APPENDIX 1
SAMPLE OF SOFTWARE REQUIREMENT SPECIFICATIONS (SRS)

<table>
<thead>
<tr>
<th>Application Name</th>
<th>Post Office Management System (POMS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>e-Governance</td>
</tr>
<tr>
<td>Technology</td>
<td>C#.NET</td>
</tr>
</tbody>
</table>

POST OFFICE MANAGEMENT SYSTEM (POMS) VER 2.0

SOFTWARE REQUIREMENT SPECIFICATION

Copyright ©Trimentus. All rights reserved. No part of this publication may be reproduced, photocopied, stored in a retrieval system, or transmitted without the express prior consent of Trimentus, Chennai, India.

Trimentus makes no representations or warranties with respect to the contents or use of this document, and specifically disclaims any express or implied warranties of merchantability for any particular purpose. Trimentus reserves the right to revise this document and make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.
1. INTRODUCTION

1.1. Purpose

An automated module for the Post Office will help in automating functions of the administration department. It helps in reducing the time spent in record keeping and the work can be carried out effectively. The searching of records in future will also become easy. The redundancy in the data due to manual data will also be tackled. The Post office will be able to access the personal information of each customer easily.

The administration department will also be able to add new branches of post office, staffs, customers, set the rates for the courier or any other parcel based on duration, weight, distance, and type of delivery.

1.2. Scope

This subsection should:

1. Identify the software product(s) to be produced by name; for example, Host DBMS, Report Generator, etc.
2. Explain what the software product(s) will, and, if necessary, will not do.
3. Describe the application of the software being specified.

1.3 Approvals and Authorizations

<table>
<thead>
<tr>
<th>Designation</th>
<th>Name</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author</td>
<td>Team Member</td>
<td>Lakshmi</td>
</tr>
<tr>
<td>Approved By</td>
<td>Team Member</td>
<td>Uthra</td>
</tr>
<tr>
<td>Authorised By</td>
<td>Project Coordinator</td>
<td>PM Shareef</td>
</tr>
</tbody>
</table>

1.4 Distribution

<table>
<thead>
<tr>
<th>Copy No</th>
<th>Holders Designation</th>
<th>Issue Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Project Coordinator</td>
<td>20&lt;sup&gt;th&lt;/sup&gt; Nov’ 08</td>
</tr>
<tr>
<td>2</td>
<td>Project Head</td>
<td>20&lt;sup&gt;th&lt;/sup&gt; Nov’ 08</td>
</tr>
<tr>
<td>3</td>
<td>Team Members</td>
<td>20&lt;sup&gt;th&lt;/sup&gt; Nov’ 08</td>
</tr>
</tbody>
</table>
1.5. Amendment Record

<table>
<thead>
<tr>
<th>No</th>
<th>Date</th>
<th>Section (s)</th>
<th>Issue No</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6th Nov' 08</td>
<td>All</td>
<td>01</td>
<td>Initial Version</td>
</tr>
<tr>
<td>2</td>
<td>20th Nov' 08</td>
<td>All</td>
<td>02</td>
<td>Second Version</td>
</tr>
</tbody>
</table>

2. SYNOPSIS

An automated module for the Post Office will help in automating functions of the administration department. It helps in reducing the time spent in record keeping and the work can be carried out effectively. The searching of records in future will also become easy. The redundancy in the data due to manual data will also be tackled. The Post office will be able to access the personal information of each customer easily.

The administration department will also be able to add new branches of post office, staffs, and customers, set the rates for the courier or any other parcel based on duration, weight, distance, and type of delivery.

3. GETTING STARTED

Post Office management System is an application which is created for the use of an automated system all over the country. The Administration Block of the Post Office, situated at the head office holds the information of the customers who sent deliveries regularly with the firm and the payment details for the delivered item.

This project helps the Post Office to maintain their records systematically. It provides functions such as setting rates based on weight, days/ duration, distance and type of delivery etc… The scheduling modeling of the Post Office Management System maintains logs and preference cards.

The administration department schedules the delivery based on the specified delivery date and weight of the item. All these details will be kept as records. So they store these information in various files for easy access in future.
4. SYSTEM REQUIREMENTS

4.1 Requirements

The minimum hardware and software requirements are as follows.

4.2 Hardware Requirements

Processor : Pentium 4
Ram capacity : At least 512 MB
Clock speed : 400MHZ
Hard disk drive : 40GB
Floppy disk drive : 3 ½ inch
Keyboard : 104 keys
Mouse : PS/2 mouse
Monitor : SVGA color monitor
Network adapter card : Ethernet Card

4.3 Software Requirements

Operating System : Windows XP
Programming language : C#
Technology used : ASP.NET
RAD Tool used : Visual Studio 2005

5. PROJECT STATEMENT

5.1 Existing Scenario

The Post Office Management System is currently using a standalone application which holds certain details. Now they want an online application which will allow the customers to view their details and whether their packets are sent safely. Also they don’t want several applications to be used at different locations. They want to use this online application at all places i.e. All over the country.

5.2 Proposed Solution

1. A Single application through which offices at different locations are connected.
2. Three categories of people can login namely, the administrator, the staff and the customers.
3. The administrator will be able to view all kinds of details like customer details, delivery details, factors to set charges, list of post offices and their addresses and ids etc… and he will also be able to update, edit, and delete them.

4. The customer can only view his own details, update his own profile, view charges for various factors, his delivery details.

5. The staff members of the post office will be able to view all the contact details of the customers, rates, delivery details, charges for various factors, update their own profile etc…

6. Each delivery item will be given a delivery id for future reference.

7. The customer will be able to sign up initially only at the post office. Later on he / she will be able to login from anywhere to view his profile or delivery details.

5.3 Functional Requirements

1. The details will be updated regularly. Example: Date received / returned will be recorded according to what had happened immediately.

2. Calculation of the charges as per the firm’s policy.

3. Charges for VPP will be collected only at the time of delivering the item

4. Only the administrator will be allowed to update any kind of details except delivery.
6. DESIGN PHASE

6.1 Data-Flow Diagram (DFD)

Figure A 1.1: Data Flow Diagram of POMS
6.2 Entity Relationship (ER) Diagram

a) New User

![Entity Relationship Diagram of POMS New User Creation](image)

Figure A 1.2: Entity Relationship Diagram of POMS New User Creation

b) Customer

![Entity Relationship Diagram of POMS Customer Creation](image)

Figure A 1.3: Entity Relationship Diagram of POMS Customer Creation
c) Delivery Type, Weight and Distance

Figure A 1.4: Entity Relationship Diagram of POMS Delivery, Weight and Distance

d) Duration

Figure A 1.5: Entity Relationship Diagram of POMS Duration
e) Set Distance

![Diagram](image1)

Figure A 1.6: Entity Relationship Diagram of POMS Distance

f) Post Office

![Diagram](image2)

Figure A 1.7: Entity Relationship Diagram of POMS Post Office Creation
g) Delivery

![Entity Relationship Diagram of POMS Delivery](image1)

**Figure A 1.8 : Entity Relationship Diagram of POMS Delivery**

6.3. Entity Relationship Diagram

![Entity Relationship Diagram of POMS](image2)

**Figure A 1.9 : Entity Relationship Diagram of POMS**

7. Database

The database will be used for read only purpose.
APPENDIX 2

SAMPLE OF SOFTWARE DESIGN DOCUMENT (SDD)

<table>
<thead>
<tr>
<th>Application Name</th>
<th>Post Office Management System (POMS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>e-Governance</td>
</tr>
<tr>
<td>Technology</td>
<td>C#.NET</td>
</tr>
</tbody>
</table>

POST OFFICE MANAGEMENT SYSTEM

VER 2.0

SOFTWARE DESIGN DOCUMENT

Copyright ©Trimentus. All rights reserved. No part of this publication may be reproduced, photocopied, stored in a retrieval system, or transmitted without the express prior consent of Trimentus, Chennai, India.

Trimentus makes no representations or warranties with respect to the contents or use of this document, and specifically disclaims any express or implied warranties of merchantability for any particular purpose. Trimentus reserves the right to revise this document and make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.
1. INTRODUCTION

This document describes the design for the Post Office will help in automating functions of the administration department. It helps in reducing the time spent in record keeping and the work can be carried out effectively. The searching of records in future will also become easy. The redundancy in the data due to manual data will also be tackled. The Post office will be able to access the personal information of each customer easily.

The administration department will also be able to add new branches of post office, staffs, customers, set the rates for the courier or any other parcel based on duration, weight, distance, and type of delivery.

2. MODULE DESCRIPTION

The First module is named as the Login Form in which only the administrator, staff and customer are authenticated. When the users are authenticated they enter into the next module the Main Menu. This varies based on the type of user as only the admin is allowed to make updations and the staff and customers are restricted to certain areas.

In common all can view the charges set for various factors like weight, duration/ days, type of delivery, distance etc…

Each one can view and edit his own profile and also edit it. They can also log out.

3. SYSTEM DESIGN DESCRIPTION

This project is being designed using an incremental approach. There are three primary stages to the design development. The ER diagrams are mentioned in the SRS.

4. DATABASE DESIGN

1. Table: New User

Table A 2.1 : New User Creation Database Table

<table>
<thead>
<tr>
<th>Username</th>
<th>nvarchar(20)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password</td>
<td>nvarchar(10)</td>
</tr>
<tr>
<td>UserType</td>
<td>nvarchar(20)</td>
</tr>
<tr>
<td>Secretquestion</td>
<td>nvarchar(100)</td>
</tr>
<tr>
<td>Secretanswer</td>
<td>nvarchar(100)</td>
</tr>
<tr>
<td>Dob</td>
<td>varchar(35)</td>
</tr>
<tr>
<td>Id</td>
<td>int</td>
</tr>
</tbody>
</table>
2. **Table: Customer**

   **Table A 2.2 : Customer Creation Database Table**

<table>
<thead>
<tr>
<th>Name</th>
<th>nvarchar(20)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phno</td>
<td>bigint</td>
</tr>
<tr>
<td>Address</td>
<td>nvarchar(30)</td>
</tr>
<tr>
<td>City</td>
<td>nvarchar(15)</td>
</tr>
<tr>
<td>State</td>
<td>nvarchar(20)</td>
</tr>
<tr>
<td>Country</td>
<td>nvarchar(30)</td>
</tr>
<tr>
<td>Pin</td>
<td>bigint</td>
</tr>
<tr>
<td>Userid</td>
<td>bigint</td>
</tr>
<tr>
<td>Cid</td>
<td>nvarchar(10)</td>
</tr>
<tr>
<td>Id</td>
<td>int</td>
</tr>
</tbody>
</table>

3. **Table: Delivery Type**

   **Table A 2.3 : Delivery Type Creation Database Table**

<table>
<thead>
<tr>
<th>Type</th>
<th>nvarchar(25)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charges</td>
<td>bigint</td>
</tr>
</tbody>
</table>

4. **Table: Weight**

   **Table A 2.4 : Weight Creation Database Table**

<table>
<thead>
<tr>
<th>Weightfrom</th>
<th>Bigint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weightto</td>
<td>bigint</td>
</tr>
<tr>
<td>Charges</td>
<td>bigint</td>
</tr>
</tbody>
</table>

5. **Table: Distance**

   **Table A 2.5 : Distance Creation Database Table**

<table>
<thead>
<tr>
<th>Distancefrom</th>
<th>Bigint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distanceto</td>
<td>bigint</td>
</tr>
<tr>
<td>Charges</td>
<td>bigint</td>
</tr>
</tbody>
</table>
6. **Table: Duration**

Table A 2.6 : Duration Creation Database Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Daysfrom</td>
<td>Int</td>
</tr>
<tr>
<td>Daysto</td>
<td>Int</td>
</tr>
<tr>
<td>Charges</td>
<td>bigint</td>
</tr>
</tbody>
</table>

7. **Table: PostOffice**

Table A 2.7 : Post Office Creation Database Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poname</td>
<td>nvarchar(20)</td>
</tr>
<tr>
<td>Address</td>
<td>nvarchar(50)</td>
</tr>
<tr>
<td>Poid</td>
<td>nvarchar(10)</td>
</tr>
<tr>
<td>Id</td>
<td>Int</td>
</tr>
</tbody>
</table>

8. **Table: SetDistance**

Table A 2.8 : SetDistance Creation Database Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FromPlace</td>
<td>nvarchar(20)</td>
</tr>
<tr>
<td>ToPlace</td>
<td>nvarchar(20)</td>
</tr>
<tr>
<td>Distance</td>
<td>bigint</td>
</tr>
</tbody>
</table>

9. **Table: Delivery**

Table A 2.9 : Delivery Creation Database Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cid</td>
<td>nvarchar(10)</td>
</tr>
<tr>
<td>Toname</td>
<td>nvarchar(20)</td>
</tr>
<tr>
<td>Toadd</td>
<td>nvarchar(25)</td>
</tr>
<tr>
<td>Tocity</td>
<td>nvarchar (20)</td>
</tr>
<tr>
<td>Tostate</td>
<td>nvarchar (30)</td>
</tr>
<tr>
<td>Topin</td>
<td>bigint</td>
</tr>
<tr>
<td>Toph</td>
<td>bigint</td>
</tr>
<tr>
<td>Typeofdeli</td>
<td>nvarchar(25)</td>
</tr>
</tbody>
</table>
APPENDIX 3

SAMPLE OF SOURCE CODE

<table>
<thead>
<tr>
<th>Application Name</th>
<th>Post Office Management System (POMS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>e-Governance</td>
</tr>
<tr>
<td>Technology</td>
<td>C#.NET</td>
</tr>
</tbody>
</table>

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="PostOffice.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Untitled Page</title>
</head>

<asp:MenuItem Text="Add New Details" Value="New Item">
<asp:MenuItem NavigateUrl="/PostOffice.aspx" Text="Post Office" Value="New Item"></asp:MenuItem>
<asp:MenuItem NavigateUrl="/CustomerInformation.aspx" Text="Customer" Value="Customer"></asp:MenuItem>
<asp:MenuItem NavigateUrl="/Delivery.aspx" Text="Delivery" Value="Delivery"></asp:MenuItem>
<asp:MenuItem Text="Settings" Value="New Item">
<asp:MenuItem Text="Charges" Value="New Item">
<asp:MenuItem NavigateUrl="/Weight.aspx" Text="Weight" Value="Weight"></asp:MenuItem>
</asp:MenuItem>
</asp:MenuItem>
</html>
<asp:MenuItem NavigateUrl="~/Distance.aspx" Text="Distance" Value="Distance"></asp:MenuItem>
<asp:MenuItem NavigateUrl="~/DeliveryType.aspx" Text="Delivery Type" Value="Delivery Type"></asp:MenuItem>
<asp:MenuItem NavigateUrl="~/Duration.aspx" Text="Duration" Value="Duration"></asp:MenuItem>
<asp:MenuItem NavigateUrl="~/SetDistance.aspx" Text="Set Distance" Value="Set Distance"></asp:MenuItem>
<asp:MenuItem NavigateUrl="~/ShowChargesforWeight.aspx" Text="Weight" Value="Weight"></asp:MenuItem>
<asp:MenuItem NavigateUrl="~/ShowChargesfordistance.aspx" Text="Distance" Value="Distance"></asp:MenuItem>
<asp:MenuItem NavigateUrl="~/ShowChargesfortod.aspx" Text="Delivery Type" Value="Delivery Type"></asp:MenuItem>
<asp:MenuItem NavigateUrl="~/ShowChargesforDuration.aspx" Text="Duration" Value="Duration"></asp:MenuItem>
<asp:MenuItem NavigateUrl="~/Showdistanceset.aspx" Text="Distance details" Value="New Item"></asp:MenuItem>
<asp:MenuItem NavigateUrl="~/MyProfile.aspx" Text="My Profile" Value="My Profile"></asp:MenuItem>
<asp:MenuItem NavigateUrl="~/Logout.aspx" Text="Log out" Value="Log out"></asp:MenuItem>
</Items>
APPENDIX 4

SAMPLE OF CODE REVIEW/TEST RECORD

<table>
<thead>
<tr>
<th>Application Name</th>
<th>Post Office Management System (POMS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>e-Governance</td>
</tr>
<tr>
<td>Technology</td>
<td>C#.NET</td>
</tr>
</tbody>
</table>

REVIEW / CODE WALKTHROUGH REPORT

<table>
<thead>
<tr>
<th>Review Report (RR/CWT) No.</th>
<th>Date</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td></td>
<td>14/11/2008</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Department Code</th>
<th>Start Time</th>
<th>End Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>RL</td>
<td>10:30 AM</td>
<td>12:00 PM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project Name</th>
<th>Review On</th>
</tr>
</thead>
<tbody>
<tr>
<td>POMS</td>
<td>14/11/2008</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total Man Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:30 Mins</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reviewer(s)</th>
<th>Venue</th>
<th>Reference Docs*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kumaravel KV</td>
<td>Trimentus Technologies</td>
<td>SRS 1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No of Problems</th>
<th>High</th>
<th>Medium</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem Closing Date</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Verifier**</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Findings List

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Problem Description</th>
<th>Severity (H/M/L)</th>
<th>Root Cause</th>
<th>Corrective Action</th>
<th>Attempt &amp; Status</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;Loading&quot; takes time</td>
<td>L</td>
<td>Requirements</td>
<td>Updated</td>
<td>Open</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Separate code is required for row editing, updating, deleting etc</td>
<td>M</td>
<td>Requirements</td>
<td>Corrected</td>
<td>Open</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Problems with windows and server authentication connections.</td>
<td>H</td>
<td>Coding</td>
<td>Updated</td>
<td>Open</td>
<td></td>
</tr>
</tbody>
</table>
Note:

1. Format of 'RR No.' is RR or CWT/Total No of Review or CWT/No of review in a particular doc/No of Problems. For Eg 1. RR/21/03/07 out of which 07 is the number of problems, 2. CWT/37/01/03.

2. 'Review On' is the Document name if it is Review or Form name(should be with Module Name) if it is CWT

3. 'Reviewer(s)' is All reviewers names

4. On or Before the Problem Closing date all findings should be closed

5. Severity High is for Major errors which effect the whole project, Medium is Minor error which can be rectified easily, & Low is Cosmetic error which will not affect the project

6. 'Attempt & Status' is how many times the Fixed part of the problem is Verified by the Verifier & its status (C for Closed, P for Under Process, R for Resolved & O for Open). Eg. 2 & O means 2 times it is verified & still it is open

7. If no finding is there then also this form should be filled up stating NIL in the 'Finding List'

* All the Docs used for Reviewing or Code walk through

** After Fixing the Findings this has to verify the Doc or Form
APPENDIX 5

DATA COLLECTION TOOLS

5.1 SAMPLE TEMPLATE: PROJECT PLAN

ABC Development

Project Plan
Purpose

This document addresses

- Life Cycle Model to be adopted for ABC project development, phases and the process to be followed
- Deliverables of the project, roles and responsibilities of people involved
- Configuration Management for the project

Objectives

Objectives of this document are to

- Explain project specific Life Cycle Model derived from QMS
- Process that shall be followed for the ABC project development.
- To plan and ensure the development of the ABC Software and to control changes to the software or to any of its components.

Scope

Applicable for projects satisfying the following criteria.

- ABC Development project of type low (Size less than 3000 Lines of Code).

Abbreviations and Acronyms

QMS       Quality Management System
CM        Configuration Management

Approvals and Authorizations

<table>
<thead>
<tr>
<th>Designation</th>
<th>Name</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author</td>
<td>Project Coordinator</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Project Coordinator’s name</td>
<td></td>
</tr>
<tr>
<td>Approved by</td>
<td>Quality Coordinator</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Quality Coordinator’s name</td>
<td></td>
</tr>
<tr>
<td>Authorised by</td>
<td>SBU Head or Practice Head</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SBU head’s Name or Practice Head’s name</td>
<td></td>
</tr>
</tbody>
</table>
Amendment Record

<table>
<thead>
<tr>
<th>No</th>
<th>Date</th>
<th>Section(s)</th>
<th>Issue No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Life Cycle Model

![Project Lifecycle Diagram]

**Figure A 5.1 : Project Lifecycle**

Dependencies

**Table A 5.1 : Project Dependencies**

<table>
<thead>
<tr>
<th>S.No</th>
<th>Dependency</th>
<th>Responsibility</th>
<th>Availability (date)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Test equipment</td>
<td>Customer</td>
<td>DD/mm/yy</td>
</tr>
<tr>
<td>2.</td>
<td>Interface Modules</td>
<td>,,</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>Documentation on ABCXX</td>
<td>,,</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>Enhanced XXXXX ABC product</td>
<td>Product Team Representative</td>
<td></td>
</tr>
</tbody>
</table>
Software Tools

Table A 5.2 : Project Software Tools

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Name of the Tool (including development tools)</th>
<th>Version/Release</th>
<th>Usage</th>
<th>User Licenses</th>
<th>Tools Database</th>
<th>Re-Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Visual Source Safe (VSS)</td>
<td>7.0</td>
<td>Software Configuration Management</td>
<td>Enterprise license</td>
<td>Demo, Help and validation report dated ....</td>
<td></td>
</tr>
</tbody>
</table>

Risk Management

Table A 5.3 : Project Risk Analysis

<table>
<thead>
<tr>
<th>S. No</th>
<th>Risks</th>
<th>Mitigation Plan</th>
<th>Resp.</th>
<th>Seveity</th>
<th>Probability</th>
<th>Limit</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Source code product may have bugs.</td>
<td></td>
<td>Medium</td>
<td>50%</td>
<td>High</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>Customer supplied modules may have defects.</td>
<td></td>
<td>High</td>
<td>50%</td>
<td>High</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Contingency Plan

<Action to be taken to mitigate risk identified as above should be recorded here>

Risk Tracking

Risks identified in project plan shall be tracked during team meetings and end phase meetings and corrective actions, if any, shall be taken.
Project Schedule

Date of project commencement – dd Month yyyy

Estimated date of project closure – dd Month yyyy. The detailed project schedule will be maintained by the project co-ordinator in the project file. Microsoft Project, is used to create and track the detailed project schedule.

Quality Assurance Plan

Schedule of Quality Control Activities

Table A 5.4 : Project Review Plan

<table>
<thead>
<tr>
<th>Activity</th>
<th>Start Date</th>
<th>End Date</th>
<th>No. of Reviews</th>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Requirement Specifications review</td>
<td>dd Month yyyy</td>
<td>dd Month yyyy</td>
<td></td>
<td>Person designated by Project Co-ordinator but not the author, Customer</td>
</tr>
<tr>
<td>Design Review</td>
<td>dd Month yyyy</td>
<td>dd Month yyyy</td>
<td></td>
<td>Person designated by project coordinator, but not the author</td>
</tr>
<tr>
<td>Code Review</td>
<td>dd Month yyyy</td>
<td>dd Month yyyy</td>
<td></td>
<td>Person designated by Project Co-ordinator but not the author</td>
</tr>
<tr>
<td>System test plan and cases review</td>
<td>dd Month yyyy</td>
<td>dd Month yyyy</td>
<td></td>
<td>Person designated by Project Co-ordinator but not the author</td>
</tr>
<tr>
<td>System testing</td>
<td>dd Month yyyy</td>
<td>dd Month yyyy</td>
<td></td>
<td>Team member designated by project coordinator</td>
</tr>
<tr>
<td>(Unit/Integration/System) Test results review</td>
<td>dd Month yyyy</td>
<td>dd Month yyyy</td>
<td></td>
<td>Person designated by Project Co-ordinator but not the tester</td>
</tr>
</tbody>
</table>

TOTAL NUMBER OF REVIEWS PLANNED
Configuration Library

Table A 5.5 : Project Configuration Library

<table>
<thead>
<tr>
<th>Configuration Item</th>
<th>Location (sub-directory with respect to /home/&lt;proj&gt;)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Documents</strong></td>
<td></td>
</tr>
<tr>
<td>Project Document</td>
<td>docs/plan</td>
</tr>
<tr>
<td>Requirements Specification Document</td>
<td>docs/srs</td>
</tr>
<tr>
<td>Design document</td>
<td>docs/ddd</td>
</tr>
<tr>
<td>System Test plans and Test Cases</td>
<td>docs/stp</td>
</tr>
<tr>
<td>Release notes</td>
<td>docs/release</td>
</tr>
<tr>
<td><strong>Software components</strong></td>
<td></td>
</tr>
<tr>
<td>Make files for the project</td>
<td>code/com</td>
</tr>
<tr>
<td>Source code for each module</td>
<td>code/&lt;module-name&gt;/src</td>
</tr>
<tr>
<td><strong>Testing utilities</strong></td>
<td></td>
</tr>
<tr>
<td>Source code for test utilities</td>
<td>tst/&lt;test-type&gt;/src1</td>
</tr>
<tr>
<td>Test scripts</td>
<td>tst/&lt;test-type&gt;/scripts</td>
</tr>
<tr>
<td>Test data</td>
<td>tst/&lt;test-type&gt;/data</td>
</tr>
<tr>
<td><strong>Others</strong></td>
<td></td>
</tr>
<tr>
<td>External Libraries used</td>
<td>ext/lib</td>
</tr>
<tr>
<td><strong>Project specific tools</strong></td>
<td></td>
</tr>
<tr>
<td>name of the tool must be specified here</td>
<td>Tools</td>
</tr>
</tbody>
</table>

Quality records are maintained in the following locations.

---

1 <test-type> is ut – unit test, it – integration test, st – system test or at - acceptance test
Table A 5.6 : Project Quality Records

<table>
<thead>
<tr>
<th>Item</th>
<th>Location (sub-directory with respect to /home/&lt;proj&gt;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Item List</td>
<td>Cilist</td>
</tr>
<tr>
<td>Test records</td>
<td>tst/&lt;test-type&gt;/testrecs</td>
</tr>
<tr>
<td>Review records</td>
<td>Rvw</td>
</tr>
</tbody>
</table>

Releases

- A list of releases along-with their status (ship-authorised or not) and list of customers to whom the releases are distributed, is maintained in the file /home/<proj>/release/relinfo.txt.

- Configuration items comprising releases are listed in the files /home/<proj>/<release-number>/rellist.txt.

File Naming and Version Identification

The version identification scheme followed for source files, header files, make files and documents is x.y. The version identification scheme followed for releases is x.y.z.p.

Source files, header files and make files will be assigned filenames in accordance with the Coding Guidelines.

The following scheme will be followed for document identification:

Table A 5.7 : Project Document Naming Convention

<table>
<thead>
<tr>
<th>Requirements Specification Document</th>
<th>&lt;proj-id&gt;rs001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design document</td>
<td>&lt;proj-id&gt;dd001</td>
</tr>
<tr>
<td>System Test plans and Test Cases</td>
<td>&lt;proj-id&gt;st001</td>
</tr>
</tbody>
</table>

Backups

The Computing Resources Group will create a daily-backup of the entire contents of the /home/<proj> directory. The backup will be available for retrieval for 4 weeks.
5.2 SAMPLE FORM: TEST REPORT

Test Report / Test status report

Project : 
Date of report : 
Test coordinator : 
Project coordinator : 
Test type : 
Test Environment checked on : 

Table A 5.8 : Test Environment Validation

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Test Environment*</th>
<th>Expected Set Up</th>
<th>Actual Set Up</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hardware</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Software</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Initialization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data Population</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*please add specific of test environment as applicable to your project

Test Environment : Accepted / Rejected

Test Start date :
Test Finish date :
Test cases defined :
Test cases executed :
Test cases passed :
Defects found :
Defects cleared :
Remarks :
Notes for future use :

Conclusion / Recommendation : Test complete / Test incomplete

Note 1: On completion of any test phase, the Test Coordinator produces the test report.

Note 2: During testing, the Test Coordinator produces the test status report at a frequency defined in the test plan.
5.3 SAMPLE CHECKLIST: CHECKLIST FOR REVIEW OF SOURCE CODE

Checklist for Review of Code

Introduction

Scope

This document provides a checklist of questions which a reviewer could use before, during and at the end of a code review. The document attempts to be complete by itself but brief at the same time so that reading the document does not distract the reviewer from the primary objective of reviewing the code. There are places where this checklist overlaps with a design checklist, the reason being that the person reviewing the code is not assumed to be the same person who reviewed the design.

Pre-review checklist

Practical considerations

1. Have you been provided with a design document to understand the code? Is the design document up-to-date (latest version)?
2. Does the design document explain
   - program architecture?
   - module breakup and functionality?
   - task breakup?
   - each routine in pseudo-code form?
3. Has the code been compiled with strict warning and error checking enabled (-Wall and -Werror options); have all compilation warnings and errors been removed?
4. Has the code been compiled with a C++ compiler in addition to a C compiler?

Program (product) architecture

1. Is the overall organisation of the code clear, including a good architectural overview and justification?
2. Are modules well defined, including their functionality and interfaces to other modules? Is the task/process breakup of the modules clear?
3. Are all the functional requirements covered sensibly, by neither too few nor too many modules?
4. Is the top-level design independent of the OS/environment which will be used?
5. Is the architecture designed to accommodate likely changes?
6. Does the architecture describe how reused code will be made to conform to other architectural objectives?
7. Does the whole architecture hang together conceptually?
8. Are all major data structures described and justified?
9. Are data structures belonging to every module localised suitably so that they are accessed/modified by well-defined access routines?
10. Is there a well-defined strategy to interface with external entities?
11. Are all interfaces modularised so that changes will not affect the rest of the code?
12. Are memory usage estimates and a strategy for memory management described and justified?
13. Does the architecture set space and speed budgets for each module?
14. Is a strategy for handling data (strings) described and are storage estimates provided?
15. Is a coherent error handling strategy provided?
16. Are error messages managed as a set to present a clean interface?
17. Are the motivations for the major decisions provided?
18. Are you (if you had to program/implement the system) comfortable with the architecture?

**Code Review Checklist for Standards**

**Statutory Compliance**

1. Are the approved design documents referred for coding?
2. Are the project/product acceptance and performance criteria as defined in design addressed?
3. Does the coding guidelines defined and used address the details on security, safety, standards, regulatory and statutory requirements, etc. and whether they have been incorporated in the code?

**Directory organisation**

1. Are all file names less than or equal to 8.3 characters in length?
2. Are portable files identified as separate from core protocol files?
3. Is the file/module grouping clear?
**File Organisation**

1. Does each file have a file header conforming to the standard convention?
2. Is each file organised in order - file header, type definitions, macros, prototypes, functions?
3. Are all lines within 80 characters in length?
4. Is each file less than or equal to 2000 lines in length?
5. Does each file hold code for one and only one module?

**Procedure organisation / layout**

1. Does each procedure have a header conforming to the standard convention?
2. Is the procedure organised - header, procedure name & parameters, procedure body?
3. Is the procedure defined in both ANSI format?
4. Does each procedure formatted into a maximum of 2 printed pages (excluding the header)?
5. Are arguments to each procedure formatted with only one argument per line?
6. Are all procedure names less than or equal to 32 characters in length?
7. Are procedures clearly separated by 2 blank lines?

**Code organisation / layout**

1. Are all lines less than or equal to 80 characters in length?
2. Are all identifiers less than or equal to 32 characters in length?
3. Are all macros less than or equal to 3 lines in length?
4. Does each line contain at most 1 statement/expression?
5. Are complicated expressions formatted for readability?
6. Are continuation lines indented sensibly as per the codign standards?
7. Are all alignments as per coding standards?
8. Are braces of “if”, “for” and “while” expressions in the right places (as per coding standards)?
9. Are sequential blocks separated from each other by blank lines?
10. Are comments aligned with the code they explain?
Reviewing a routine

Functional

1. Does the comment in the header explain the routine and its function clearly?
2. Are there relevant comments in the code body explaining the code?
3. Does the routine have a name which clearly defines its purpose and describes everything the routine does?
4. Is the routine required? Is its job/function clearly defined? Does it fit in neatly with the overall architecture?
5. Have all parts of the routine which would benefit from being put into routines of their own, been put into routines of their own? Have relatively independent groups of statements been moved into their own routines?
6. Does the routine have functional cohesion - doing only 1 thing and doing it well?
7. Is the routine loosely coupled with the rest of the code?

Parameters and return values

1. Are parameters in a sensible order (input-modify-output); is this order consistent with the order of parameters in similar routines?
2. Are all parameters used?
3. Do the parameters to the routine make dependencies obvious?
4. If the function has a return value, does it return a value at all exit points?
5. Does the routine use the minimum number of returns possible, unless necessary?
6. Does the routine have 7 (or fewer) parameters?

Body of the routine

1. Are any and all assumptions/dependencies (including interfaces) documented?
2. Are any algorithms used specified and explained?
3. Does the routine check validity of input data?
4. Do you thoroughly understand the code? Is it easy to understand?
5. Does the routine handle exceptions gracefully?
6. Is the routine designed to handle changes gracefully?
7. Have all debugging aids been installed so that they can be (de)activated easily?
8. Does the code check return values of all functions which it calls? Do the formal and actual parameters match for all sub-routine/function calls?
9. Is any defensive code designed to help the user rather than the programmer?
10. Does the routine ensure that nothing in the routine, other than the return values, affects code outside the routine?
11. Are related statements grouped together? Does the code make dependencies among statements obvious?
12. Are references to the same variable close together?

**Recursion**

1. Is recursion the best way to implement the routine - is it necessary?
2. Will the recursion definitely stop?
3. Is recursion limited to one routine?
4. Does the stack size support the recursion depth?

**Data and types**

1. Do you have a list of data types used along with their descriptions?
2. Does the code use a different type for each kind of data that might change?
3. Does the code avoid redefining predefined types?
4. Are the data structures simple so that they minimise complexity?

**Declaration**

1. Is each variable assigned the correct length, type and storage?
2. Are static variables clearly identified?
3. Are all declarations free of assumptions about compiler/machine specific sizes?

**Initialization**

1. Is each variable initialised?
2. Is each variable initialised close to where it is used?
3. Is each variable initialised whenever it is used?

Names
1. Does the name fully and accurately describe what the variable represents?
2. Is the name oriented towards a real-world entity represented rather than a program type?
3. Do names of types/pointers have a prefix which indicates a type/pointer?
4. Does the name conflict with a name used in any standard library?
5. Does the code avoid arbitrary/mis-leading/similar-sounding names?
6. Do all variable names have prefixes which indicate their size in bytes?

Usage
1. Do all variables have the smallest scope possible?
2. Are all global variables documented?
3. Are access routines used instead of global data?
4. Is the code free of data structures whose elements are accessed both globally as well as through access routines?
5. Are references to the same variable close together?
6. Are all variables used?
7. Is the usage of access routines versus global data consistent?
8. Are variables used only for the purpose for which they’re named?

Arrays, pointers and structures
1. Are all array references within bounds?
2. Is the code free of off-by-one errors?
3. Is the code free of references to dangling pointers?
1. Is the code free of assumptions about alignment of elements within structures?
Numbers, characters & strings

2. Does the code avoid magic numbers/strings?
3. Are all constants defined and used rather than placing the numbers in the code?
4. Are type conversions obvious?

Comparisons, Computations & Expressions

1. Is the code free of comparisons/computations involving variables of different sizes/types (mixed-mode)?
2. Is the code free of assumptions about the precedence of operators in expressions?
3. Are macros used to simplify readability whenever an expression is used repeatedly?
4. Do all expressions use the correct operators?
   (Some common mistakes - use of = in place of ==
   - use of || in place of && etc...)

Conditionals

If statements

1. Is the conditional check and the resultant path through the code clear?
2. Could there be a third possibility other than the if and the else?
3. Are the if and else clauses used correctly - not reversed?
4. Does the normal case follow the if rather than the else?
5. Are any complicated tests encapsulated in (boolean) function calls?
6. Have the number of decision counts been kept to the minimum required?
7. If the number of decision counts is large (6 or more) would it be simpler to have a nested if chain?
8. Are numbers, characters and pointers compared to 0/NULL/FALSE explicitly?
9. Are boolean expressions stated positively?

If-Then-ElseIf chains

1. Are the most common cases tested first?
2. Are all cases covered?
3. Is the chain better than using a case statement?
4. Is the chain short enough to view all at once?
5. Are the nesting levels 3 or fewer in number?

Case statements
1. Are cases ordered meaningfully?
2. Are the actions for each case simple - calling other routines if necessary?
3. Does the default clause detect and report unexpected cases?
4. Does the end of each case have a break?
5. Are intentional fall-throughs commented clearly?

goto
1. Is the goto used only as a last resort to make the code more readable/maintainable?
2. If used for efficiency, has the gain in efficiency been measured and documented?
3. Are gotos limited to one per routine?
4. Does the goto go forward, not backward?

Loops
1. Is the loop non-empty?
2. Is initialization code directly before the loop?
3. Is the loop short enough to view all at once?
4. Is a for loop used when the number of iterations is known?
5. Is a while loop used when the number of iterations is unknown?
6. Is the code free of unintended infinite loops?
7. Is the loop header reserved for loop control?
8. Does the loop index have a meaningful name?
9. Are housekeeping chores grouped, at the beginning/end of the loop?
10. Is the loop well-defined - does it perform only one function?
11. Is the loop’s termination condition clear and obvious?
12. Does the code inside a for loop avoid monkeying with the loop index unnecessarily?
13. Are the nesting levels 3 (or fewer) in number? Has the nesting been kept to the minimum required?

**Input/Output**

1. Have attributes for all files been specified correctly?
2. Have attributes for all OPEN/CREATE calls been specified correctly?
3. Have end-of-file conditions been checked explicitly?
4. Is displayed text free of spelling/grammatical errors?

**Comments**

1. Is there a short description which gives an overall view of how the code is organised?
2. Is the purpose of each file/module explained?
3. Is the source code listing self-explanatory?
4. Can someone pick up the source code listing and immediately start to understand it?
5. Do comments explain, instead of simply repeat, the code?
6. Are comments clear and correct?
7. Do comments prepare the reader for the code to follow?
8. Are all assumptions/limitations commented?
9. Are the ends of long or complex control structures been commented?

**End-of-review checklist**

1. Is the code straight-forward and does it avoid cleverness?
2. Is the code written in terms of the real-world problem domain as much as possible rather than in terms of computer science structures?
3. Is the code clearly traceable from the design document?
4. Has unused code been removed? Have all unnecessary comments been removed?
5.4. SAMPLE CHECKLIST: APPLICATION DEVELOPMENT STANDARDS

Coding Standards for C#

Cross Language Coding Rules

These rules are to followed irrespective of the language of development

File and Directory naming rules

File names and directory names should be of maximum 15 chars

The possible extensions are tabulated below:

<table>
<thead>
<tr>
<th>File type</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td>.h, .cpp, .c</td>
</tr>
</tbody>
</table>

Comment rules

Comments should not be cryptic and should not have flavor of the language

1. Comments should not be nested
2. Unutilised code should not be commented
3. See CSD_Appendix.txt for File Headers, Function headers etc.
4. The format of Review and fix comment is
5. `<RVW>TLA:yyyymmdd:comment</RVW> :: <RFX>TLA:yyyymmdd:comment</RFX>`
6. The format of fix comment is
7. `<BFX>TLA:yyyymmdd:bugref:comment</BFX>`

Identifier naming rules

1. All variables should be of the format `<Type Acronym><Variable Name>`
2. All control identifiers should be of the format `<Control Prefix><MeaningfulName>`
3. Apart from the above mentioned format, identifiers may also have their language specific prefixes
4. Avoid names that are similar or just differ in case
5. Function names should be meaningful and begin with a verb
   Example: GetTimeSheet
6. Function parameters should be prefixed as shown below:

<table>
<thead>
<tr>
<th>Parameter Type</th>
<th>Parameter Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>In Parameter</td>
<td>Pi</td>
</tr>
<tr>
<td>Out Parameter</td>
<td>Po</td>
</tr>
<tr>
<td>In Out Parameter</td>
<td>Pio</td>
</tr>
</tbody>
</table>

   Format: <Parameter Prefix>_<<Type Acronym><Variable Name>
7. All global variables should be prefixed by “g_”
   Example: g_shtRecordCount
8. Constants should be in Capitals and separated by an “_”. They should not be prefixed by a Type Acronym
   Example: NO_OF_RECORDS

**Formatting rules**

1. Be uniform in using braces and parenthesis
2. Insert 2 spaces on Tab in the Editor Options. Use Tab to indent uniformly
3. Ensure that the code is indented suitably

**Guidelines**

1. Declare variables close to usage
2. Avoid loops that execute statements at least once without a conditional check
   Example: Do...While and Do...Loop
3. Limit number of nested “if” statements (Recommended is 2 levels)
4. Use local variables unless there is a compelling reason to use global scope with in a module
5. Comment the logic not the syntax
6. Preferred file size (Recommended is 500 lines)
7. Let functions be less than a screen full (Recommended is 40 lines)
8. Be familiar with the editor’s feature like search, replace, cut paste, column marking, key stroke macros, regular expressions, indentation settings, tab settings.
C++

**Directory structure rules**

The directory structure for files should be as follows

- `<ModuleName>` - contains the module source files
- `Include` - contains the include files
- `TestProg` - contains the source files for test programs

**Include file rules**

1. Add the module directory to “include path” variable in your machine
2. Use relative paths to include files, don’t give absolute paths

Example:

```cpp
#include "\include\UtilityFunc.h"    // Use this
#include "c:\strad\fep\Fep.h"        // Don’t use this
```

**File wrapper rules**

*All source files should encapsulate their code within*

```cpp
#ifndef __<FileName><Extension>
#define __<FileName><Extension>
// Your code here
#endif // OF __<FileName><Extension>
```

Where `FileName` – stands for File Name (max 15 characters)

Extension – stands for File Extensions (h, cpp or c)

**Comment rules**

1. Single line comment should use `//` style
2. Multi line comment should use `/* */` style
3. Multi line comments should be of the following format

Example:

```cpp
/*constant used in file name declaration */
const MAX_WIDTH = 35;
/*constant used in file name declaration */
const MAX_WIDTH = 35; /* constant used in file name declaration */
const MAX_WIDTH = 35; /* constant used in file name declaration */
```
**Identifier naming rules**

1. All identifiers should live in namespace.  
Namespace are of the pattern nm<MeaningfulName>  
Classes are of the pattern c<MeaningfulName>  
Exception Classes are of the pattern c<MeaningfulName>Exception  
Templated classes are of the pattern tc<MeaningfulName>

**Table A 5.9 : Variable Type Acronyms**

<table>
<thead>
<tr>
<th>Simple Data Type</th>
<th>Type Acronym</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>Bln</td>
</tr>
<tr>
<td>Character</td>
<td>Chr</td>
</tr>
<tr>
<td>Double</td>
<td>Dbl</td>
</tr>
<tr>
<td>Enumeration</td>
<td>Enm</td>
</tr>
<tr>
<td>Float</td>
<td>Flt</td>
</tr>
<tr>
<td>Integer</td>
<td>Int</td>
</tr>
<tr>
<td>Long</td>
<td>Lng</td>
</tr>
<tr>
<td>Long Double</td>
<td>Ldb</td>
</tr>
<tr>
<td>Object</td>
<td>Obj</td>
</tr>
<tr>
<td>Short</td>
<td>Sht</td>
</tr>
<tr>
<td>Unsigned Character</td>
<td>Uchr</td>
</tr>
<tr>
<td>Unsigned Integer</td>
<td>Uint</td>
</tr>
<tr>
<td>Unsigned Long</td>
<td>Ulng</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COM Types</th>
<th>Type Acronym</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variant</td>
<td>Var</td>
</tr>
<tr>
<td>BSTR</td>
<td>Btr</td>
</tr>
<tr>
<td>Interface</td>
<td>Inf</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Collection Type</th>
<th>Type Acronym</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>Arr</td>
</tr>
<tr>
<td>List</td>
<td>Lst</td>
</tr>
<tr>
<td>Vector</td>
<td>Vec</td>
</tr>
<tr>
<td>String</td>
<td>Str</td>
</tr>
<tr>
<td>Map</td>
<td>Map</td>
</tr>
<tr>
<td>Set</td>
<td>Set</td>
</tr>
</tbody>
</table>
**Format:**

Simple data types: `<DataType Acronym>`

Collection Types  : `<CollectionType Acronym><DataType Acronym>`

Example:

```c
unsigned short  ushtCounter;
unsigned char   arruchrFileName[MAX_WIDTH];
```

Prefix ‘p’ before variable type to indicate pointer type (Add prefix ‘p’ for each indirection)

Example:

```c
Declaration                           Variable Used as
unsigned long*                       Pointer
pulngEmployeeId;
unsigned long**                      Pointer to Pointer
ppulngEmployeeId;
unsigned long***                     Pointer to Pointer to
pppulngEmployeeId;
```

Derive naming along the same lines for further indirection

Variable naming rules based on scope is shown below:

**Table A 5.10 : Variable Naming Scope**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Variable Name Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block/Function</td>
<td><code>&lt;Variable Type Acronym&gt;&lt;Variable Name&gt;</code></td>
<td>char arrchrName[MAXLENGTH ];</td>
</tr>
<tr>
<td>Class</td>
<td>m_&lt;Variable Type Acronym&gt;&lt;Variable Name&gt;</td>
<td>char* m_pchrActive;</td>
</tr>
<tr>
<td>Global</td>
<td>g_&lt;Variable Type Acronym&gt;&lt;Variable Name&gt;</td>
<td>cCount</td>
</tr>
<tr>
<td></td>
<td></td>
<td>g_objCounter ;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Where cCount is a class</td>
</tr>
</tbody>
</table>
Static Variables naming rules

s<scope specifier>_<Variable Type Acronym><Variable Name>

Example:

```
static cCount sm_objCounter       - Static Class Data Member
static cCount s_objCounter       - Static Local Variable
static cCount sg_objCounter      - Static Global Variable
```

**Coding Rules**

1. The type int should never be used as its size is machine dependent. use long, short or char instead.
2. All variables should be in a finite state before use. It is expected that each variable is initialised at declaration either explicitly or via constructor.

Example:

```
short shtCounter;       - Don’t use this
short shtCounter = 0;    - Use this
```

Example:

```
const char FILENAME[] = “x.x” - Use this
const char* FILENAME = “x.x” - Don’t use this
```

3. Do not use hard coded values in the source code, even as array dimensions. Instead use defined consts or STL containers.
4. Functions should have only r-value or void as return type l-values should be returned through function Parameters
5. Function parameters , whose values should not be modified within the function, should be declared as const
6. Do not allocate memory for parameters within the function and delete it outside or vice-versa
7. Inline member functions should be defined in the class declaration
8. Base class Destructors should be virtual

**Guidelines**

**Specific to C++**

1. All the implemented code should be within a single Try-Catch
   Do not use protected data members
Have public, protected and private members of a class grouped and placed in order as follows

Example:

```cpp
class cEmployee {
    public: // all public members here
        cEmployee();
        virtual ~cEmployee();
        HRESULT Get_Name(char *po_pchrName);
    protected: // all protected members here
        cEmployee(const char *pi_pchrName, const char *pi_pchrID);

        private: // all private members here
            char* m_pchrName;
};
```

Constructors and Destructors should be the first two entries in sections (public, protected, private) where they occur.

Be familiar with programming tools like cl, link, dumpbin, grep, make, etc., and their command line options.

**Specific to COM Interfaces**

1. Avoid [out] as parameter type as it doesn't work with automation clients such as JavaScript and VBScript
2. Communication between Server components and clients should be through VARIANT/VARIANTARG data type
3. Use Safearray for accessing arrays, don't hard code or use constants for accessing array parameters to Clients from COM interface and vice-versa
4. Use _bstr_t, _variant_t classes instead of BSTR, VARIANT types inside interfaces for better performance