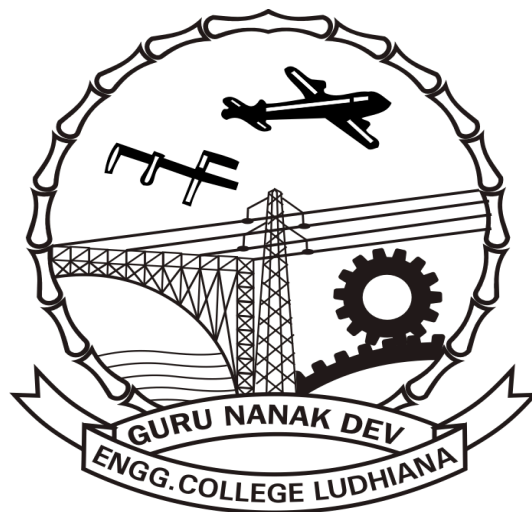


INSTRUCTION MANUAL

SOFTWARE ENGINEERING LAB (BTCS- 606)



Prepared by

**Er. Lovepreet Kaur
Assistant Professor (CSE)**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

**GURU NANAK DEV ENGINEERING COLLEGE
LUDHIANA – 141006**

DECLARATION

This Manual of Software Engineering Lab (BTCS-606) has been prepared by me as per syllabus of Software Engineering Lab (BTCS-606).

Signature

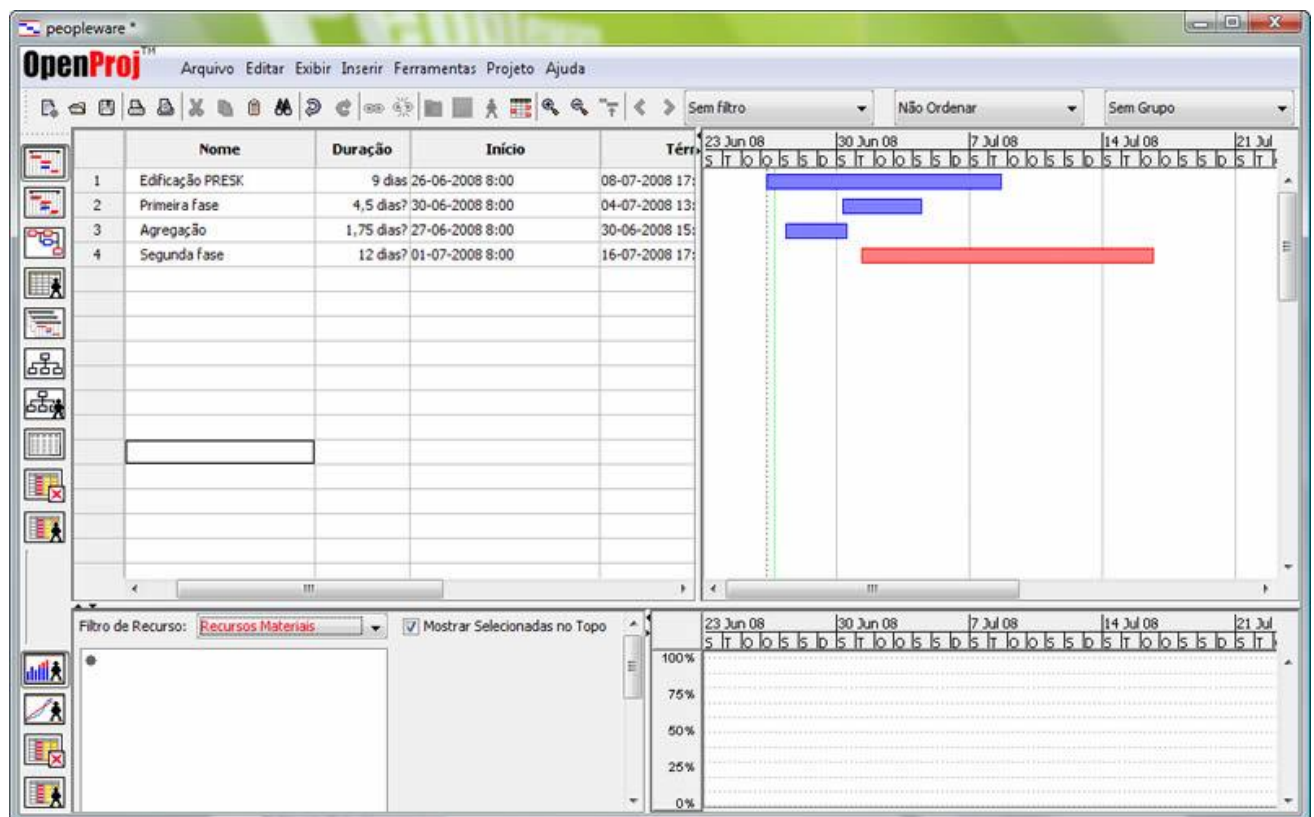
LIST OF EXPERIMENTS:

1. Study and usage of OpenProj or similar software to draft a project plan
2. Study and usage of OpenProj or similar software to track the progress of a project
3. Preparation of Software Requirement Specification Document, Design Documents and Testing Phase related documents for some problems
4. Preparation of Software Configuration Management and Risk Management related documents
5. Study and usage of any Design phase CASE tool
6. To perform unit testing and integration testing
7. To perform various white box and black box testing techniques
8. Testing of a web site

Practical 1: Study and usage of OpenProj or similar software to draft a project plan

AIM: - Introduction to OpenProj Software.

OpenProj is a free, open-source project management solution. OpenProj is ideal for desktop project management and supports opening Microsoft or Primavera files. OpenProj 1.4 is an open source desktop project management application. This application is an alternative to Microsoft Project. OpenProj provides control, tracking and management of projects. To create a new project the only field required is the Project Name on the Create New Project window and the start date of the project can be change it if you don't want to start the day that you're creating your project. You can add tasks in the Gantt diagram providing the start and end dates of each task; also you can add information to each task such the predecessors and successors tasks, resources, notes, etc. One great feature is that provides the Work Breakdown Structure (WBS) to order and control the tasks of the project for people who manages projects this is a very important tool. Another great feature is the Resources Breakdown Structure (RBS) to define the structure of the resources, teams, providers, etc. Task Usage and Resource Usage are features to control your project and provide a good track on it. The Report tool provides information about the current status of your project.



OpenProj provides tools to open at the bottom of the window and provide different views of the project at the same time; Histogram, Chart, Task Usage and Resource Usage are the tools that you can

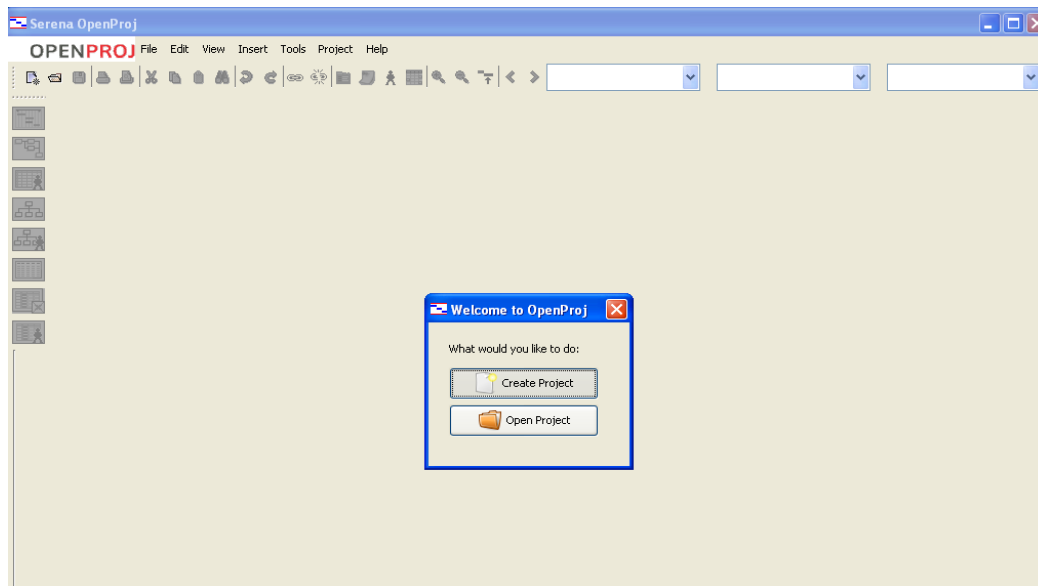
open at the bottom of the window to have a better perspective of the advance or delay on your projects.

The GUI is friendly and easy to use, very intuitive if you are get use to work managing projects. The installation process is very easy to perform and it requires Java 5 minimum or Java 6 recommended. OpenProj works on Linux, Unix, Mac or Windows platforms, and it's free. The features of OpenProj are as follows:-

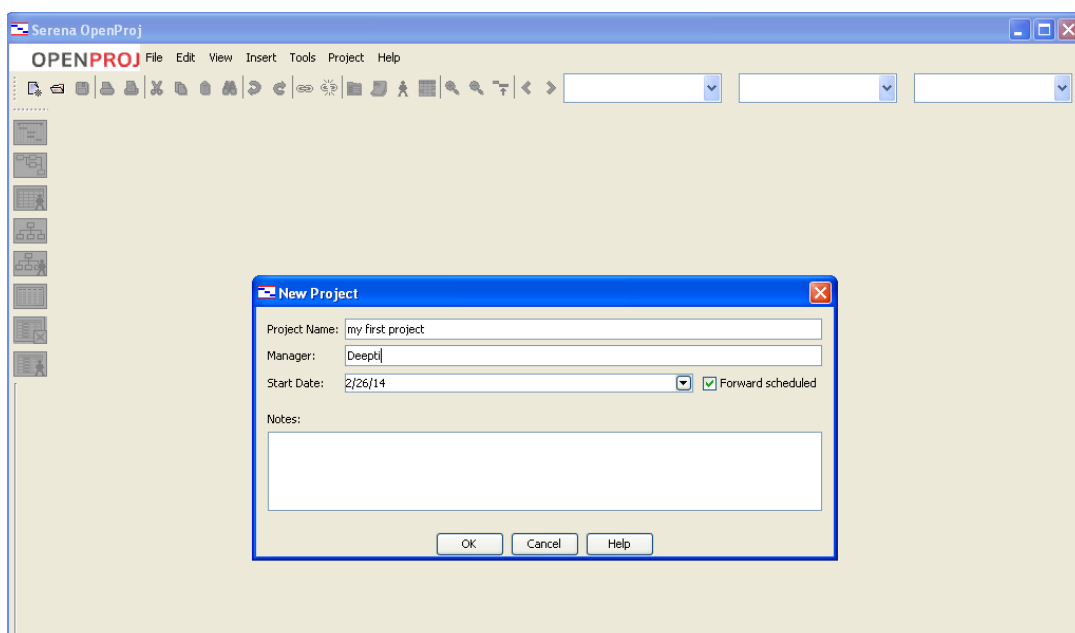
- **Gantt chart:-** A Gantt chart is a type of bar chart, developed by Henry Gantt in the 1910s, that illustrates a project schedule. Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project. Terminal elements and summary elements comprise the work breakdown structure of the project. Modern Gantt charts also show the dependency (i.e. precedence network) relationships between activities.
- **PERT graph:-** The Program (or Project) Evaluation and Review Technique, commonly abbreviated PERT, is a statistical tool, used in project management, that is designed to analyze and represent the tasks involved in completing a given project. First developed by the United States Navy in the 1950s, it is commonly used in conjunction with the critical path method (CPM).
- **Resource Breakdown Structure (RBS) chart:-** In project management, the resource breakdown structure (RBS) is a hierarchical list of resources related by function and resource type that is used to facilitate planning and controlling of project work. In some cases, a geographic division may be preferred. Each descending (lower) level represents an increasingly detailed description of the resource until small enough to be used in conjunction with the work breakdown structure (WBS) to allow the work to be planned, monitored and controlled.
- **Work Breakdown Structure (WBS) chart: -** A work breakdown structure (WBS), in project management and systems engineering, is a deliverable oriented decomposition of a project into smaller components. A work breakdown structure element may be a product, data, service, or any combination thereof.

AIM:- To create projects in OpenProj Software.

1. Having downloaded and installed the OpenProj software; start OpenProj from the start menu of Windows or via the icon on your desktop. After the license agreement, OpenProj starts with a choice of creating a new project or opening an existing one. Choose 'Create Project'



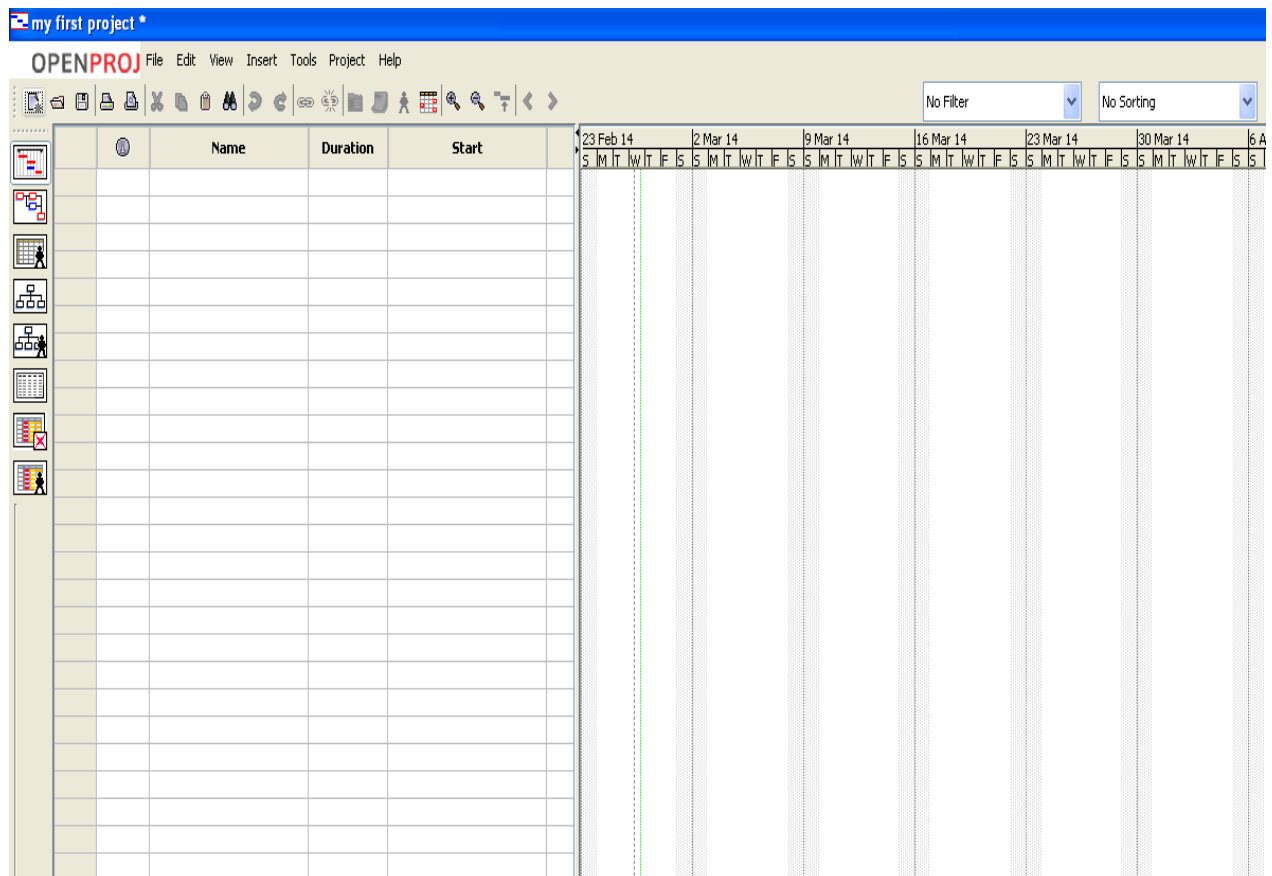
2. Enter a name for the project, the project manager's name and a start date (which can also be modified later if required). Provide a description in the Notes section as you see fit.



3. The application window will be displayed consisting of following main components:-
 - **Indicator Column:-** In this column, symbols are displayed to show the status of the corresponding tasks (see right).
 - **Task Number Column:-** Task numbers are assigned by OpenProj automatically when data is entered. The user does not enter information into this box.
 - **Time Scale.** In many views, OpenProj shows a timeline whose scale can be changed via the + and - magnifying glass icon on the menu above.
 - **Information View Type:-** The area along the left side of the window lists a number of buttons representing the available views. Gantt, Network, Resources, WBS (work

breakdown structure), RBS (resource breakdown structure), Reports, Task Usage and Resource Usage.

The lower view buttons open a split-screen where resource mapping tables and resource utilization as a graphic can be displayed. You can toggle views on and off by clicking the same button repeatedly.



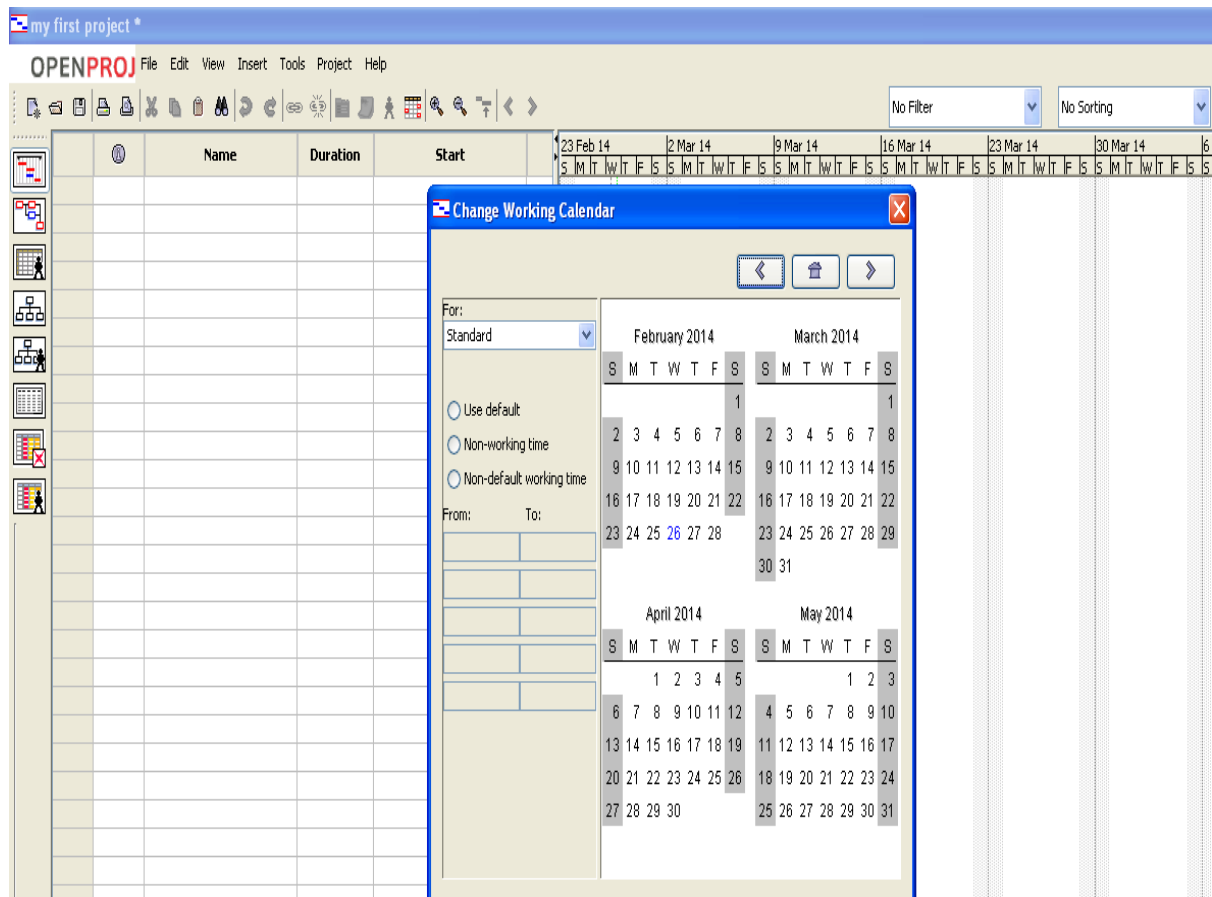
AIM:- To adjust the calendar while creating project in OpenProj Software.

For accurate tracking of your project it is essential to first set up the OpenProj calendar to reflect the working hours in your organization. OpenProj offers three default calendars:

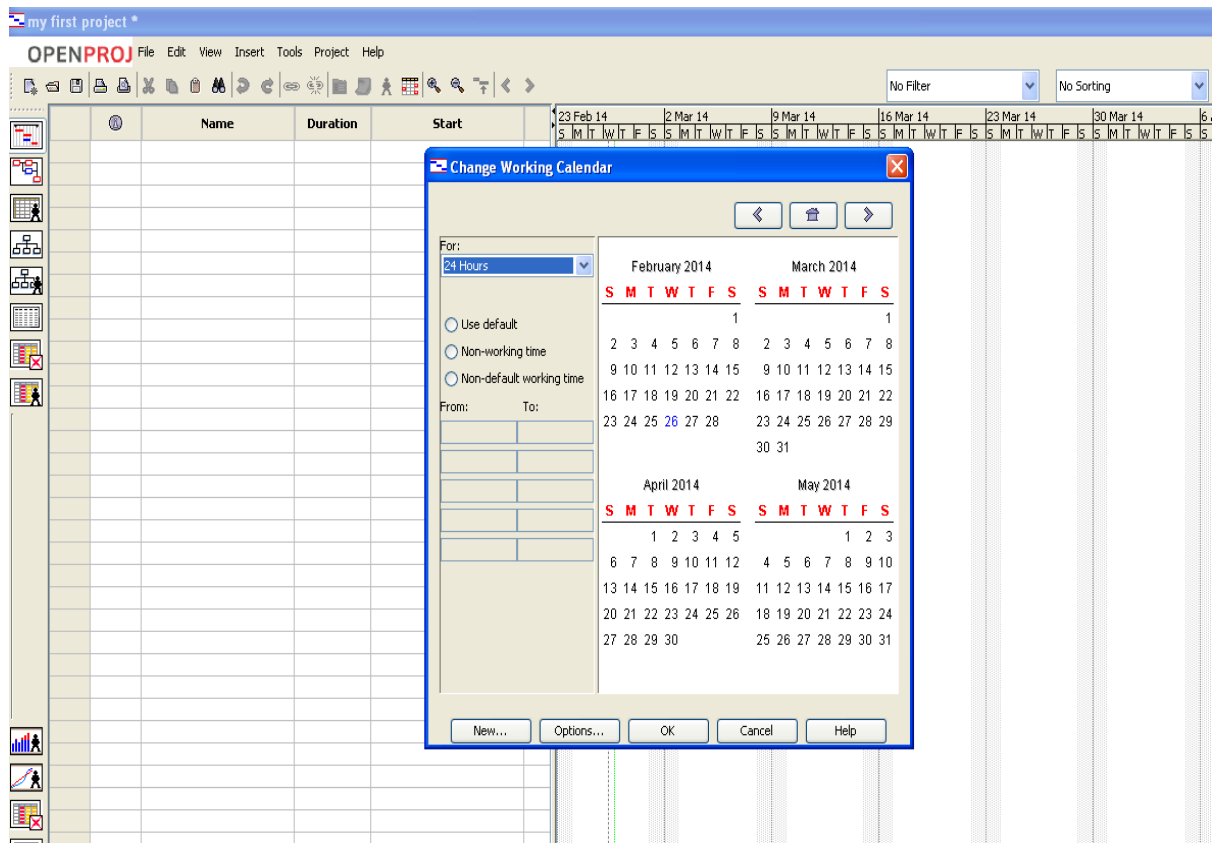
- Standard Calendar (Mon-Fri, 08:00 to 17:00 with 1hr lunch break at noon)
- Night Shift (Mon-Sat 23:00 to 08:00 with a 1hr break from 03:00 to 04:00)
- 24-Hour Calendar (24/7 schedule)

You can assign any one of these to be your project calendar; however none of the default calendars contain public holidays so it is often necessary to create your own custom calendar to reflect your company's general working hours.

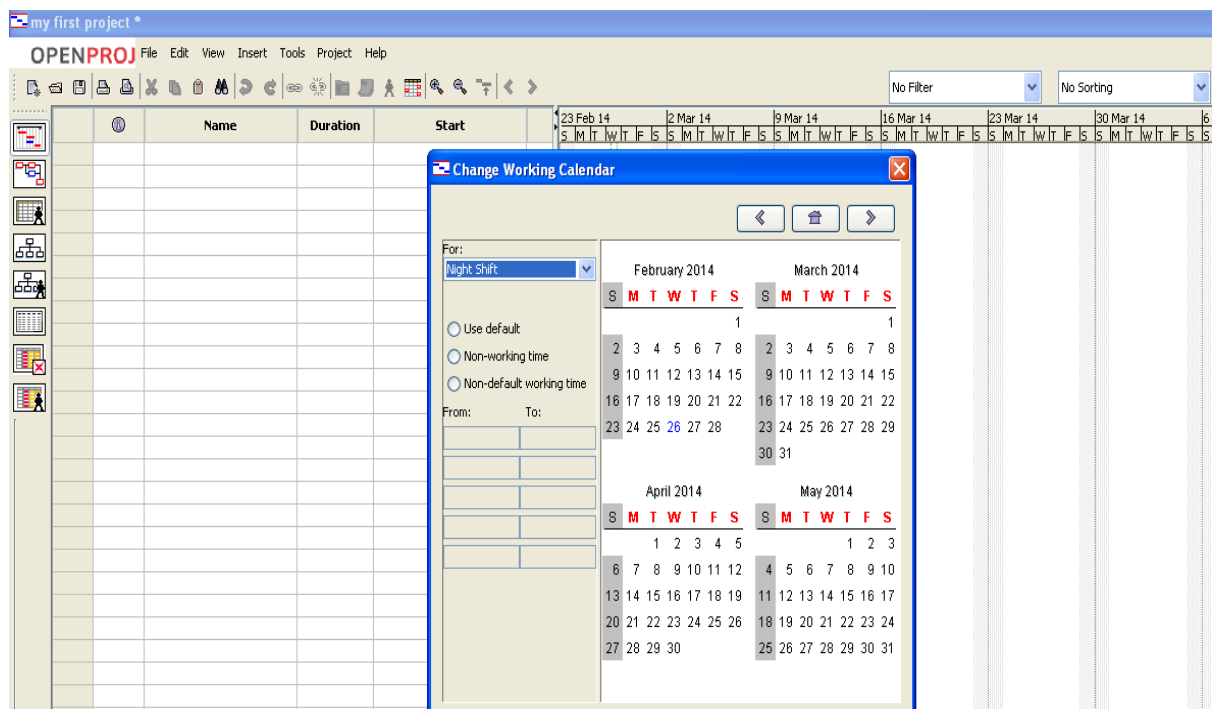
1. Open the Calendar with the menu tool Tools – Change Working Calendar .



Standard Calendar

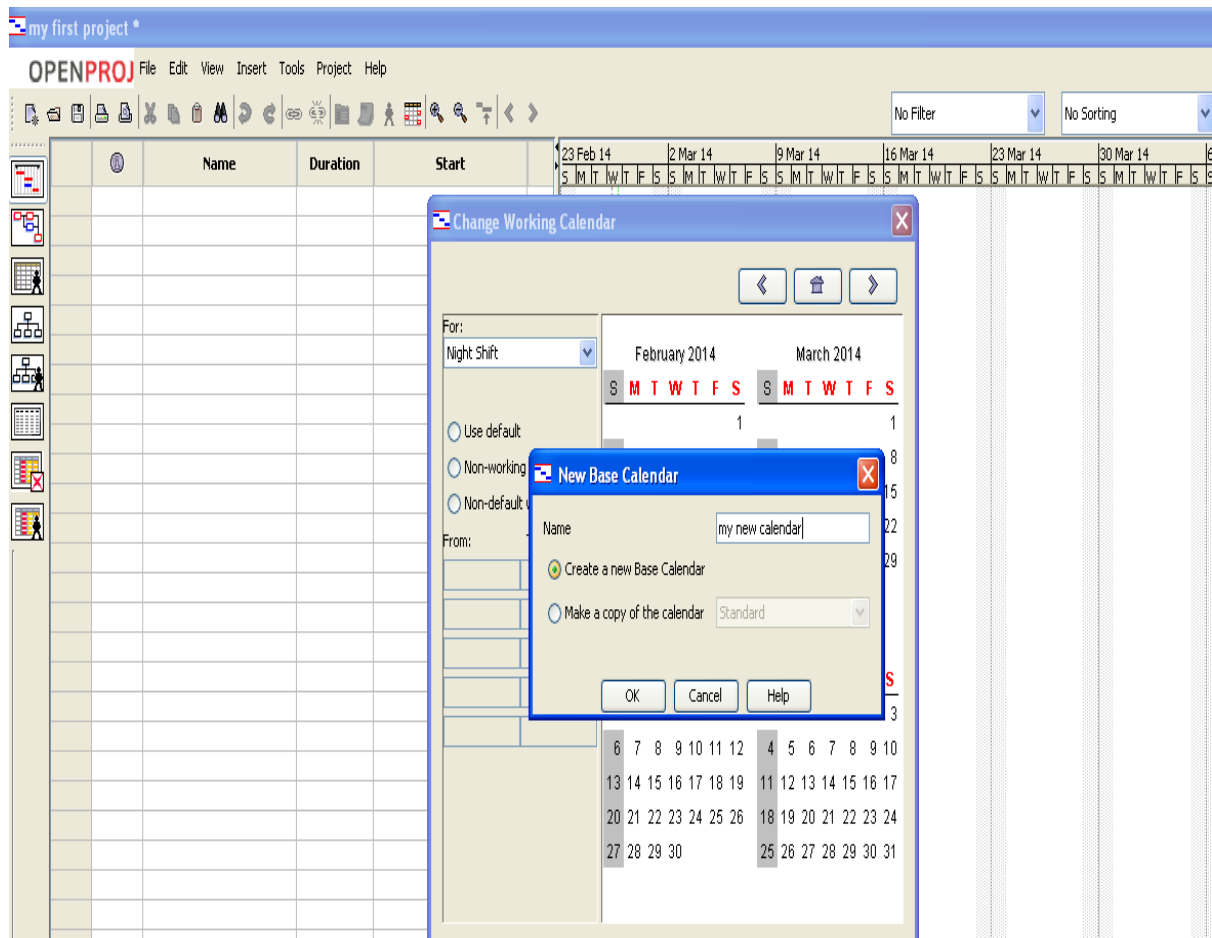


24 Hours Calendar



Night Shift Calendar

2. Click on New to create a new calendar.
3. Enter a name for your new calendar and click OK.
4. Mark the days for which you want to change the working time (e.g. for all Fridays you can click the column header 'F', or you can hold ctrl and click to select individual days). To make a global selection you would hold ctrl and click S,M,T,W,F,S in turn.
5. Click "Non-default working time" on the left and change the working hours within the fields below.
6. To mark company or public holidays, select the relevant days as above and click 'Non-working time'.
7. Edited calendar entries will appear in red.
8. Click 'Options' at the bottom to set the corresponding working hours per day, hours per week and days per month. If you do not do this, OpenProj will not schedule the tasks correctly.
9. Confirm changes by clicking OK.



10. You must now assign your custom Project calendar to your project. To do this, click Project –Project Information from the top menu and set the Base Calendar to your new custom base calendar name. This is also the screen where you enter the Project Start Date. Click close when done.

Project Information

General Statistics Notes

Name: test

Manager:

Start: 4/26/12 8:00 AM

Finish: 4/26/12 8:00 AM

☒ Forward scheduled

Priority: 500

Project Type: Other

Division:

Net Present Value: 0

Risk: 0.0

Current Date:

Status Date: 4/26/12

Base Calendar: My Project Calendar

Project Status: Planning

Expense Type: None

Group:

Benefit: 0

Close Help

OpenProj can also associate calendars to individuals and to specific tasks. In this way you can designate individual vacation periods or you can specify a task to follow a different schedule to the one in your base calendar. So, in fact you can use three different types of calendars within your project:

1. Project calendar (your base calendar for the project as defined above)
2. Resource calendar (for individuals working time and vacation days)
3. Task calendar (for tasks that do not follow the regular working hours, e.g. a task cannot start on a Friday because it would be interrupted by the weekend or maybe a task that runs continuously for 24hrs).


The calendar is stored with the particular project file, so if you start another project it is necessary to create the user defined calendar again.

Practical 2 :Study and usage of OpenProj or similar software to track the progress of a project

AIM:- To create Gantt Charts and Network charts in OpenProj Software.

When you create a new project, OpenProj by default opens the Gantt chart view on the right pane and the task input table on the left pane. You can drag the vertical line dividing the two panes to fit your monitor as you desire. To enter a task:

1. Click in the Name column in the first row.
2. Enter a name for the task.
3. Confirm the input by clicking the mouse or pressing the tab button which takes you to the duration entry field.
4. If you do not know the duration at this stage you do not have to input it. OpenProj enters a default value of 1day? which you can alter at a later stage.
5. If you have an estimate of the duration, then feel free to enter it into the cell. OpenProj automatically converts the duration to days. For example, enter 1w (1 week) and OpenProj will set the duration to 5 days. The input of 1 month would result in a display of 20 days and 1 hour would be displayed as 0.125 days (or 1/8th of a day).
6. Confirm the input by pressing Enter or clicking on the next row.
7. It is important that you enter only task names and durations at this stage. Do not be tempted to enter any other information right now. Especially not a starting date.
8. It's good practice even at this stage to enter general headings (phases) for tasks e.g "Design" and then more specific tasks underneath, but never enter a duration for a general heading.

		Name	Duration
1		Design	1 day?
2		Gather informatic	5 days
3		Analysis	10 days

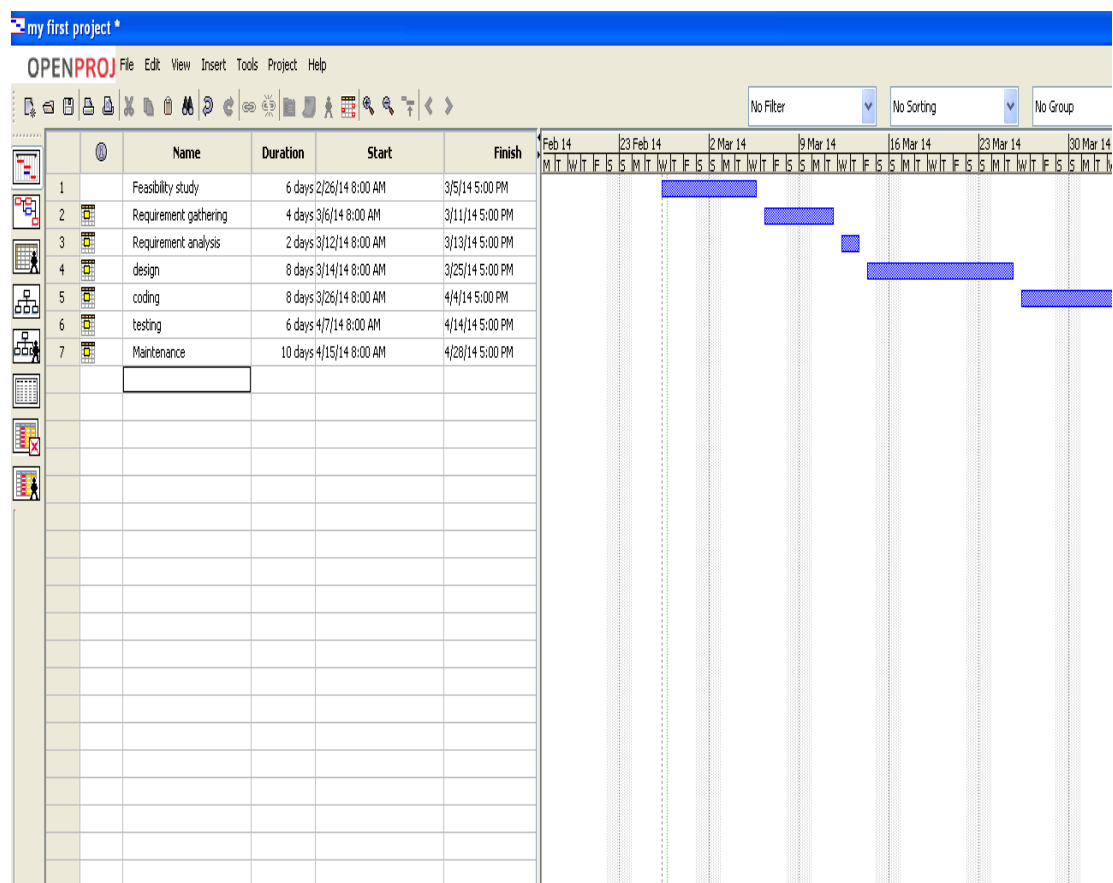
1.Gantt Charts

A Gantt chart, commonly used in project management, is one of the most popular and useful ways of showing activities (tasks or events) displayed against time. On the left of the chart is a list of the

activities and along the top is a suitable time scale. Each activity is represented by a bar; the position and length of the bar reflects the start date, duration and end date of the activity. This allows you to see at a glance:

- What the various activities are
- When each activity begins and ends
- How long each activity is scheduled to last
- Where activities overlap with other activities, and by how much
- The start and end date of the whole project

To summarize, a Gantt chart shows you what has to be done (the activities) and when (the schedule).



Gantt Chart

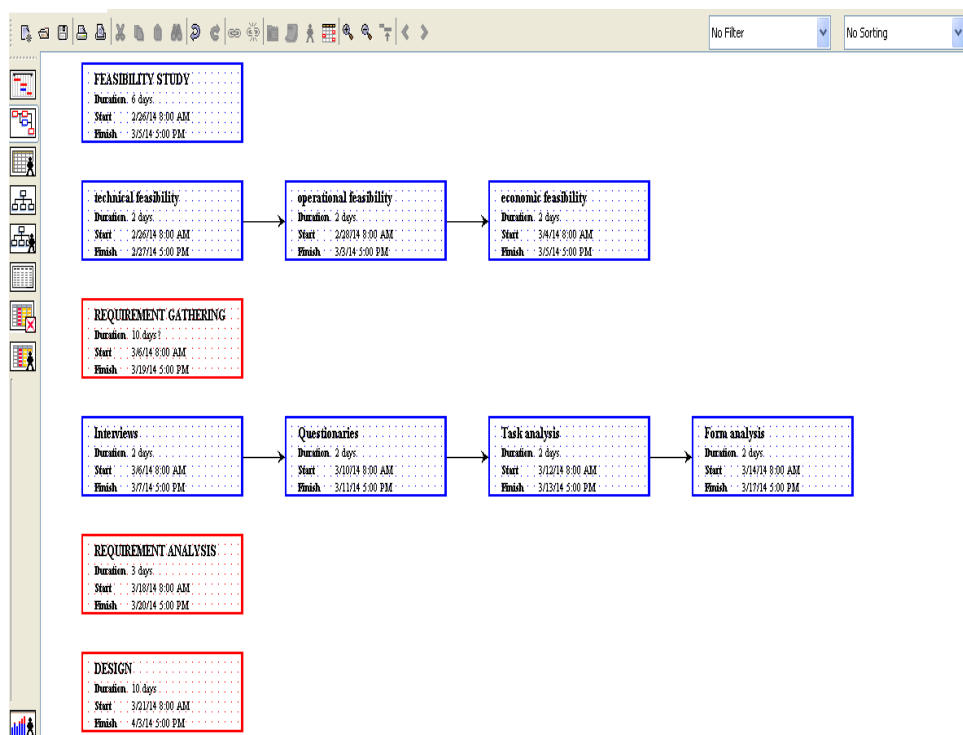
2. Network Chart

A network diagram is essentially a flow chart that includes all of the project elements and how they relate to one another. It shows parallel activities and the links between each activity. Network logic is the collection of activity dependencies that make up a network diagram for a particular project. In other words, certain tasks are dependent on one another to complete the project. This creates a logical

stream of events that will lead to completion of the project. The network diagram lets you do the following:

- Define the project's path
- Determine the sequence of tasks to be completed
- Look at the relationship between activities
- Determine the dependencies
- Make adjustments as tasks are completed
- Take a broad look at the project path and clearly see the relationships and dependencies between tasks

CLICK “View” and then “Network diagram”. It will then generate the diagram shown below.



Network Chart

AIM :- To set dependencies and establishes deadlines in OpenProj software.

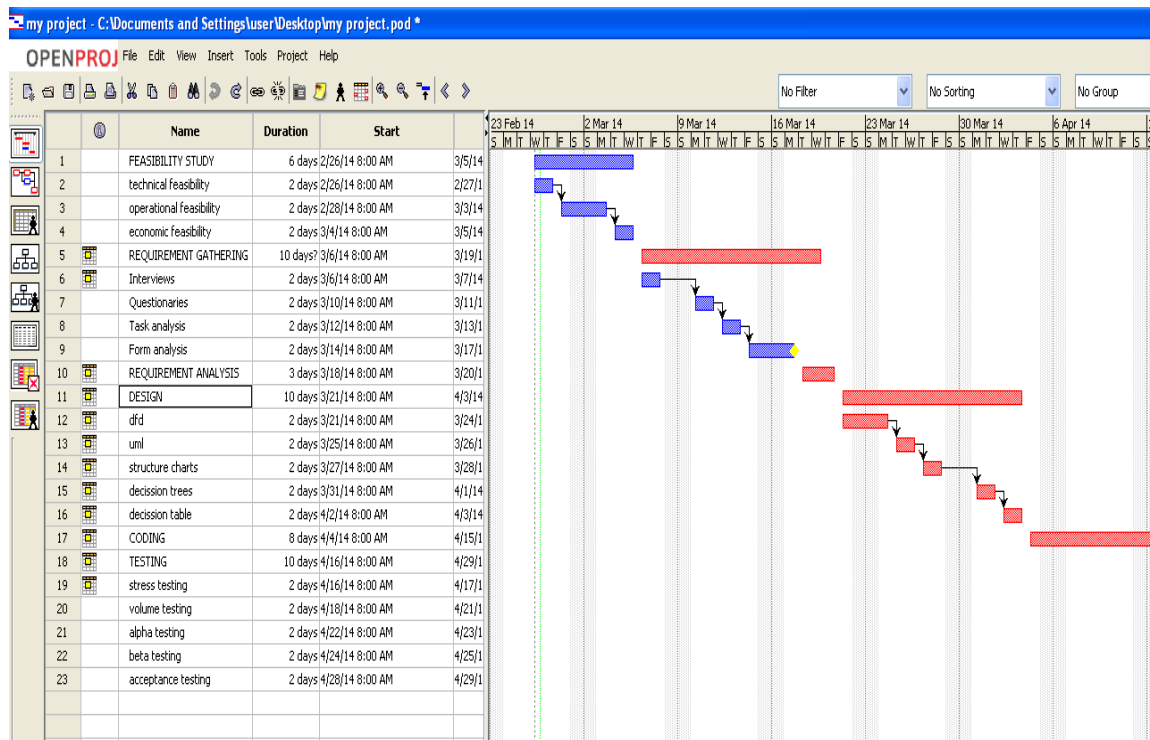
Dependencies allow you to show the relationships between tasks and set rules for when tasks can be started or finished. OpenProj uses four types of dependencies:

- **FS (Finish-to-Start):-** This is the default relationship in OpenProj. It defines that one task has to finish before the next one can start. For Example, ‘printing a document’ cannot start until ‘editing the document’ is finished.
- **SS (Start-to-Start):-** The start of a task is dependent on the start of its predecessor. In other words, the task can only start after the predecessor task has started (or at a later date)
- **FF (Finish-to-Finish):-** The completion of a task is dependent on the completion of its predecessor. In other words, the task can only finish at the same time (or after) the previous task has finished
- **SF (Start-to-Finish):-** The finish of the next task depends on the start of the previous task. In other words, the first task begins after the second task ends.

When dependencies are created, the start and finish dates of tasks are usually affected. OpenProj automatically edits start and finish dates so that they adhere to these new constraints. Any change to dependencies will update task dates and Gantt bars automatically. You can link tasks listed below. If you make a mistake at any time you can undo with the menu: Edit-Undo or press Ctrl-Z. The following steps are taken to set dependencies.

1. Select the task rows in the table that are to be linked with a dependency. In the example above we know all the tasks are dependent on each other, so we will select rows 2,3,4,5 & 6 with the mouse and click (menu: Edit / Link) or simply click the task linking icon .
2. You will see that all the tasks have now become dependent on each other with one click, rather than typing each individual task dependency in manually. You can select multiple or separated rows by using the CTRL-click action (for multiple selections).

You can delete dependencies by clicking on the number in the “predecessors” box and deleting it with the delete key. Alternatively you can select the linked tasks in the table and use the menu item Edit – Unlink or click the icon or click the link line between the two tasks in the Gantt chart and select the “Remove” option in the pop-up menu.



All task-related information is managed centrally via the Task Information window. For every task you have the ability to modify information and task settings within six tabs. It's good practice to edit information in this menu rather than in the columns in the table view of the spreadsheet. The editing window consists of following tabs as follows.

Task Information - 10

General | Predecessors | Successors | Resources | Advanced | Notes

Name: REQUIREMENT ANALYSIS

Duration: 3 days ☐ Estimated

Percent Complete: 0% Priority: 520

Cost: \$0.00 Work: 24 hours

Dates

Start: 3/18/14 8:00 AM Finish: 3/20/14 5:00 PM

Baseline Start: Baseline Finish:

Close Help

General: - General task date information and percentage complete information.

Predecessors: - List of all predecessor tasks and dependency type settings.

Successors: - Information about successor tasks and dependency type settings.

Resources: - Shows the assigned resources and allocations for this task.

Advanced: - Task type and date constraint information.

Notes Allows entry of notes about the task.

Deadlines are used to monitor the progress of individual tasks. A deadline set for a particular date will trigger a notification as soon as that date slips and an icon in the indicator column is displayed.

Deadlines also appear in the Gantt chart as yellow diamonds. To set a deadline:

1. Select the task in Name column for which you'd like to apply a deadline.
2. Open the task information Dialog Box by double-clicking the task name, or click the icon.
3. Select the Advanced tab and enter the desired date into the Deadline: field.
4. Confirm your entry and click Close.

Task Information - 10

General Predecessors Successors Resources **Advanced** Notes

Name: REQUIREMENT ANALYSIS

WBS: ☐ Display task as milestone

Constraints

Constraint Type: Start No Earlier Than Constraint Date: 3/18/14 8:00 AM

Deadline:

Type:

Task Calendar:

Earned Value Method:

Close Help

Sun	Mon	Tue	Wed	Thu	Fri	Sat
23	24	25	26	27	28	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Practical 3.: Preparation of Software Requirement Specification Document, Design Documents and Testing Phase related documents for some problems

a) Software Requirement Specification Document

An SRS is basically an organization's understanding (in writing) of a customer or potential client's system requirements and dependencies at a particular point in time (usually) prior to any actual design or development work. It's a two-way insurance policy that assures that both the client and the organization understand the other's requirements from that perspective at a given point in time.

The SRS document itself states in precise and explicit language those functions and capabilities a software system (i.e., a software application, an eCommerce Web site, and so on) must provide, as well as states any required constraints by which the system must abide. The SRS also functions as a blueprint for completing a project with as little cost growth as possible. The SRS is often referred to as the "parent" document because all subsequent project management documents, such as design specifications, statements of work, software architecture specifications, testing and validation plans, and documentation plans, are related to it.

It's important to note that an SRS contains functional and nonfunctional requirements only; it doesn't offer design suggestions, possible solutions to technology or business issues, or any other information other than what the development team understands the customer's system requirements to be.

A well-designed, well-written SRS accomplishes four major goals:

- It provides feedback to the customer. An SRS is the customer's assurance that the development organization understands the issues or problems to be solved and the software behavior necessary to address those problems. Therefore, the SRS should be written in natural language (versus a formal language, explained later in this article), in an unambiguous manner that may also include charts, tables, data flow diagrams, decision tables, and so on.
- It decomposes the problem into component parts. The simple act of writing down software requirements in a well-designed format organizes information, places borders around the problem, solidifies ideas, and helps break down the problem into its component parts in an orderly fashion.
- It serves as an input to the design specification. As mentioned previously, the SRS serves as the parent document to subsequent documents, such as the software design specification and statement of work. Therefore, the SRS must contain sufficient detail in the functional system requirements so that a design solution can be devised.
- It serves as a product validation check. The SRS also serves as the parent document for testing and validation strategies that will be applied to the requirements for verification.

SRSs are typically developed during the first stages of "Requirements Development," which is the initial product development phase in which information is gathered about what requirements are needed--and not. This information-gathering stage can include onsite visits, questionnaires, surveys, interviews, and perhaps a return-on-investment (ROI) analysis or needs analysis of the customer or client's current business environment. The actual specification, then, is written after the requirements have been gathered and analyzed.

Why Should Technical Writers be Involved with Software Requirements Specifications?

Unfortunately, much of the time, systems architects and programmers write SRSs with little (if any) help from the technical communications organization. And when that assistance is provided, it's often limited to an edit of the final draft just prior to going out the door. Having technical writers involved throughout the entire SRS development process can offer several benefits:

- Technical writers are skilled information gatherers, ideal for eliciting and articulating customer requirements. The presence of a technical writer on the requirements-gathering team helps balance the type and amount of information extracted from customers, which can help improve the SRS.
- Technical writers can better assess and plan documentation projects and better meet customer document needs. Working on SRSs provides technical writers with an opportunity for learning about customer needs firsthand--early in the product development process.
- Technical writers know how to determine the questions that are of concern to the user or customer regarding ease of use and usability. Technical writers can then take that knowledge and apply it not only to the specification and documentation development, but also to user interface development, to help ensure the UI (User Interface) models the customer requirements.
- Technical writers, involved early and often in the process, can become an information resource throughout the process, rather than an information gatherer at the end of the process.

In short, a requirements-gathering team consisting solely of programmers, product marketers, systems analysts/architects, and a project manager runs the risk of creating a specification that may be too heavily loaded with technology-focused or marketing-focused issues. The presence of a technical writer on the team helps place at the core of the project those user or customer requirements that provide more of an overall balance to the design of the SRS, product, and documentation.

What Kind of Information Should an SRS Include?

You probably will be a member of the SRS team (if not, ask to be), which means SRS development will be a collaborative effort for a particular project. In these cases, your company will have developed SRSs before, so you should have examples (and, likely, the company's SRS template) to

use. But, let's assume you'll be starting from scratch. Several standards organizations (including the IEEE) have identified nine topics that must be addressed when designing and writing an SRS:

1. Interfaces
2. Functional Capabilities
3. Performance Levels
4. Data Structures/Elements
5. Safety
6. Reliability
7. Security/Privacy
8. Quality
9. Constraints and Limitations

Begin with an SRS Template

The first and biggest step to writing an SRS is to select an existing template that you can fine tune for your organizational needs (if you don't have one already). There's not a "standard specification template" for all projects in all industries because the individual requirements that populate an SRS are unique not only from company to company, but also from project to project within any one company. The key is to select an existing template or specification to begin with, and then adapt it to meet your needs.

In recommending using existing templates, I'm not advocating simply copying a template from available resources and using them as your own; instead, I'm suggesting that you use available templates as guides for developing your own. It would be almost impossible to find a specification or specification template that meets your particular project requirements exactly. But using other templates as guides is how it's recommended in the literature on specification development. Look at what someone else has done, and modify it to fit your project requirements. (See the sidebar called "Resources for Model Templates" at the end of this article for resources that provide sample templates and related information.)

Table 1 shows what a basic SRS outline might look like. This example is an adaptation and extension of the IEEE Standard 830-1998:

Table 1 A sample of a basic SRS outline

1.	Introduction
1.1	Purpose

1.2	Document	conventions
1.3	Intended	audience
1.4	Additional	information
1.5	Contact information/SRS	team members
1.6	References	
2.	Overall	Description
2.1	Product	perspective
2.2	Product	functions
2.3	User classes and	characteristics
2.4	Operating	environment
2.5	User	environment
2.6	Design/implementation	constraints
2.7	Assumptions and dependencies	
3.	External Interface	Requirements
3.1	User	interfaces
3.2	Hardware	interfaces
3.3	Software	interfaces
3.4	Communication protocols and interfaces	
4.	System	Features
4.1	System	feature A
4.1.1	Description and	priority
4.1.2		Action/result
4.1.3	Functional	requirements
4.2	System feature B	
5.	Other Nonfunctional	Requirements
5.1	Performance	requirements
5.2	Safety	requirements
5.3	Security	requirements
5.4	Software quality	attributes
5.5	Project	documentation
5.6	User documentation	
6.	Other	Requirements
Appendix A: Terminology/Glossary/Definitions list		
Appendix B: To be determined		

Table 2 shows a more detailed SRS outline, showing the structure of an SRS template as found on Ken Rigby's informative Web site at http://neon.airtime.co.uk/users/wysywig/srs_mt.htm. Reprinted with permission.

Table 2 A sample of a more detailed SRS outline

1. Scope	<p>1.1 Identification.</p> <p><i>Identify the system and the software to which this document applies, including, as applicable, identification number(s), title(s), abbreviation(s), version number(s), and release number(s).</i></p> <p>1.2 System overview.</p> <p><i>State the purpose of the system or subsystem to which this document applies.</i></p> <p>1.3 Document overview.</p> <p><i>Summarize the purpose and contents of this document.</i></p> <p>This document comprises six sections:</p> <ul style="list-style-type: none"> • Scope • Referenced documents • Requirements • Qualification provisions • Requirements traceability • Notes <p>Describe any security or privacy considerations associated with its use.</p>
2. Referenced Documents	<p>2.1 Project documents.</p> <p><i>Identify the project management system documents here.</i></p> <p>2.2 Other documents.</p> <p>2.3 Precedence.</p> <p>2.4 Source of documents.</p>
3. Requirements	<p>This section shall be divided into paragraphs to specify</p>

	<p>the Computer Software Configuration Item (CSCI) requirements, that is, those characteristics of the CSCI that are conditions for its acceptance. CSCI requirements are software requirements generated to satisfy the system requirements allocated to this CSCI. Each requirement shall be assigned a project-unique identifier to support testing and traceability and shall be stated in such a way that an objective test can be defined for it.</p> <p>3.1 Required states and modes.</p> <p>3.2 CSCI capability requirements.</p> <p>3.3 CSCI external interface requirements.</p> <p>3.4 CSCI internal interface requirements.</p> <p>3.5 CSCI internal data requirements.</p> <p>3.6 Adaptation requirements.</p> <p>3.7 Safety requirements.</p> <p>3.8 Security and privacy requirements.</p> <p>3.9 CSCI environment requirements.</p> <p>3.10 Computer resource requirements.</p> <p>3.11 Software quality factors.</p> <p>3.12 Design and implementation constraints.</p> <p>3.13 Personnel requirements.</p> <p>3.14 Training-related requirements.</p> <p>3.15 Logistics-related requirements.</p> <p>3.16 Other requirements.</p> <p>3.17 Packaging requirements.</p> <p>3.18 Precedence and criticality requirements.</p>
4. Qualification Provisions	To be determined.
5. Requirements Traceability	To be determined.
6. Notes	This section contains information of a general or explanatory nature that may be helpful, but is not mandatory.

	<p>6.1 Intended use.</p> <p>This Software Requirements specification shall ...</p> <p>6.2 Definitions used in this document.</p> <p><i>Insert here an alphabetic list of definitions and their source if different from the declared sources specified in the "Documentation standard."</i></p> <p>6.3 Abbreviations used in this document.</p> <p><i>Insert here an alphabetic list of the abbreviations and acronyms if not identified in the declared sources specified in the "Documentation Standard."</i></p> <p>6.4 Changes from previous issue.</p> <p><i>Will be "not applicable" for the initial issue.</i></p> <p>Revisions shall identify the method used to identify changes from the previous issue.</p>
--	--

Identify and Link Requirements with Sources

As noted earlier, the SRS serves to define the functional and nonfunctional requirements of the product. Functional requirements each have an origin from which they came, be it a *use case* (which is used in system analysis to identify, clarify, and organize system requirements, and consists of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal), government regulation, industry standard, or a business requirement. In developing an SRS, you need to identify these origins and link them to their corresponding requirements. Such a practice not only justifies the requirement, but it also helps assure project stakeholders that frivolous or spurious requirements are kept out of the specification.

To link requirements with their sources, each requirement included in the SRS should be labeled with a unique identifier that can remain valid over time as requirements are added, deleted, or changed. Such a labeling system helps maintain change-record integrity while also serving as an identification system for gathering metrics. You can begin a separate requirements identification list that ties a requirement identification (ID) number with a description of the requirement. Eventually, that requirement ID and description become part of the SRS itself and then part of the Requirements Traceability Matrix, discussed in subsequent paragraphs. Table 3 illustrates how these SRS ingredients work together.

Table 3 This sample table identifies requirements and links them to their sources

ID No.	Paragraph No.	Requirement	Business Rule Source

17	5.1.4.1	Understand/communicate using SMTP protocol	IEEE STD xx-xxxx
18	5.1.4.1	Understand/communicate using POP protocol	IEEE STD xx-xxxx
19	5.1.4.1	Understand/communicate using IMAP protocol	IEEE STD xx-xxxx
20	5.1.4.2	Open at same rate as OE	Use Case Doc 4.5.4

What Should I Know about Writing an SRS?

Unlike formal language that allows developers and designers some latitude, the natural language of SRSs must be exact, without ambiguity, and precise because the design specification, statement of work, and other project documents are what drive the development of the final product. That final product must be tested and validated against the design and original requirements. Specification language that allows for interpretation of key requirements will not yield a satisfactory final product and will likely lead to cost overruns, extended schedules, and missed deliverable deadlines.

Table 4 shows the fundamental characteristics of a quality SRS, which were originally presented at the April 1998 Software Technology Conference presentation "Doing Requirements Right the First Time." Reprinted with permission from the Software Assurance Technology Center at NASA These quality characteristics are closely tied to what are referred to as "indicators of strength and weakness," which will be defined next.

Table 4 The 10 language quality characteristics of an SRS

SRS Quality Characteristic	What It Means
<i>Complete</i>	SRS defines precisely all the go-live situations that will be encountered and the system's capability to successfully address them.
<i>Consistent</i>	SRS capability functions and performance levels are compatible, and the required quality features (security, reliability, etc.) do not negate those capability functions. For example, the only electric hedge trimmer that is safe is one that is stored in a box and not connected to any

	electrical cords or outlets.
<i>Accurate</i>	SRS precisely defines the system's capability in a real-world environment, as well as how it interfaces and interacts with it. This aspect of requirements is a significant problem area for many SRSs.
<i>Modifiable</i>	The logical, hierarchical structure of the SRS should facilitate any necessary modifications (grouping related issues together and separating them from unrelated issues makes the SRS easier to modify).
<i>Ranked</i>	Individual requirements of an SRS are hierarchically arranged according to stability, security, perceived ease/difficulty of implementation, or other parameter that helps in the design of that and subsequent documents.
<i>Testable</i>	An SRS must be stated in such a manner that unambiguous assessment criteria (pass/fail or some quantitative measure) can be derived from the SRS itself.
<i>Traceable</i>	Each requirement in an SRS must be uniquely identified to a source (use case, government requirement, industry standard, etc.)
<i>Unambiguous</i>	SRS must contain requirements statements that can be interpreted in one way only. This is another area that creates significant problems for SRS development because of the use of natural language.
<i>Valid</i>	A valid SRS is one in which all parties and project participants can understand, analyze, accept, or approve it. This is one of the main reasons SRSs are written using natural language.
<i>Verifiable</i>	A verifiable SRS is consistent from one level of abstraction to another. Most attributes of a specification are subjective and a conclusive assessment of quality requires a technical review by domain experts. Using

	indicators of strength and weakness provide some evidence that preferred attributes are or are not present.
--	---

What makes an SRS "good?" How do we know when we've written a "quality" specification? The most obvious answer is that a quality specification is one that fully addresses all the customer requirements for a particular product or system. That's part of the answer. While many quality attributes of an SRS are subjective, we do need indicators or measures that provide a sense of how strong or weak the language is in an SRS. A "strong" SRS is one in which the requirements are tightly, unambiguously, and precisely defined in such a way that leaves no other interpretation or meaning to any individual requirement.

The Goddard Space Flight Center (GSFC) studied dozens of NASA requirements specifications that revealed nine categories of SRS quality indicators. The individual components in each category are words, phrases, and sentence structures that are related to quality attributes. The nine categories fall into two classes: those related to individual specification statements, and those related to the total SRS document. Table 5 summarizes the classes, categories, and components of these quality indicators. This table was also originally presented at the April 1998 Software Technology Conference presentation "Doing Requirements Right the First Time." Reprinted with permission from the Software Assurance Technology Center at NASA (<http://www.gsfc.nasa.gov/>).

Table 5 Quality measures related to individual SRS statements

<i>Imperatives:</i> Words and phrases that command the presence of some feature, function, or deliverable. They are listed below in decreasing order of strength.	
Shall	Used to dictate the provision of a functional capability.
Must or must not	Most often used to establish performance requirement or constraints.
Is required to	Used as an imperative in SRS statements when written in passive voice.
Are applicable	Used to include, by reference, standards, or other documentation as an addition to the requirement being specified.
Responsible for	Used as an imperative in SRSs that are written for systems with pre-defined architectures.

Will	Used to cite things that the operational or development environment is to provide to the capability being specified. For example, The vehicle's exhaust system will power the ABC widget.
Should	Not used often as an imperative in SRS statements; however, when used, the SRS statement always reads weak. Avoid using Should in your SRSs.

Continuances: Phrases that follow an imperative and introduce the specification of requirements at a lower level. There is a correlation with the frequency of use of *continuances* and SRS organization and structure, up to a point. Excessive use of *continuances* often indicates a very complex, detailed SRS. The *continuances* below are listed in decreasing order of use within NASA SRSs. Use *continuances* in your SRSs, but balance the frequency with the appropriate level of detail called for in the SRS.

1. **Below:**
2. **As follows:**
3. **Following:**
4. **Listed:**
5. **In particular:**
6. **Support:**

Directives: Categories of words and phrases that indicate illustrative information within the SRS. A high ratio of total number of *directives* to total text line count appears to correlate with how precisely requirements are specified within the SRS. The *directives* below are listed in decreasing order of occurrence within NASA SRSs. Incorporate the use of *directives* in your SRSs.

1. **Figure**
2. **Table**
3. **For example**
4. **Note**

Options: A category of words that provide latitude in satisfying the SRS statements that contain them. This category of words loosens the SRS, reduces the client's control over the final product, and allows for possible cost

and schedule risks. You should avoid using them in your SRS. The *options* below are listed in the order they are found most often in NASA SRSs.

1. Can

2. May

3. Optionally

Weak phrases: A category of clauses that can create uncertainty and multiple/subjective interpretation. The total number of *weak phrases* found in an SRS indicates the relative ambiguity and incompleteness of an SRS. The *weak phrases* below are listed alphabetically.

adequate	be able to	easy	provide for
as a minimum	be capable of	effective	timely
as applicable	but not limited to	if possible	tbd
as appropriate	capability of	if practical	
at a minimum	capability to	normal	

Size: Used to indicate the *size* of the SRS document, and is the total number of the following:

1. Lines of text

2. Number of imperatives

3. Subjects of SRS statements

4. Paragraphs

Text Structure: Related to the number of statement identifiers found at each hierarchical level of the SRS and indicate the document's organization, consistency, and level of detail. The most detailed NASA SRSs were nine levels deep. High-level SRSs were rarely more than four levels deep. SRSs deemed well organized and a consistent level of detail had *text structures* resembling pyramids (few level 1 headings but each lower level having more numbered statements than the level above it). Hour-glass-shaped *text structures* (many level 1 headings, few a mid-levels, and many at lower levels) usually contain a greater amount of introductory and administrative information. Diamond-shaped *text structures* (pyramid shape followed by decreasing statement counts at levels below the pyramid) indicated that

subjects introduced at higher levels were addressed at various levels of detail.

Specification Depth: The number of imperatives found at each of the SRS levels of text structure. These numbers include the count of lower level list items that are introduced at a higher level by an imperative and followed by a continuance. The numbers provide some insight into how much of the Requirements document was included in the SRS, and can indicate how concise the SRS is in specifying the requirements.

Readability Statistics: Measurements of how easily an adult can read and understand the requirements document. Four readability statistics are used (calculated by Microsoft Word). While readability statistics provide a relative quantitative measure, don't sacrifice sufficient technical depth in your SRS for a number.

1. Flesch Reading Ease index

2. Flesch-Kincaid Grade Level index

3. Coleman-Liau Grade Level index

4. Bormuth Grade Level index

Conclusion

There's so much more we could say about requirements and specifications. Hopefully, this information will help you get started when you are called upon-or step up-to help the development team. Writing top-quality requirements specifications begins with a complete definition of customer requirements. Coupled with a natural language that incorporates strength and weakness quality indicators-not to mention the adoption of a good SRS template--technical communications professionals well-trained in requirements gathering, template design, and natural language use are in the best position to create and add value to such critical project documentation.

Design Documents

A software design document (SDD) is a written description of a software product, that a software designer writes in order to give a software development team overall guidance to the architecture of the software project. An SDD usually accompanies an architecture diagram with pointers to detailed feature specifications of smaller pieces of the design. Practically, a design document is required to coordinate a large team under a single vision. A design document needs to be a stable reference, outlining all parts of the software and how they will work. The document is commanded to give a fairly complete description, while maintaining a high-level view of the software.

There are two kinds of design documents called HLDD (high-level design document) and LLDD (low-level design document).

HLD - High Level Design (HLD) is the overall system design - covering the system architecture and database design. It describes the relation between various modules and functions of the system. data flow, flow charts and data structures are covered under HLD. High Level Design gives the overall System Design in terms of Functional Architecture details and Database design. This is very important for the ETL developers to understand the flow of the system with function and database design wise. In this phase the design team, testers and customers are plays a major role. Also it should have projects standards, the functional design documents and the database design document also.

LLD - Low Level Design (LLD) is like detailing the HLD. It defines the actual logic for each and every component of the system. Class diagrams with all the methods and relation between classes comes under LLD. Programs specs are covered under LLD. This document is need to do during the detailed phase, the view of the application developed during the high level design is broken down into separate modules and programs for every program and then documented by program specifications.

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs (errors or other defects).

It involves the execution of a software component or system to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

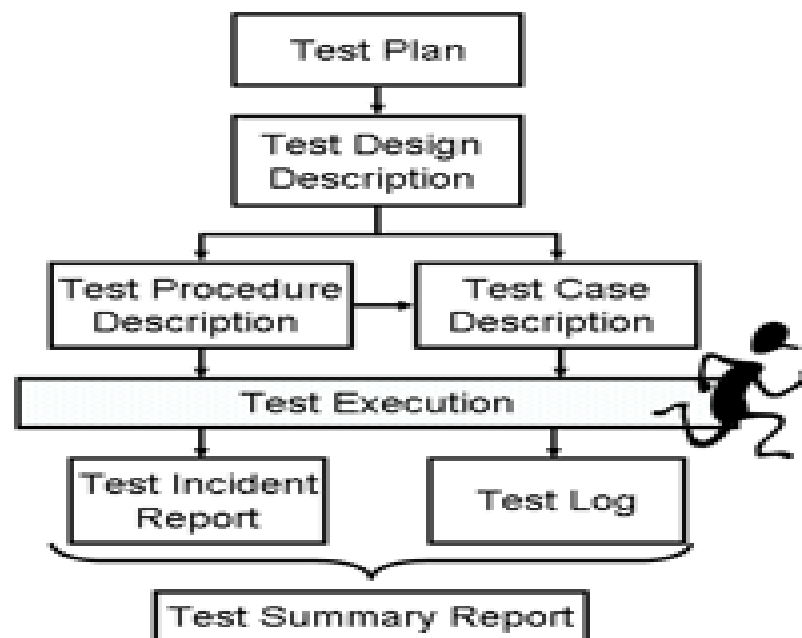
- meets the requirements that guided its design and development,
- responds correctly to all kinds of inputs,
- performs its functions within an acceptable time,
- is sufficiently usable,
- can be installed and run in its intended environments, and
- achieves the general result its stakeholders desire.

As the number of possible tests for even simple software components is practically infinite, all software testing uses some strategy to select tests that are feasible for the available time and resources. As a result, software testing typically (but not exclusively) attempts to execute a program or application with the intent of finding software bugs (errors or other defects).

Software testing can provide objective, independent information about the quality of software and risk of its failure to users and/or sponsors.¹

Software testing can be conducted as soon as executable software (even if partially complete) exists. The overall approach to software development often determines when and how testing is conducted. For example, in a phased process, most testing occurs after system requirements have been defined and then implemented in testable programs. In contrast, under an Agile approach, requirements, programming, and testing are often done concurrently

Test documentation is the complete suite of artefacts that describe test planning, test design, test execution, test results and conclusions drawn from the testing activity. As testing activities typically consume 30% to 50% of project effort, testing represents a project within a project. Testing activities must therefore be fully documented to support resource allocation, monitoring and control. This page identifies the types of documents you need to set up and run your test program and summarises their content.



Source: [IEEE 829]

Test Plan

The test plan describes the testing process in terms of the features to be tested, pass/fail criteria and testing approach, resource requirements and schedules.

1. Introduction
2. Test items
3. Features to be tested
4. Testing approach
5. Item pass/fail criteria
6. Suspension and resumption
7. Deliverables

8. Tasks
9. Environmental needs
10. Responsibilities
11. Staffing and training needs
12. Costs and schedule
13. Risks and contingencies

Test Design Description

The Test Design Description refines the Test Plan's approach, identifying specific features to be tested and defining the test cases and test procedures that will be used.

1. Features to be tested
 - Test items covered
 - Feature or feature combinations to be tested
 - References to software requirements specifications and software design descriptions
2. Testing approach
 - Testing techniques to be used
 - Method of analysing test results (e.g. automated or manual)
 - Reasons for selection of various test cases
 - Environmental needs of test cases
3. Test identification

For each feature to be tested provide:

- Test procedure identifiers and descriptions
- Test case identifiers and descriptions
- The pass/fail criteria.

Practical 4. Preparation of Software Configuration Management and Risk Management related documents

The purpose of Software Configuration Management is to establish and maintain the integrity of the products of the software project throughout the project's software life cycle. Software Configuration Management involves identifying configuration items for the software project, controlling these configuration items and changes to them, and recording and reporting status and change activity for these configuration items .

Configuration management (CM) refers to a discipline for evaluating, coordinating, approving or disapproving, and implementing changes in artefacts that are used to construct and maintain software systems. An artifact may be a piece of hardware or software or documentation. CM enables the management of artifact from the initial concept through design, implementation, testing, baselining, building, release, and maintenance.

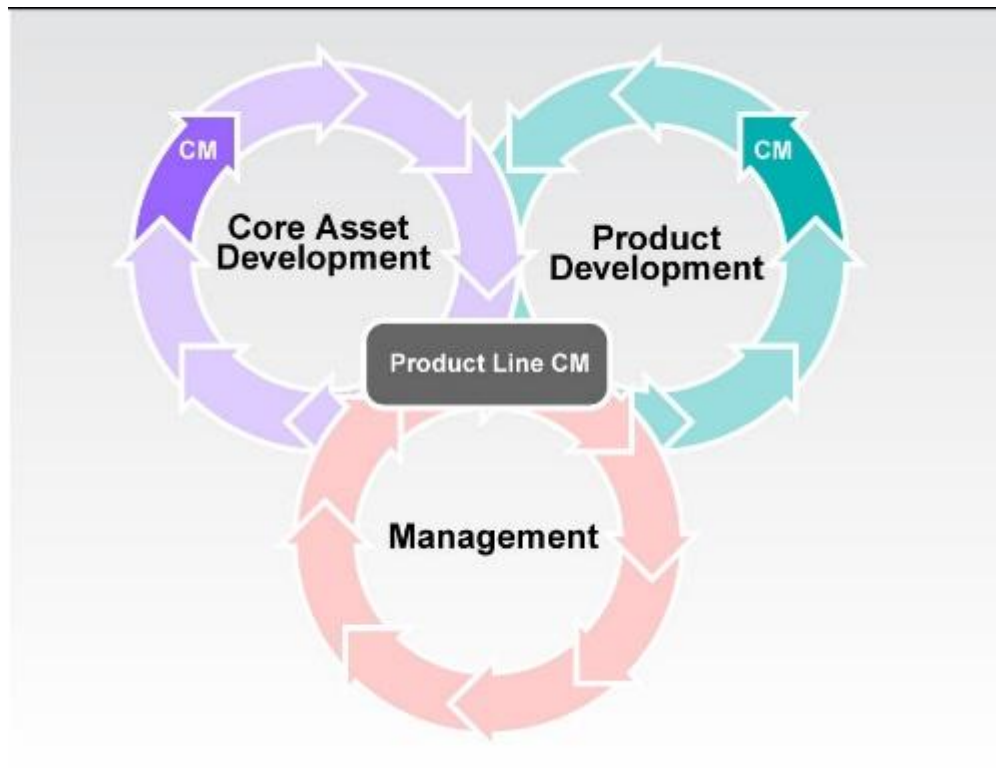
CM is intended to eliminate the confusion and error brought about by the existence of different versions of artifacts. Artifact change is a fact of life: plan for it or plan to be overwhelmed by it. Changes are made to correct errors, provide enhancements, or simply reflect the evolutionary refinement of product definition. CM is about keeping the inevitable change under control. Without a well-enforced CM process, different team members (possibly at different sites) can use different versions of artifacts unintentionally; individuals can create versions without the proper authority; and the wrong version of an artifact can be used inadvertently. Successful CM requires a well-defined and institutionalized set of policies and standards that clearly define

- the set of artifacts (configuration items) under the jurisdiction of CM
- how artifacts are named
- how artifacts enter and leave the controlled set
- how an artifact under CM is allowed to change
- how different versions of an artifact under CM are made available and under what conditions each one can be used
- how CM tools are used to enable and enforce CM

These policies and standards are documented in a CM plan that informs everyone in the organization just how CM is carried out.

Aspects Peculiar to Product Lines

CM is, of course, an integral part of any software development activity, but it takes on a special significance in the product line context. As illustrated in the following figure, CM for product lines is generally viewed as a multidimensional version of the CM problem for one-of-a-kind systems.



Configuration Management and Software Product Lines

The core assets constitute a configuration that needs to be managed, each product in the product line constitutes a configuration that must be managed, and the management of all these configurations must be coordinated under a single process.

CM for product lines is therefore more complex than it is for single systems. CM capabilities such as parallel development, distributed engineering, build and release management, change management, configuration and workspace management, and process management must be supported by the tools, processes, and environments put in place for CM in a product line context.

- In single-system CM, each version of the system has a configuration associated with it that defines the versions of the configuration items that went into that system's production. In product line CM, a configuration must be maintained for each version of *each product*.
- In single-system CM, each product with all of its versions may be managed separately. In product line CM, such management is untenable, because the core assets are used across all products. Hence, the entire product line is usually managed with a single, unified CM process.
- Product line CM must control the configuration of the core asset base and its use by all product developers. It must account for the fact that core assets are usually produced by one team and used in parallel by several others. Single-system CM has no such burden: the component developers and product developers are the same people.

- Only the most capable CM tools can be used in a product line effort. Many tools that are adequate for single-system CM are simply not sufficiently robust to handle the demands of product line CM. (See the "Tool Support" practice area for a more complete discussion of tools.)

The mission of product line CM is allowing the rapid reconstruction of any product version that may have been built using various versions of the core assets and development/operating environment plus various versions of product-specific artifacts. One product line manager summed up the problem this way: "Sometime, in the middle of the night, one of your customers is going to call you and tell you that his version of one of your products doesn't work. You are going to have to duplicate that product in your test lab before you can begin to troubleshoot.

Risk Management Plan :

Purpose of the Risk management plan:

A risk is an event or condition that, if it occurs, could have a positive or negative effect on a project's objectives. Risk Management is the process of identifying, assessing, responding to, monitoring, and reporting risks. This Risk Management Plan defines how risks associated with the <Project Name> project will be identified, analyzed, and managed. It outlines how risk management activities will be performed, recorded, and monitored throughout the lifecycle of the project and provides templates and practices for recording and prioritizing risks.

The Risk Management Plan is created by the project manager in the Planning Phase of the CDC Unified Process and is monitored and updated throughout the project. The intended audience of this document is the project team, project sponsor and management.

Risk management Procedure:

Process

The project manager working with the project team and project sponsors will ensure that risks are actively identified, analyzed, and managed throughout the life of the project. Risks will be identified as early as possible in the project so as to minimize their impact. The steps for accomplishing this are outlined in the following sections. The <project manager or other designee> will serve as the Risk Manager for this project.

Risk Identification

Risk identification will involve the project team, appropriate stakeholders, and will include an evaluation of environmental factors, organizational culture and the project management plan

including the project scope. Careful attention will be given to the project deliverables, assumptions, constraints, WBS, cost/effort estimates, resource plan, and other key project documents.

A Risk Management Log will be generated and updated as needed and will be stored electronically in the project library located at <file location>.

Risk Analysis

All risks identified will be assessed to identify the range of possible project outcomes. Qualification will be used to determine which risks are the top risks to pursue and respond to and which risks can be ignored Qualitative Risk Analysis

The probability and impact of occurrence for each identified risk will be assessed by the project manager, with input from the project team using the following approach:

Probability

- High – Greater than <70%> probability of occurrence
- Medium – Between <30%> and <70%> probability of occurrence
- Low – Below <30%> probability of occurrence

Impact

- High – Risk that has the potential to greatly impact project cost, project schedule or performance
- Medium – Risk that has the potential to slightly impact project cost, project schedule or performance
- Low – Risk that has relatively little impact on cost, schedule or performance

Impact	H			
	M			
	L			
		L	M	H
Probability				

Risks that fall within the RED and YELLOW zones will have risk response planning which may include both a risk mitigation and a risk contingency plan.

Quantitative Risk Analysis

Analysis of risk events that have been prioritized using the qualitative risk analysis process and their affect on project activities will be estimated, a numerical rating applied to each risk based on this analysis, and then documented in this section of the risk management plan.

Risk Response Planning

Each major risk (those falling in the Red & Yellow zones) will be assigned to a project team member for monitoring purposes to ensure that the risk will not “fall through the cracks”.

For each major risk, one of the following approaches will be selected to address it:

- Avoid – eliminate the threat by eliminating the cause

- Mitigate – Identify ways to reduce the probability or the impact of the risk
- Accept – Nothing will be done
- Transfer – Make another party responsible for the risk (buy insurance, outsourcing, etc.)

For each risk that will be mitigated, the project team will identify ways to prevent the risk from occurring or reduce its impact or probability of occurring. This may include prototyping, adding tasks to the project schedule, adding resources, etc.

For each major risk that is to be mitigated or that is accepted, a course of action will be outlined for the event that the risk does materialize in order to minimize its impact.

Risk management plan approval

The undersigned acknowledge they have reviewed the Risk Management Plan for the <Project Name> project. Changes to this Risk Management Plan will be coordinated with and approved by the undersigned or their designated representatives.

[List the individuals whose signatures are desired. Examples of such individuals are Business Steward, Project Manager or Project Sponsor. Add additional lines for signature as necessary. Although signatures are desired, they are not always required to move forward with the practices outlined within this document.]

Signature:	_____	Date:	_____
Print Name:	_____		
Title:	_____		
Role:	_____		

Practical 5: Study and usage of any Design phase CASE tool.

Introduction: CASE (Computer Aided Software Engineering) technologies are tools that provide the assistance to the developer in the development of software. Main purpose of the CASE tools is to decrease the development time and increase the quality of software. Even the presence of these qualities CASE tools are not being used most often or freely. These tools are not used freely as they should be; there are some points that need improvement, so that the use of CASE tools can be increased. No a days most of the software houses don't bother to use the CASE tools in their development process. And finally the hurdles in the promotion of the CASE tools as a standard.CASE tools reduce the time and cast of software development and ensure the quality of software. The objective of introducing Computer Aided Software Engineering (CASE) tools was the reduction of the time, cost of software development and for the enhancement of the quality of the systems developed (Diane Lending et al. 1998).

Practical 5: Study and usage of any Design phase CASE tool.

Introduction: CASE (Computer Aided Software Engineering) technologies are tools that provide the assistance to the developer in the development of software. Main purpose of the CASE tools is to decrease the development time and increase the quality of software. Even the presence of these qualities CASE tools are not being used most often or freely. These tools are not used freely as they should be; there are some points that need improvement, so that the use of CASE tools can be increased. Now a days most of the software houses don't bother to use the CASE tools in their development process. And finally the hurdles in the promotion of the CASE tools as a standard. CASE tools reduce the time and cost of software development and ensure the quality of software. The objective of introducing Computer Aided Software Engineering (CASE) tools was the reduction of the time, cost of software development and for the enhancement of the quality of the systems developed (Diane Lending et al. 1998).

CASE technologies are tools that provide automated assistance for software development to the developers. The goal of introducing CASE tools is to reduce the time of development, reduce the cost of software, and the enhancement of the quality of the software. Upper CASE tools are used to capture, analyze and organize the models of system. These models help designers to focus on the systems linear behaviour. Lower CASE tools are used for development and software maintenance phases. Using these tools developers classify the scope and boundaries, describe current system, model requirements, prototyping, prepare design, etc. Integrated CASE tools provide support for Upper CASE and Lower CASE tool activities. Integrated CASE tools helps specifically in Analysis & Design, Maintenance and system planning. Integrated CASE tools provide the support throughout whole development lifecycle.

History of CASE Tools

Right from the beginning of software development, software engineer have been in search of such tools that will prove helpful for them in the modeling of system, understanding of system in easy way and in many other ways to provide ease to the people related to computer science. From 1968 to 1971 they are a lot of improvement done in this regard like the top-down structure design, benefit of modularity and step by step code refinement were admitted, up to 1975 quality, and reliability of software were the big issues and testing procedures were adopt for software reliance. The main start of CASE tools start in 1980's, when documentation, diagrams and design tools were introduce in this field. In same decade automated analysis design check, automated code generation and linking design utilities were introduce. In same decade importance of CASE tools was considered for the large scale system and CASE establish as an industry. Now a dyas there are lot of CASE tools even more than one CASE toils for same problem. Now tools are available that help developers to analyze, design and for the documentation of the software. Even CASE tools are

available for deployment and maintenance of the software. Tools like Erwin, Rational Rose has create there importance in the market due to their qualities and reliability. In last ten years CASE tools has emerged as an interactive technology and has gained good position in the software development fields. They are still in the process of improvement like the other fields of the world, because every human created thing in this world has always room for improvement.

Need of CASE

Software developers always looking for such CASE tools that help them in many different ways during the different development stages of software, so that they can understand the software and prepare a good end product that efficiently fulfill the user requirements. CASE tools provide the ways that can fulfill this requirement of software developers. These tools provide computerized setting to software developers to analyze a problem and then design its system model.

Good Points of CASE Tools

1. They provides better perceptive of system.
2. Facilitates communication among team members.
3. Tools are more effective for large scale systems and immense projects.
4. CASE tools provide visibility of processes and logic.
5. CASE tools improve quality and productivity of software.
6. CASE tools reduce the time for error correction and maintenance.
7. CASE tools provide clear readability and maintainability of the system.

Factors influencing the success of case:

CASE tools facing many troubles in creating a suitable place in market e.g.

1. Even after the completion of the design it is not necessary that it will fulfill the requirements.
2. Though CASE tools are helpful for the developer but do not assure that the design is according to the requirements.

3. Good quality CASE tools are very expensive and prove costly for the development.
4. CASE tools also required training for the user that increase the overall cost of development.
5. Almost every tool has its limitation that decreases its use and popularity.
6. Some tools may have very limited functionality and may not address all possible domain activities.
7. Every tool has a specific methodology for designing and modeling of the system. Due to this you're bound to follow them that decrease the flexibility which decrease the use of CASE tools.
8. Frequent users get used to it and afterward developers try to use the same approach and tool for other projects whether the tool addresses the target projects problems or not.

Use of CASE Tools

The purpose of CASE is to reduce the cost and time required for the system development and the focus is on the quality of the end product. CASE is not being used as it was being expected. Most of the companies are reluctant to adopt the CASE tools. It is observed that CASE is being used but not in many companies. "The reasons for abandonment included cost, lack of measurable returns, and unrealistic expectations. Organizations that used CASE tools and found that large numbers of their systems developers were not using CASE tools. He reported that in 57% of the organizations surveyed that were using CASE tools, less than 25% of the systems developers used the tools" (Diane Lending 1998). And in those companies where CASE is adopted only few people are using CASE tools. In a survey of 67 companies it has been observed that 69% companies had never used CASE tools. And those people who are using CASE tools admitted that the use of CASE tools improved the standard of documentation and in result the system was easier to test and maintain. But people who used CASE tools also admitted that using CASE tools requires more time and effort and also adds in overall development time.

Less than 30% of the potential users and developers use the CASE and those who use it, uses the simplest and basic functionality of the CASE tools. This shows that even after getting so much popular CASE tools are not adopted and used so much in the software development industry as they expected or as they should be used. Just after one year of introduction 70% of CASE tools are never used, 20 are used by one group and 5 % of CASE tools are used in general [Juhani Iivar, 1996]. It has been observed that CASE Tools are not being used from the time CASE got recognition. CASE is being approached experimentarily and involves a very long learning curve.

Implications in CASE Tools

CASE is not being used as it was expected; there are many factors that affect the use of CASE tools.

Cost-Many tools are costly and most companies are unwilling to implement these CASE tools by the fact that it increases the overall cost of the project. And second thing if a company does not get any extra by using CASE tools then why should they use CASE tools, which are expensive for it in the end.

Time Limitation-There is always a time limit for every software project for development. Most of developers do not adopt CASE because it requires a lot of time to train developers and perform all CASE activities and there are always deadlines to complete the project. You have to meet deadlines of the company but if you use CASE tools you cannot finish your project on time, which is also a factor in less use of CASE tools.

Training-Training is a big concern while any company going to use a new tool, which is quite costly procedure. They have to train their developer for the efficient use of that particular tool. Most of the companies avoid using CASE tools because of this fact that they would have to train a large number of staff and it is expensive and as well as time consuming.

Lack of Concern-Most of the developers who use CASE tools are not fully satisfied from the CASE tools, which are why they are quite neutral about the usefulness of the CASE tools. The developer does not fully enjoy using CASE tools and if some of them do so, they use only limited functionality of the CASE tool. Many developers use CASE tools as a requirement of organization only. Developers do not seem motivated to use CASE tools and this lack of interest is one of the big factor in less use of CASE tools.

Technical Limitations-Every CASE tool follows a methodology for the model the system. People who use any specific CASE tool for a longer period of time get used with the methodology of that tool and they try to apply the same methodology for other projects. Most of the tools have their hardware and software requirements. These requirements should meet to use that tool. This is another hurdle in CASE tool adoption for those companies who does not meet hardware or software requirements to use CASE tools. Tool should not be platform dependent either of software or hardware.

Technology Transfer and Insertion-One of the main challenges in CASE adoption is technology change. CASE depends on planning, managing and early experiences and it can not be guaranteed even by experienced people that CASE can easily be incorporated.

Environment-Environment is another factor that plays important role in the failure of CASE tools. Is there any practice exists in the organization to adopt new technology and effectively learn it, opposition is always there for the new emerging technology and public don't like to change the existing technology, because they have to learn it before using it. Behavior of people towards change in organization is also very important, if developers and other related people in an organization are willing to implement and adopt new technology,

then there are fewer problems.

Selection of CASE Tool

It is also a big concern that an organization must choose a appropriate CASE tool that suite there organization, but unfortunality most of the time developer are not consulted and management use to select the CASE tool that is a big hurdle in the way of CASE tool penetration. Before selecting any new technology, people who uses them must be consulted because they can better argue that how they can effectively use the CASE tool. As there are many tools for same problem in the market tool selection is a big concern as well.

Conclusion- CASE tools plays an important role in Software Development and getting its appreciation in software development industry slowly but surly. But CASE is still in growth and there are different perceptions about CASE.CASE tools are not being used as they are expected to. Though, there are many new CASE tools available in the market and integrated CASE tools provides significant help and support for developers and programmers throughout SDLC in design and development activities but CASE tools have never been first choice for the majority of developers. Research about CASE usage in different organizations has shown that CASE is not always welcomed by developers and programmers and has never been as a tool of choice in organizations. And where CASE is used, the developers are using only minimal required functionality.

Practical 6: To perform unit testing and integration testing.

In computer programming, unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine if they are fit for use. Intuitively, one can view a unit as the smallest testable part of an application. In procedural programming, a unit could be an entire module, but it is more commonly an individual function or procedure. In object-oriented programming, a unit is often an entire interface, such as a class, but could be an individual method. Unit tests are short code fragments created by programmers or occasionally by white box testers during the development process. Ideally, each test case is independent from the others. Substitutes such as method stubs, mock objects, fakes, and test harnesses can be used to assist testing a module in isolation. Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended.

Benefits of Unit Testing: The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a strict, written contract that the piece of code must satisfy. As a result, it affords several benefits.

Finds problems early

Unit testing finds problems early in the development cycle.

In test-driven development (TDD), which is frequently used in both Extreme Programming and Scrum, unit tests are created before the code itself is written. When the tests pass, that code is considered complete. The same unit tests are run against that function frequently as the larger code base is developed either as the code is changed or via an automated process with the build. If the unit tests fail, it is considered to be a bug either in the changed code or the tests themselves. The unit tests then allow the location of the fault or failure to be easily traced. Since the unit tests alert the development team of the problem before handing the code off to testers or clients, it is still early in the development process.

Facilitates change

Unit testing allows the programmer to refactor code at a later date, and make sure the module still works correctly (e.g., in regression testing). The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified.

Readily available unit tests make it easy for the programmer to check whether a piece of code is still working properly. In continuous unit testing environments, through the inherent practice of sustained maintenance, unit tests will continue to accurately reflect the intended use of the executable and code in the face of any change. Depending upon established development practices and unit test coverage, up-to-the-second accuracy can be

maintained.

Simplifies integration

Unit testing may reduce uncertainty in the units themselves and can be used in a bottom-up testing style approach. By testing the parts of a program first and then testing the sum of its parts, integration testing becomes much easier.

An elaborate hierarchy of unit tests does not equal integration testing. Integration with peripheral units should be included in integration tests, but not in unit tests. Integration testing typically still relies heavily on humans testing manually; high-level or global-scope testing can be difficult to automate, such that manual testing often appears faster and cheaper.

Integration testing :To perform integration testing, first, all unit testing has to be completed. Upon completion of unit testing, integration testing begins. Integration testing is black box testing. The purpose of integration testing is to ensure distinct components of the application still work in accordance to customer requirements. Test cases are developed with the express purpose of exercising the interfaces between the components. This activity is carried out by the test team. Integration testing is considered complete, when actual results and expected results are either in line or differences are explainable, or acceptable, based on client input.

Step 1 - Identify Unit Interfaces: The developer of each program unit identifies and documents the unit's interfaces for the Responding to queries from terminals for information, Managing transaction data entered for processing, Obtaining, updating or creating transactions on computer files , Passing or receiving information from other logical processing units, Sending messages to terminals, Providing the results of processing to some output device or unit.

Step 2 - Reconcile Interfaces for Completeness: The information needed for the integration test template is collected for all program units in the software being tested. Whenever one unit interfaces with another, those interfaces are reconciled. For example, if program unit A transmits data to program unit B, program unit B should indicate that it has received that input from program unit A. Interfaces not reconciled are examined before integration tests are executed.

Step 3 - Create Integration Test Conditions: One or more test conditions are prepared for integrating each program unit. After the condition is created, the number of the test condition is documented in the test template.

Step 4 - Evaluate the Completeness of Integration Test Conditions: The following list of questions will help guide evaluation of the completeness of integration test conditions recorded on the integration testing template. This list can also help determine whether test conditions created for the integration process are complete.

Practical 7: To perform various white box and black box testing techniques

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing). In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. in-circuit testing (ICT).

White-box testing can be applied at the unit, integration and system levels of the software testing process. Although traditional testers tended to think of white-box testing as being done at the unit level, it is used for integration and system testing more frequently today. It can test paths within a unit, paths between units during integration, and between subsystems during a system-level test. Though this method of test design can uncover many errors or problems, it has the potential to miss unimplemented parts of the specification or missing requirements.

White-box test design techniques include the following code coverage criteria:

- Control flow testing
- Data flow testing
- Branch testing
- Statement coverage
- Decision coverage
- Modified condition/decision coverage
- Prime path testing
- Path testing

Basic Procedures: White-box testing's basic procedures involve the understanding of the source code that you are testing at a deep level to be able to test them. The programmer must have a deep understanding of the application to know what kinds of test cases to create so that every visible path is exercised for testing. Once the source code is understood then the source code can be analyzed for test cases to be created. These are the three basic steps that white-box testing takes in order to create test cases:

1. Input involves different types of requirements, functional specifications, detailed designing of documents, proper source code, security specifications.^[2] This is the preparation stage of white-box testing to layout all of the basic information.
2. Processing involves performing risk analysis to guide whole testing process, proper test plan, execute

test cases and communicate results.^[2] This is the phase of building test cases to make sure they thoroughly test the application the given results are recorded accordingly.

3. Output involves preparing final report that encompasses all of the above preparations and results.

Advantages:

White-box testing is one of the two biggest testing methodologies used today. It has several major advantages:

1. Side effects of having the knowledge of the source code is beneficial to thorough testing.
2. Optimization of code by revealing hidden errors and being able to remove these possible defects.
3. Gives the programmer introspection because developers carefully describe any new implementation.
4. Provides traceability of tests from the source, allowing future changes to the software to be easily captured in changes to the tests.
5. White box tests are easy to automate.
6. White box testing give clear, engineering-based, rules for when to stop testing.

Disadvantages:

Although white-box testing has great advantages, it is not perfect and contains some disadvantages:

1. White-box testing brings complexity to testing because the tester must have knowledge of the program, including being a programmer. White-box testing requires a programmer with a high-level of knowledge due to the complexity of the level of testing that needs to be done.
2. On some occasions, it is not realistic to be able to test every single existing condition of the application and some conditions will be untested.
3. The tests focus on the software as it exists, and missing functionality may not be discovered.

Black Box Testing Technique:

Black-box testing is a method of software testing that examines the functionality of an application without peering into its internal structures or workings. This method of test can be applied to virtually every level of software testing: unit, integration, system and acceptance. It typically comprises most if not all higher level testing, but can also dominate unit testing as well.

Test cases

Test cases are built around specifications and requirements, i.e., what the application is supposed to do. Test cases are generally derived from external descriptions of the software, including specifications, requirements and design parameters. Although the tests used are primarily functional in nature, non-functional tests may also be used. The test designer selects both valid and invalid inputs and determines the correct output without

any knowledge of the test object's internal structure.

Test design techniques

Typical black-box test design techniques include:

- Decision table testing
- All-pairs testing
- State transition analysis
- Equivalence partitioning
- Boundary value analysis
- Cause–effect graph
- Error guessing

Practical 8: Testing of a web site

Web Testing in simple terms is checking your web application for potential bugs before its made live or before code is moved into the production environment.

During this stage issues such as that of web application security, the functioning of the site, its access to handicapped as well as regular users and its ability to handle traffic is checked.

Web Application Testing Checklist:

Some or all of the following testing types may be performed depending on your web testing requirements.

Functionality Testing:

This is used to check if your product is as per the specifications you intended for it as well as the functional requirements you charted out for it in your developmental documentation. Testing Activities Included:

Test all links in your web pages are working correctly and make sure there are no broken links. Links to be checked will include –

- Outgoing links
- Internal links
- Anchor Links
- MailTo Links

Test Forms are working as expected. This will include-

- Scripting checks on the form are working as expected. For example- if a user does not fill a mandatory field in a form an error message is shown.
- Check default values are being populated
- Once submitted , the data in the forms is submitted to a live database or is linked to an working email address
- Forms are optimally formatted for better readability

Test Cookies are working as expected. Cookies are small files used by websites to primarily remember active

user sessions so you do not need to log in every time you visit a website. Cookie Testing will include

- Testing cookies (sessions) are deleted either when cache is cleared or when they reach their expiry.
- Delete cookies (sessions) and test that login credentials are asked for when you next visit the site.

Test HTML and CSS to ensure that search engines can crawl your site easily. This will include

- Checking for Syntax Errors
- Readable Color Schemas
- Standard Compliance. Ensure standards such W3C, OASIS, IETF, ISO, ECMA, or WS-I are followed.

Test business workflow- This will include

- Testing your end - to - end workflow/ business scenarios which takes the user through a series of webpage's to complete.
- Test negative scenarios as well, such that when a user executes an unexpected step, appropriate error message or help is shown in your web application.

2.Usability testing:

Usability testing has now become a vital part of any web based project. It can be carried out by testers like a small focus group similar to the target audience of the web application.

Test the site Navigation:

- Menus , buttons or Links to different pages on your site should be easily visible and consistent on all web pages

Test the Content:

- Content should be legible with no spelling or grammatical errors.
- Images if present should contain an "alt" text.

3.Interface Testing:

Three areas to be tested here are - Application , Web and Database Server

Application: Test requests are sent correctly to the Database and output at the client side is displayed correctly. Errors if any must be caught by the application and must be only shown to the administrator and not the end user.

Web Server: Test Web server is handling all application requests without any service denial.

Database Server: Make sure queries sent to the database give expected results.

Test system response when connection between the three layers (Application, Web and Database) cannot be established and appropriate message is shown to the end user.

4.Database Testing:

Database is one critical component of your web application and stress must be laid to test it thoroughly. Testing activities will include-

- › Test if any errors are shown while executing queries
- › Data Integrity is maintained while creating , updating or deleting data in database.
- › Check response time of queries and fine tune them if necessary.
- › Test data retrieved from your database is shown accurately in your web application

5. Compatibility testing.

Compatibility tests ensures that your web application displays correctly across different devices. This would include-

Browser Compatibility Test: Same website in different browsers will display differently. You need to test if your web application is being displayed correctly across browsers , javascript , AJAX and authentication is working fine. You may also check for Mobile Browser Compatibility.

The rendering of web elements like buttons , text fields etc changes with change in Operating System. Make sure your website works fine for various combination of Operating systems such as Windows , Linux , Mac and Browsers such as Firefox , Internet Explorer , Safari etc.

6. Performance Testing:

This will ensure your site works under all loads. Testing activities will include but not limited to -

- Website application response times at different connection speeds
- Load test your web application to determine its behavior under normal and peak loads
- Stress test your web site to determine its break point when pushed to beyond normal loads at peak time.

- Test if a crash occurs due to peak load , how does the site recover from such an event
- Make sure optimization techniques like gzip compression , browser and server side cache enabled to reduce load times

7. Security testing:

Security testing is vital for e-commerce website that store sensitive customer information like credit cards. Testing Activities will include-

- Test unauthorized access to secure pages should not be permitted
- Restricted files should not be downloadable without appropriate access
- Check sessions are automatically killed after prolonged user inactivity
- On use of SSL certificates , website should re-direct to encrypted SSL pages.

8. Crowd Testing:

You will select a large number of people (crowd) to execute tests which otherwise would have been executed a select group of people in the company. Crowd sourced testing is an interesting and upcoming concept and helps unravel many a unnoticed defects.