
6.1 Introduction

This chapter describes the implementation aspects of the *SoftRisk* prototype. Its implementation language, algorithms, and difficulties will be discussed.

6.2 Development Environment

As with any software, to develop the *SoftRisk* prototype, two parts software and hardware are invoked. Their specifications are stated below:

6.2.1 Software Used

To meet the requirements of *SoftRisk* prototype, the JAVA language has been selected as the implementation language of the prototype. JAVA has been selected because of many reasons. Some of them are listed below:

- Java capabilities can cover the implementation needs of *SoftRisk*
- Java is an object oriented, networks, Internet, and multimedia language, which leaves the door open for any future improvements.
- Java is supported to work on multi-platforms.
- Java provides many programming features and components. So *SoftRisk* prototype development can benefit from them.
The following are brief descriptions of the JAVA compilers and the server that have been used to develop the SoftRisk prototype.

- Java Development Kit (JDK) Version 1.1: The JDK contains software and tools that developers need to compile, debug, and run the JAVA applets and applications. The Kit has been introduced by Sun Microsystems.

- Symantec Café Version 1.8: Symantec corporation has launched this product in 1997. It incorporates the JDK 1.1 from Sun Microsystems. It has been used to develop the user-interface and database components of the prototype.

- IDS server Version 2.0.2: For the Internet version, the IDS server has been used to create deploy database interactive the JAVA applets. It supports JDBC-ODBC. This Internet access server has been introduced by IDS Software.

- In additional to the above development compilers and server, the Microsoft Access 97 has been used to build the database tables and, SQL statements are used to establish database queries.

### 6.2.2 Hardware Used

Most SoftRisk components especially the main classes (e.g. graphic and calculation classes) have been developed on the SPARC Sun workstation, but the implementation has faced some difficulties at the developing of database part. Therefore, the implementation of SoftRisk has been shifted to PCs.
6.3 *SoftRisk* Components Description

The first menu of *SoftRisk* contains four icons: the first one is to start running *SoftRisk* components which leads to prototype password and main pull-down menu. The second icon is to get some information about the prototype. The third one is used to get on-line help. The last icon is used to exit from the prototype. As Figure 6.1 shows *SoftRisk* comprises of six main components:

- Projects' risks handling
- Risks Documentation
- Risk Assessment and Prioritization
- Monitoring and Controlling
- Statistical data preparation
- Probability and Magnitude Assistance

However, all *SoftRisk* classes have been developed using JAVA.

![Diagram](attachment:image.png)

**Figure 6.1:** *SoftRisk* Main Components
6.3 *SoftRisk* Components Description

The first menu of *SoftRisk* contains four icons: the first one is to start running *SoftRisk* components which leads to prototype password and main pull-down menu. The second icon is to get some information about the prototype. The third one is used to get on-line help. The last icon is used to exit from the prototype. As Figure 6.1 shows *SoftRisk* comprises of six main components:

- Projects' risks handling
- Risks Documentation
- Risk Assessment and Prioritization
- Monitoring and Controlling
- Statistical data preparation
- Probability and Magnitude Assistance

However, all *SoftRisk* classes have been developed using JAVA.

![Diagram of SoftRisk components](image)

**Figure 6.1:** *SoftRisk* Main Components
The Following algorithm traces the main steps of SoftRisk prototype in dealing with software risks.

Step1: // Identification
  (1.1) Get (Risks’ data);
  (1.1) Get (Projects’ data, Projects’ risk data);
Step2: // Estimation
  (2.1) If (assistant is needed) {
    Call (Prob. Mag. estimator); // To estimate risk’s Prob. Or Mag
  }
  (2.3) Get (Risk’s Prob. Mag. estimation);
Step3: // Documentation
  (3.1) Document1 (Risks’ data);
  (3.2) Document2 (Project’s data, Project’s risk data);
Step4: // Assessment
  (4.1) Accept (Selected project);
  (4.2) Compute (RE);
Step5: // Priority
  (5.1) Priority (Risks List based on RE);
  (5.2) Specify (Most top ten risks);
Step6: // Monitoring
  (6.1) Get (Current top risk items);
  (6.2) Call (Graphic builder);
  (6.3) Display (Line Graph);
Step7: // Controlling
  (7.1) Select (Inspection time);
  (7.2) Display (Priority list);
  (7.3) Specify (Risk);
  (7.4) Display (Risk’s important data, Controlling levels, Reduction techniques);
  (7.5) Control (Risk);
  (7.6) Update (Prob. & Mag.);
  (7.7) Document (Actions, Responsibly);
  (7.8) Reassess (Compute RE, RCV);
Step8:
  (8.1) Perform (Simple Statistical Processes);
  (8.2) If not (Project-end) Go to step1;

The following sections will describe the implementation of SoftRisk components.
6.3.1 Projects' Risks Handling

The object of Projects_Risks class handles this task. As Figure 6.2 illustrates this task involves four main components: specifying a group of risks, selecting one of project's records, adding data.

![Diagram of Projects' Risks Handling Components](image)

**Figure 6.2:** Projects' Risks Handling Components

The main goals of this task are to identify, estimate, and document all specific project’s risks data in PRSF. The documented data will be useful for tracking the risks situation, statistical operations, and performing further risk management steps.

However, once a project is infected by any risk, a new record will be created and all relevant data will be identified and documented immediately in PRSF. The PRSF fields are considered as the first part of the risk matrix which is presented in Table 5.1 in chapter 5.

Before describing the above components, it should be noted that the project code is a unique identity for each project only, but one project can have any number of records holding the same code and password. The following sections describe the components.
6.3.1.1 Specifying A Group of Risks

It is an optional feature. It allows the user to specify what risks he/she wants to work on at data-entry time. A list of registered risks in the RskDB will be shown to the user. The user can select one, a few or all of the risks from the list. But if he/she does not select, all risks will be selected by default. Figure 6.3 illustrates the sub-components of this component.

![Diagram showing the process of specifying risks](Image)

Figure 6.3: Specifying Risks for Entering Purpose

6.3.1.2 Selecting One of Project’s Records

As can be seen in Figure 6.5, user should assign one of the projects’ records before performing any operation. SoftRisk supports this operation by furnishing the user with a list of registered projects in PRSF, and a table consists of three columns: project’s records codes, risk codes, and inspection numbers. From the provided data, the user can decide and detect which record he/she wants to work on.
The project's password should be entered as well. If the password is correct the user can perform any of the database functions. Figure 6.4 clarifies that the selected record will be displayed before any action. In the provided frame, the *Ok-button* confirms the actions, while *Cancel-button* is to ignore the actions.

**Figure 6.4:** Selecting One of Projects Records for Display, Update, or Delete
Figure 6.5: Projects’ Risk Record Selection

6.3.1.3 Adding Data

Adding a new data means that a new record will be created. It can be for old or new project. This task implies identifying risk, and documenting all related data. Hereafter a descriptions of its five sub-components:

- Project-Code checker
- Entering immediate identification data
- Risk code provider
- Probability and magnitude estimator
- Inspection numbers detector
a) Project-Code Checker

The following algorithm is enough to clarify how the project-code’s checker works.

**Step1:** // Accepting project identity data
(1.1) Accept (Project-Name, Project-Code, Project-Password);

**Step2:** // Comparing with the PRSF registered records
(2.1) if (Project-Code is new) Go to Step3;
else if (Project-Password is correct) go to Step3;
Otherwise Message ("duplicated code or wrong password");

**Step3:** // Copying data into P_Risk_C main frame
(2.3) PRSF-frame ← (Project-Name, Project-Code, Project-Password);

**Step4:** if (clear) clear (the fields);

Finishing this check leads to resumption of other data-entry tasks.

b) Entering an Immediate Identification Data

Immediate data refers to the type of data, which does not need a preparation by other components. In additional to the contents of Project-Code checker fields (i.e. Project-Code, Project-Name, and Project-Password) it involves developer’s name, risk’s cause, project’s type and size, and lifecycle phase. However, the project type, phase, and size are drop-down menus which consist of the following items:

<table>
<thead>
<tr>
<th>Project type:</th>
<th>Project Phase:</th>
<th>Project size:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application project</td>
<td>Requirements</td>
<td>Small Scale &lt; 10 KDSI *</td>
</tr>
<tr>
<td>System project</td>
<td>Analysis</td>
<td>Medium Scale 10 – 100 KDSI</td>
</tr>
<tr>
<td>Language project</td>
<td>Design</td>
<td>Large Scale 100-500 KDSI</td>
</tr>
<tr>
<td>Utility project</td>
<td>Implementation</td>
<td>Very Large &gt; 500 KDSI</td>
</tr>
<tr>
<td>CASE Tool Project</td>
<td>Integration</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Maintenance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Retirement</td>
<td></td>
</tr>
</tbody>
</table>

* The project size is scaled by Kilo Delivery Source Instruction (KDSI)
c) Risk Code Provider

Each risk has its own code called Risk-Code. This code can be used as a linker between PRSF records and RskDB. This component calls SpecifyCode method in Risk_Documentation class to specify the Risk_Code. The user clicks on Risk-code button, and a special frame containing a list of all registered risks in RskDB will be opened immediately with the following options:

- If the user has already selected a sort of risks to work on, only those risks will be listed, otherwise, all risks will be listed by default.
- The user can highlight any one of the risks and select it. Then its code will be automatically invoked into the PRSF opened record.
- The user can display the risk’s record before deciding to select it.
- If the provided list does not include the risk (i.e. the risk is new), the user has to complete a new record for the risk in the RskDB before he/she can use it. Once its record is completed the risk code will automatically be invoked into PRSF record.

c) Probability and Magnitude Estimator

Estimating risk’s probability and magnitude is a difficult task to software developers. Quantitative data is much accurate, but qualitative data is much easier for developers to estimate. Therefore, most software developers prefer to estimate the probabilities and magnitudes of the risks in the qualitative form rather than the quantitative form data.
SoftRisk provides the user with radio buttons to estimate the probabilities and magnitudes in qualitative form. The radio buttons are labeled as:

- N for Negligible
- M for Medium
- VH for Very High
- L for Low
- H for High
- EH for Extra High

The user’s qualitative estimation will be converted into its equivalent quantitative data, for internal SoftRisk calculations.

SoftRisk assists the user in estimating the probabilities and magnitudes of software risks. Estimation assistance is given once the user clicks on the provided button. An object of the Prob_Mag_Utilization class will be created and a new frame containing a checklist will appear. Figure 6.6 shows the frame of probability side. The user has to tick on the checklist provided boxes. Therefore, SoftRisk evaluates ticks and introduces qualitative estimation. The following algorithm clarifies the assistance estimation of the probability and magnitude steps. The user can accept or ignore the suggested estimation.

Step 1: // Displaying the risk’s Prob. or Mag. estimation checklist
(1.1) Display (estimation checklist); // 10 elements

Step 2: // Obtaining user’s selections (i.e. the ticks on positive elements)
(2.1) Accept (user’s selection); // User ticks

Step 3: // Computing the overall sum of positive ticks weights
(3.1) SUM ← (selected item weights); // overall sum between 1 and 100

Step 4: // converting the sum into its equivalent qualitative form
// and displaying the estimated values
(4.1) If (SUM<25) Show("N"); // N- Negligible
(4.2) If (SUM>=25 & & SUM<42) Show("L"); // L- Low
(4.3) If (SUM>=42 & & SUM<59) Show("M"); // M- Medium
(4.1) If (SUM>=59 & & SUM<75) Show("H"); // H- High
(4.1) If (SUM>=75 & & SUM<92) Show("VH"); // VH- Very High
(4.1) If (SUM>=92) Show("EH"); // EH- Extra High

122
Figure 6.6: Risk Magnitude Estimator

d) Inspection Number Detector

Each PRSF record must be entered in conjunction with an inspection number. Based on the last registered inspection number of the desired project, SoftRisk automatically finds out the inspection number. It provides the user with some selection options so that he/she can select current, previous, or next inspection number.

Once all PRSF frame fields are completed, the user has to confirm the record creation by clicking Add button and Ok button to confirm, or Cancel button to ignore. However, on-line help is also available.
6.3.2 Risks Documentation

This component has been developed to document fixed risks data. The data will be documented in RskDB, which can be considered as a second part of the risk matrix. However, the Risk_Documentation class handles this task which aims to document any kind of software development risks and make it ready to use by any software development projects.

All risks’ records must be unique records. Risks' data comes at any time from any available resources (e.g. related software surveys, PRSF, previous projects, developer experience, publications, and Internet). Risks data are stored in RskDB which can be used by all SoftRisk components.

RskDB frame (its design has been shown in Figure 5.3 in chapter 5) accepts all significant data that describe the risk and its nature. These data involve the description of the risk, reduction techniques, deadlines, and controlling levels.

Furthermore, Risk_Documentation class creates an object of Prob_Mag_Preparation class to prepare the risk’s probability and magnitude estimator checklist. Figure 6.7 illiterates that Risk_Documentation class handles these two tasks: adding a new risk record, and working on previous registered records. Following sections describe them.
6.3.2.1 Adding Data to RskDB

As it can be seen in the left side of Figure 6.7 firstly, SoftRisk checks risk code to ensure its uniqueness (a special method handles this task). Once the risk is confirmed new, the user can resume entering the remaining data which involves two tasks: entering immediate risk’s data, and preparing the risk’s probability and magnitude estimator checklist.
a) Entering an Immediate Risk’s Data

As what was stated before, immediate data is the data which do not need a preparation by other classes. Immediate risks data involves risk’s code, name and the following:

- Risk description: A short description of the risk nature in a text field form.
- Risk cause: The main reason that causes or leads the risk appears or arises. It is in a text field form.
- Reduction plans: A provided text area to accept mitigation, contingency and crisis plan. The reduction plans contain the techniques and all relevant information e.g. its requirements and implementation procedure steps.
- Two groups of radio buttons, scaled from Negligible to Extra High, are provided to detect the invoking deadlines of the contingency and crisis plans.
- Another two groups of radio buttons are also scaled from Negligible to Extra High to detect when the risk will be at the highest level, and when it will be at the acceptable level.
- Drop-down menus are used to determine the most important software project type and phase which most likely will be attacked by the desired risk.

b) Preparing Risk’ S Probability and Magnitude Estimator

This part creates an object of the Prob_Mag_Preparation class. A special frame will be opened to prepare the estimation checklist. This preparation must be done for each risk
along with entering of other risk's data. However, the mentioned frame includes risk code, risk name, ten fields for the estimation checklist items, and other ten fields to specify weights for the items.

*CheckWeights* method checks the sum of weights to ensure that the sum is exactly 100 (i.e. this to ease the evaluation task). The weights are not necessary the same for all items and the number of items can be different from one risk to another.

### 6.3.2.2 Working on Previous Registered Records

The right side of Figure 6.7 illustrates this task. However, before working on any of the registered risks records the phase must be specified. A provided drop-down menu allows the user to specify the phase. As it can be understood from Figure 6.8 the user has also to specify the risk itself.

Once, the selection is finished the user can display, update, or delete risk's record. The following buttons are provided to deal with the opened record:

*Show* button: This button displays the contents of the record. If there is any changes on the record it can be saved.

*Update* button: The user can update the risk's records and its probability and magnitude estimation checklist.

*Delete* button: If the user clicks on this button the selected record will be displayed for deleting purpose.
All the above tasks are confirmed only if the user clicks on the Ok button.

![Risks Database](image)

**Figure 6.8:** Specifying A Risk Record

### 6.3.3 Risk Assessment and Prioritization

An object of `Assessment_and_Prioritization` class handles this task. `SoftRisk` performs steps to assess software risks. It prepares a priority list for them for the monitoring and controlling purposes. The following algorithm clarifies those steps:

**Step1:** // Specifying a project for monitoring and controlling purposes  
(1.1) SP ← Select (Project);  
**Step2:** // Copying some selected data to a buffer file.  
// The data is sorted by (RE = Prob. * Mag.)  
(2.1) Bufferdata ← Copy (SP selected data e.g. `P_Precode, P_Pno, P_dur, P_RE, P_Pnum, P_Padec` Sorted by RE);  
**Step3:** // Building a semi ready priority list of the risks  
(3.1) BufferReady ← Copy (important data from Bufferdata and its matched data in RskDB Sorted by inspection numbers);
Step 4: // Determining the most top ten for each inspection time
(4.1) For k = 0 to Number of inspections {
   // For each inspection time the following steps will be done
   (4.1.1) Tem-Vectors ( ) ← (Inspection records in BufferReady);
   (4.1.2) D-Count ← Count (Inspection registered records);
   (4.1.2.1) For j = 0 to D-Count { // Specify the most top ten risks
      Ready-Vectors( ) ← At most ten elements of Tem-Vectors( ). (Project record code, RE);
   } Next j;
} Next k;

Step 5: Calling the graphic builder
(5.1) Graphic-builder (Ready-Vectors( ));

Some vectors and two temporary database tables (Bufferdata and BufferReady) support the assessment and prioritization task. The Bufferdata table (Table 6.1) holds data sorted based on RE values, while BufferReady table (Table 6.2) holds the risks data sorted by inspection numbers and RE.

<table>
<thead>
<tr>
<th>Inspection Number</th>
<th>Record’s Risk Exposure RE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EH</td>
</tr>
<tr>
<td>2</td>
<td>VH</td>
</tr>
<tr>
<td>2</td>
<td>VH</td>
</tr>
<tr>
<td>1</td>
<td>H</td>
</tr>
<tr>
<td>2</td>
<td>H</td>
</tr>
<tr>
<td>1</td>
<td>M</td>
</tr>
<tr>
<td>1</td>
<td>L</td>
</tr>
<tr>
<td>2</td>
<td>L</td>
</tr>
</tbody>
</table>

Table 6.1: Bufferdata Example

Other record fields will follow the RE field sorting.
Table 6.2: BufferReady Example

<table>
<thead>
<tr>
<th>Inspection Number</th>
<th>Record's Risk Exposure RE</th>
<th>Other record fields will follow the inspection field sorting.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EH</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>VH</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>VH</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>L</td>
<td></td>
</tr>
</tbody>
</table>

6.3.4 Monitoring and Controlling

Through `Monitoring_and_Controlling` class `SoftRisk` provides the user with risk monitoring and controlling capability which indirectly invokes two other classes namely `Graphic_Builder` and `Controlling classes` to perform this task. The components that are assigned for this task are listed and described below:

- Selecting a project.
- Monitoring Preparation.
- Graphic Builder.
- Risk Controlling.

6.3.4.1 Selecting A Project

Before starting the monitoring and controlling tasks, a list of all registered projects will be displayed to the user. The user has to specify a project to work on. Once the
project is selected from the list, the monitoring and controlling steps will be started automatically.

6.3.4.2 Monitoring Preparation

Once the project is selected an object of the Assessment_and_Prioritization class will be created. This object prepares a priority list of the top ten risks of each inspection time. These lists will be the input of the coming components (graphic builder and risk controlling).

6.3.4.3 Graphic Builder

Graphics make the monitoring task much easier, more friendly and faster than the data reporting. That is why project managers prefer to use graphics. Hence, SoftRisk is supported to represent projects risks situations by a simple line graph for monitoring and controlling purposes. The line graph represents top ten risks for each risk inspection which were specified by Assessment_and_Prioritization class. The risks can be different from inspection to inspection in terms of their types and numbers. This depends on RE values and how many times the project has taken risk.

The graph area has been divided into three main zones that are colored in green, yellow and red for low, medium and catastrophic risks respectively. The x-axis represents the inspection numbers, whereas the y-axis represents RE values.

The graph is also scaled horizontally from negligible risk to catastrophic risk to get more understanding of the risk situation. Moreover, vertical lines divide the graph to distinct the inspection times.
Scrolling is supported by the graph. Detailed information about each risk can be obtained by clicking on the Control-Panel button, which creates an object of Controlling class.

6.3.4.4 Risk Controlling

Once top risks are detected, the next step is to control them. A suitable reduction technique to reduce their impact must be performed. An object of Controlling class handles this task in order to:

- Support the monitoring task by allowing the user to get more details about the risk (priority list, levels, etc.).
- Introduce reduction advice (i.e. in terms of resolution techniques) to prevent or reduce the risk impact.
- Accept new risk’s probability and magnitude and re-assess the situation.
- Track the efficiency of the reduction techniques by documenting the actions that are taken against the risk and other related information.

Risk controlling consists of seven sub-components: (selection, risk bibliography data, reduction advisor controlling, dead-lines, levels, controlling and re-estimation, re-assessment and actions documentation).
a) Selection

It allows the user to specify the inspection number and the risk that he/she wants to work on. Since the call is originally came from the graphic builder it means that the project is already specified. This component involves two lists: one is to specify the inspection number and the second to detect the exact risk.

b) Risk’s Bibliography Data

A group of text fields displaying identification data related to the selected risk: e.g. its name, cause and its description. These data are important to understand the nature of the risk.

c) Reduction Advisor

One of SofiRisk's goals is to introduce reduction advice. A special component has been designed and implemented to perform this task. The user clicks on one of provided buttons, which directly displays the selected reduction technique. It could be mitigation technique, contingency plan, or crisis plan. This information will be displayed in a text area format.

d) Dead-Lines

Invoking any one of the reduction techniques depends on the risk impact level RE. Deadlines of invoking contingency and crisis plan will be shown to the user. So he/she can perform a suitable reduction technique at the right time.
e) Controlling Levels

Through this component the user can observe some important levels e.g. risk exposure, risk’s highest allowed level and the risk’s acceptable level which are described below:

- **Risk Exposure**

  It is a text field showing the existing value of the RE (i.e. the last registered RE) following algorithm converts the RE from quantitative form into its equivalent qualitative form to make it much easier for the users to understand:

  **Step1:** // To estimate the RE level within values between 0-6
  // REVALUE is the quantitative value of RE
  (1.1) REVALUE=REVALUE/36*6;

  **Step2:** // to convert it into its equivalent qualitative form
  // Negligible, Low, Medium, High, Very High and Extra High
  (2.1) if (REVALUE <=1.5 ) display("N");
  (2.2) if (REVALUE >1.5 && REVALUE <=2.5 ) display ("L");
  (2.3) if (REVALUE >2.5 && REVALUE <=3.5 ) display ("M");
  (2.4) if (REVALUE >3.5 && REVALUE <=4.5 ) display ("H");
  (2.5) if (REVALUE >4.5 && REVALUE <=5.5 ) display ("VH");
  (2.6) if (REVALUE >5.5) display("EH");

- **Highest Allowed Level**

  It is a level whereby if any risk’s RE exceeds, the project will be in trouble or it can be terminated. It is a very sensitive indicator to keep the RE level below the critical level, otherwise the risk will be very difficult to control.
• Acceptable Level

It is a level to determine whether the risk is acceptable or not. Once the RE of the risk goes down and reaches the acceptable level it can be considered that the risk is under control and the project is safe.

f) Controlling and Re-estimation

A suitable reduction technique (i.e. mitigation, contingency, or crisis plan) should be performed to prevent or reduce risk’s impact. After performing the suitable reduction technique, the changes of risk probability and magnitude should be re-estimated, so that the new situation can be assessed. This component provides radio buttons (scaled from Negligible to Extra High) for risk re-estimation. Risk probability or magnitude estimation assistance is also provided.

g) Re-assessment and Actions Documentation

Risk re-assessment task implies three activities:

• Saving the New Estimated Probability and Magnitude Values

*SoftRisk* tracks project situation from the risk perspective. Any related changes are important and should be documented. This component documents any changes in the risk probability and magnitude.
• Re-assessing and Computing RE\textsubscript{change} and Risk Change (RCV) rates

\textit{RE}_{\text{change}} \text{ is a percentage rate to illustrate the difference between the old and new RE, whereas, RCV is a percentage rate to illustrate the amount of the reduction. The following algorithm clarifies how the prototype computes the \textit{RE}_{\text{change}}, and the risk change value (RCV) rates:}

\textbf{Step 1:} // Getting the old and new probabilities and magnitudes

// RPROB\_OLD, RMAG\_OLD are the last probability and magnitude values

// PPROB\_new, PMAG\_new are the newest probability and magnitude values

(1.1) Read (RPROB\_OLD, RMAG\_OLD);
(1.2) Accept (PPROB\_new, PMAG\_new);

\textbf{Step 2:} // Computing \textit{RE}\_old, and \textit{RE}\_new

// \textit{RE}\_old is the last Risk Exposure value
// \textit{RE}\_new is the new Risk Exposure value

(2.1) \textit{RE}\_old = RPROB\_OLD * RMAG\_OLD;
(2.2) \textit{RE}\_new = PPROB\_new * PMAG\_new;

\textbf{Step 3:} // Computing the percentage of risk change \textit{RE}\_change:

(3.1) \textit{RE}\_change = \textit{RE}\_new-\textit{RE}\_old;
(3.2) \textit{RE}\_change = Math.round ((\textit{RE}\_change/36)* 100);

\textbf{Step 4:} // Computing the percentage of Risk Change Value RCV

(4.1) RRp = PPROB\_new - RPROB\_OLD;
(4.2) RRm = PMAG\_new - RMAG\_OLD;
(4.3) RCV = RRp * RRm;
(4.4) RCV = Math.round ((RCV/36)*100);

RCV is a Risk Change Value or in other words its the amount of reduction. RR\textsubscript{p} is a reduction amount in risk probability
RR\textsubscript{m} is a reduction amount in risk magnitude

• Documenting the Reduction Actions

Figure 6.9 shows a frame which consists of a text area field and a text field. The frame is opened to document the actions that could be taken to
prevent or reduce the risk and any related information. The name of the person in charge will be saved as well.

![Diagram of SoftRisk Implementation](image)

**Figure 6.9:** Reduction Action Documentation

### 6.3.5 Statistical Data Preparation

Statistical is a very important issue for predicting the future and planning for further steps. Hence, *SoftRisk* is intended to prepare risks data in a form that can be used for statistical operations. The targeted data involve the documented data in both RskDB, and PRSF. However, this class creates objects of some other classes to handle statistics preparations. The following sections describe that.
6.3.5.1 Risks Threats

This table contains four columns entitled: risk code, risk name, risk description, and risk phase. This table is displayed in order to detect the nature of the risks, which could affect software development projects. The user can sort the table based on any of the field’s titles. He/she can also print the contents, and move the list up and down.

6.3.5.2 Risks Frequency

The main goal of the frequently risks list is to determine the frequency of each risk. Those risks with a high frequency can be considered in the future plans. The user can specify the projects that he/she needs to check the number of times each risk has attacked them (see the provided algorithm). From the table, the developer can discover the highest frequency risks. The higher frequency, the higher the probability.

**Step1:** // Copying all registered risks codes into a special array
(1.1) All-Risks-Codes[] ← Copy(All risks codes in the RskDB);

**Step2:** // Assigning the target project and copy them into a special array
(2.1) Selected-Projects[] ← Accept (Target-projects);

**Step3:** // Obtaining the risks codes which affected the target projects
(3.1) For i=0 to Selected-Projects.length() {
    Projects’-Risks-Codes[i] ← Add(All project’s risks codes);
}

**Step4:** // Counting the frequency of each risk
(4.1) For u =0 to All-Risks-Codes.length(){
    (4.1.1) For l =0 to projects’- Risks-Codes.length(){
        (4.1.1.2) If (Projects’-Risks-Codes[l] = All-Risks-Codes[u])
            {R-Count[u] = R-Count[u] +1 ;}
    }
}

**Step5:** // Displaying the result
(5.1) For f=0 to All-Risks-Codes.length(){
    Display (All-Risks-Codes[f], R-Count[f])
}
6.3.5.3 Risks Sorting

A special table contains, projects' codes, risks' names, risks' probabilities, and risks' magnitudes sorted by RE values. It is an important table to prepare plans to the future risk management strategies.

Risk management effort can be focused primarily on those risks with highest impacts on the projects. Other projects can gain from this part to estimate the probabilities and magnitudes of the similar risks. Furthermore, monitoring risks' probabilities and magnitudes trend can monitor efficiency of managing techniques and the relations between risks.

6.3.5.4 Risk Infection

Figure 6.10 clarifies how the user can specify some risks to get a list of projects that have been infected by those risks. Furthermore, the user can obtain a list of all risks that have affected any one of the projects. This approach aims to examine the hidden interaction amongst the projects and the risks, and amongst the projects.

**Figure 6.10:** Diagram of Infected Projects Processing
6.3.5.5 Risks Similarity

The user has to specify projects that he/she wants to find out the similarity amongst them. Hence, risk impact level (e.g. Negligible, Low, etc.), and lifecycle phase(s) must be specified as well. Display button lists all risks that affected those projects. Display-similarity button lists all similar risks of those projects. The following algorithm clarifies how this part works:

**Step 1:** // Obtaining the target projects and specifying risks impact level
(1.1) selected_Pnames[] ← (Target-projects); // target projects
(1.2) P_L ← convert risk impact level into 1-6; // P_L is the risk level

**Step 2:** // Specifying the target phase
(2.1) While (specifying phases) {
    (2.1.1) PH ← phase_code; // PH is the lifecycle phase
    (2.1.2) Selected (PH);
}
Selected (PH) {
    // Finding matched projects to the indicated phases and risks level
    For r=0 to selected_Pnames.length( ){
        While (! PRSF.eof( )) {
            Read (Pname, PRprob, PRmag, PRnum);
            RE-value ← Prprob * PRmag;
            RE ← convert RE_value into levels 1-6;
            If (RE=R_L) {
                all_match_projects[] ← Pname;
                unique_match_projects[] ← add Pname if not added before;
                All_Risks_Codes[] ← PRnum;
                unique_Risk_Codes[] ← add PRnum if not added before;
            }
        }
    }
}

**Step 3:** // displaying a list of the risks which are infected the projects
(3.1) For k=0 to unique_Risk_Codes.length{
    (3.1.1) While (! RskDB.eof( )) {
        (3.1.1.1) If (DRNo = unique_Risk_Codes[k]){
            Display_List ← DRName;
        }
    }
}
Step 4: // Determining the similarity of risks
(4.1) For h=0 to unique_Risk_Codes.length{
    (4.1.1) Element ← unique_Risk_Codes[h];
    (4.1.2) Scan (Element);
    (4.1.3) Compare (Element);
    (4.1.4) Remove (unnecessary temporary data);
}

Scan (Element){
    TempV[]; f=0;
    TempV. clear();
    For c=0 to All_Risks_Codes.length{
        If (All_Risks_Codes[c]= Element){
            TempV[f] ← all_match_projects[c];
            f++;
        }
    }
}

Compare (Element){
    Unique_tempV[]; f=0;
    Unique_tempV.clear();
    For h=0 to TempV.length{
        Unique_tempV[f] ← add TempV[h] if it has not been added before;
        f++;
    }
    For h=0 to unique_TEMPV.length {
        If (Unique_tempV.length >= unique_match_projects.length){
            Similarity_list ← DRName from RskDB with code(Element);
        }
    }
}

Step 6: //displaying the similarity
(6.1) Display (Similarity);

Determining similarity is a very useful feature in understanding the interaction and
interrelation amongst the projects' risks. Many points can be stated to illustrate how the
similarity is useful. Following are some of them:
Chapter 6  

- Indicating where those risks have a relation (e.g. in which projects, levels, or phases).
- The dependency of the risks. Some risks may increase or decrease based on other risks.
- Risks that have specific impact (e.g. very high) on certain projects for certain phases.
- Some resolutions can be shared amongst those projects for similar risks.
- The similarity list can be used to compare projects’ risks.

6.3.6 Implementation Problems and Difficulties

During the implementation stage the development of SoftRisk has encountered some problems. Most of them are related to the JAVA programming language which is used to develop the SoftRisk. Following are a few of those problems:

- JAVA has been chosen as an implementation language of SoftRisk. When the implementation of SoftRisk was started the JAVA language was still under construction. There was no stability and insufficient material and support were available.
- It was so difficult to find solutions for the programming problems because few people were familiar with some of the JAVA aspect.
- SoftRisk has been implemented on Sun workstation. But because of some problems that are related especially to interaction between the Java applets
and the database connectivity (JDBC), the implementation has been shifted to PCs.

- JAVA is an Internet language, but there are many problems related to its database applets security. Many other problems related to the relation between the JAVA and Microsoft applications.

- Tables were not supported by early versions of Java Development Kit (JDK). The SoftRisk tables have been designed and coded without using any ready functions.

6.4 Conclusion

This chapter has been started with a description of SoftRisk development environment (Software used, hardware used, etc.). However, the implementation phase has been described by describing SoftRisk components, classes, menu-hierarchies, and development algorithms.

Furthermore, the description of the implementation phase has been supported with some screen shots of SoftRisk windows. Finally, some of implementation problems have been listed which are mainly related to development language. As the SoftRisk is implemented, it is the time to test and validate it, which is included in the coming chapter.