by experienced requirement engineers.
e) Most critical modules should be review through senior and experienced peers.
f) Try to find as much SRS faults as possible, and try to reduce the fault density.
g) Identify the ambiguities as well as inconsistencies as early as possible.
h) Keep the level of inheritance as low as possible, because unnecessary data members and methods in sub classes increase the complexity of the class hierarchy.
i) Keep the level of class coupling as minimum as possible.
j) Sharing of data and method across the methods of other classes must be avoided until necessary.
k) Critical data members should not be declared as public until unavoidable.
l) Sensitive data should not be passed as parameter to other methods.
m) Try to enhance the encapsulation in the design as much as possible.

n) Develop the classes as cohesive as possible.

o) Visibility options should be handle very intelligently. Only in unavoidable situations data and methods should be declared as public.

These revisions will definitely proved to be significant in making the finally delivered software more reliable.

4.8 CONCLUSION

This chapter has comprehensively implemented the framework, proposed in the chapter 3. All the phases have been discussed, quantitative values have been analyzed thoroughly followed by the suggestive measure or guidelines to further improve the reliability of the developing software up to its design stage.