Subject Name: **Software Engineering**

Subject Code: **CS-403**

Semester: **4th**

NEED AND TYPES OF MAINTENANCE:

Software Maintenance: Software maintenance is an activity in which program is modified after it has been put into use. In this usually it is not preferred to apply major software changes to system's architecture. Maintenance is a process in which changes are implemented by either modifying the existing systems architecture or by adding new components into the system.

Need for maintenance:

Software maintenance is widely accepted part of SDLC now a day. It stands for all the modifications and updating done after the delivery of software product. There are number of reasons, why modifications are required, some of them are briefly mentioned below:

- **Market Conditions:** Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.
- **Client Requirements:** Over the time, customer may ask for new features or functions in the software.
- **Host Modifications:** If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.
- **Organization Changes:** If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.

Types of maintenance:

Various types of software maintenance are:

- **Corrective Maintenance:** Means the maintenance for correcting the software faults.
- **Adaptive Maintenance:** Means maintenance for adapting the change in environment (different system or operating systems).
- **Perfective Maintenance:** Means modifying or enhancing the system to meet the new requirements.
- **Preventive Maintenance:** This includes modifications and updating to prevent future problems of the software.

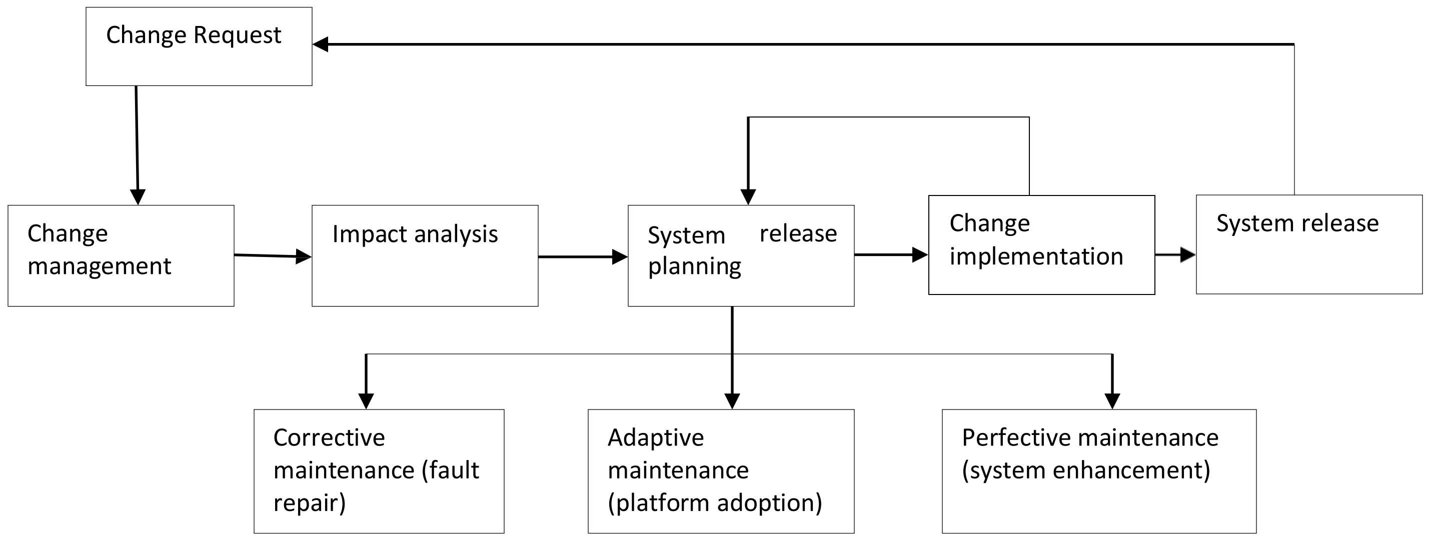
Change Request
Change management
Impact analysis
System release planning
Change implementation
System release
Corrective maintenance (fault repair)
Adaptive maintenance (platform adoption)
Perfective maintenance (system enhancement)

Figure 5.1: Maintenance Process

Issues in software maintenance:

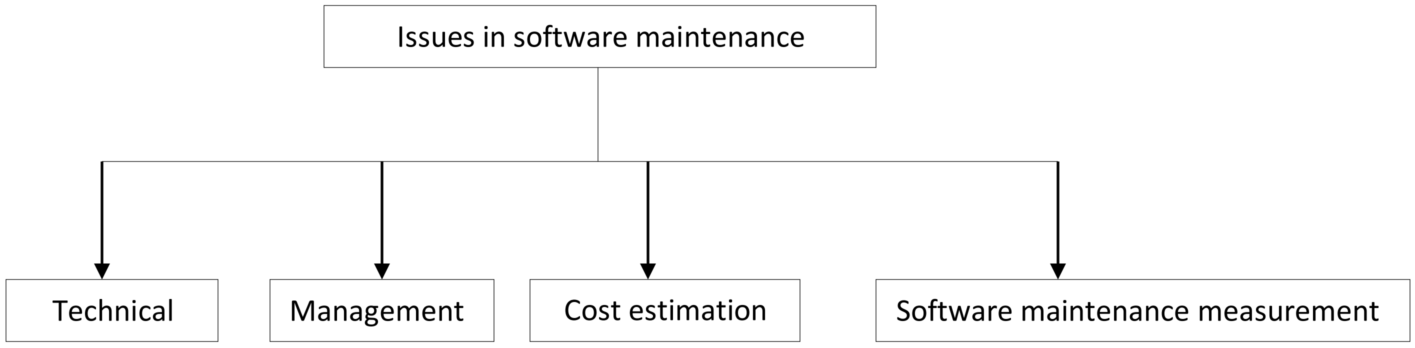
Issues in software maintenance
Technical
Management
Cost estimation
Software maintenance measurement

Figure 5.2: Software Maintenance Issues

SOFTWARE CONFIGURATION MANAGEMENT (SCM):

Software configuration management is a set of activities carried out for identifying, organizing and controlling changes throughout the life cycle of computer software.

During the development of software "change must be managed and controlled" in order to improve quality and reduce error.

Hence software configuration management is a quality assurance activity that is applied throughout the software process.

SCM helps to eliminate the confusion often caused by miscommunication among team members. The SCM system controls the basic components such as software objects, program code, test data, test output, design documents and user manuals.

The SCM system has the following advantages:

- Reduced redundant work.
- Effective management of simultaneous updates.
- Avoids configuration-related problems.
- Facilitates team coordination.
- Helps in building management; managing tools used in builds.
- Defect tracking: It ensures that every defect has traceability back to its source.

Software Configuration Items (SCIs): Information that is created as part of the software engineering process.
Baselines: A Baseline is a software configuration management concept that helps us to control change. Signals a point of departure from one activity to the start of another activity. Helps control change without impeding justifiable change.

Elements of SCM

There are four elements of SCM:

1. Software Configuration Identification
2. Software Configuration Control
3. Software Configuration Auditing
4. Software Configuration Status Reporting

The SCM Process

The SCM process defines a series of tasks:

- Identification of objects in the software configuration
- Version Control
- Change Control

- Configuration Audit, and
- Reporting

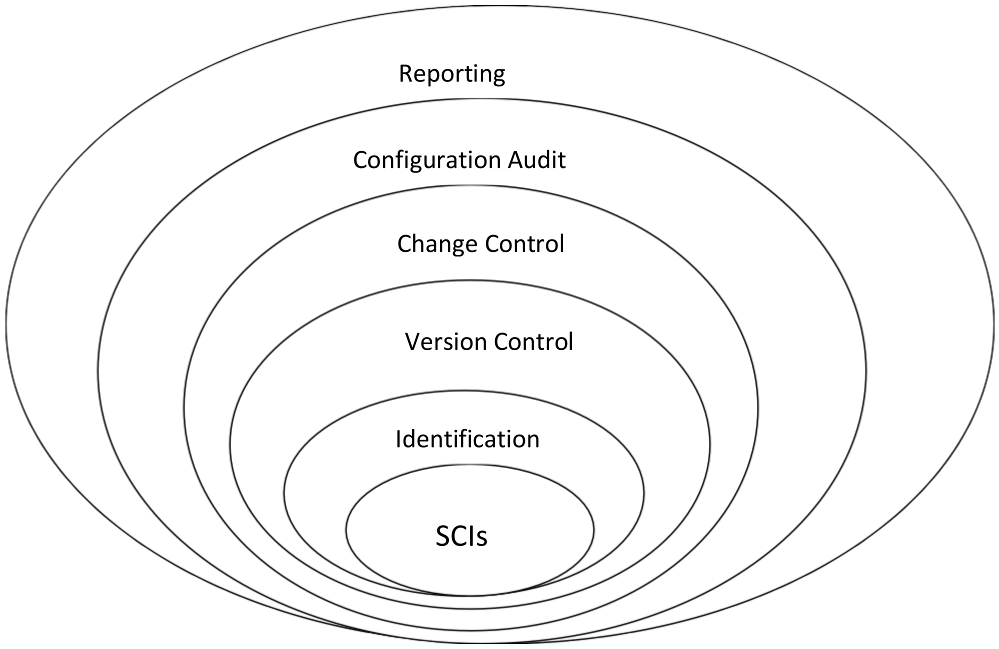

Figure 5.3: SCM Process

SOFTWARE CHANGE MANAGEMENT:

Change control is function of configuration management, which ensures that all changes made to software system are consistent and made as per organizational rules and regulations.

A change in the configuration of product goes through following steps –

- **Identification:** A change request arrives from either internal or external source. When change request is identified formally, it is properly documented.
- **Validation:** Validity of the change request is checked and its handling procedure is confirmed.
- **Analysis:** The impact of change request is analyzed in terms of schedule, cost and required efforts. Overall impact of the prospective change on system is analyzed.
- **Control:** It is decided that the changes are worth incorporation or not. If it is not, change request is refused formally.
- **Execution:** If the previous phase determines to execute the change request, this phase takes appropriate actions to execute the changes, through a thorough revision if necessary.
- **Close request:** The change is verified for correct implementation and merging with the rest of the system. This newly incorporated change in the software is documented properly and the request is formally closed.

VERSION CONTROL:

Version Control is a system or tool that captures the changes to a source code element: file, folder, image or binary. This is beneficial for many reasons, but the most fundamental reason is it allows you to track changes on a per file basis.

Version Control Benefits:

- Secure Access to your Source Code
- File History
- Facilitate Team Communication

- Baseline Trace Ability
- Automated Merge Capabilities
- Ensures no one Over-Writes Someone Else's Code
- Allows for Better Control for Parallel Development

CHANGE CONTROL AND REPORTING:

Change control is a systematic approach to managing all changes made to a product or system. The purpose is to ensure that no unnecessary changes are made, that all changes are documented, that services are not unnecessarily disrupted and that resources are used efficiently. Change control is an essential step in software life cycle. The change control can be carried out using following steps:

- A change request initiates a changes
- The configuration object is “checked out” of the database.
- The changes are applied to the object.
- The object is then “checked in” to the database where automatic version control is applied.

PROGRAM COMPREHENSION TECHNIQUES:

Program comprehension is a domain of computer science concerned with the ways software engineers maintain existing source code.

Program comprehension tools only play a supporting role in other software engineering activities of design, development, maintenance, and re-documentation. Software Engineering discipline which aims at understanding computer code written in a high-level programming language. Program Comprehension is useful for reuse, maintenance, reverse engineering and many other activities in the context of Software Engineering.

Program Comprehension Tool:

A program that aims at making the understanding of a software application easier, through the presentation of different perspectives (views) of the overall system or its components. A PC Tool has modules to:

- Extract information by parsing the source code
- Store and handle the information extracted
- Visualize all the retrieved information

RE-ENGINEERING:

Software re-engineering means re-structuring or re-writing part or all of the software engineering system. It is needed for the application which requires frequent maintenance.

Software re-engineering is a process of software development which is done to improve the maintainability of a software system.

Re-engineering a software system has two key advantages:

- **Reduced risk:** As the software already exists, the risk is less as compared to developing new software.
- **Reduced cost:** The cost of re-engineering is significantly less than the costs of developing new software.

Re-engineering process activities:

- **Source code translation:** In this phase code is converted into new language.
- **Reverse Engineering:** Under this activity the program is analyzed and understood thoroughly.
- **Program structure improvement:** Restructure automatically for understandability.
- **Program modularization:** The program structure is reorganized.
- **Data re-engineering:** Finally clean-up and restructure system data.

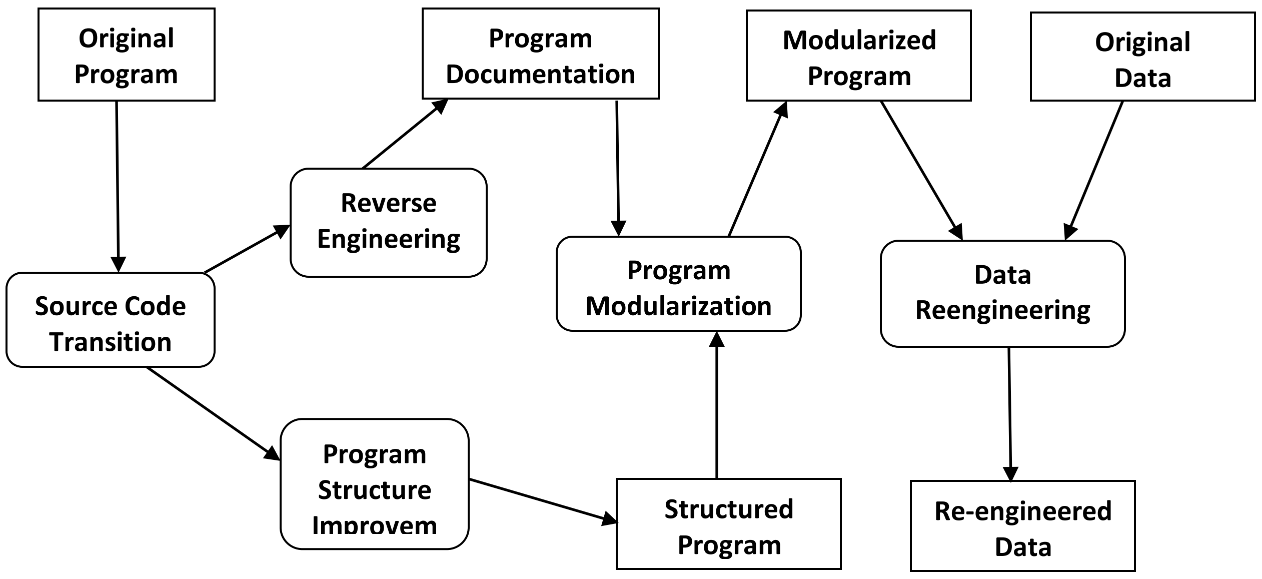

Figure 5.4: Re-engineering Process Activities

REVERSE ENGINEERING:

Reverse engineering is the process of design recovery. In reverse engineering the data, architectural and procedural information is extracted from a source code.

The reverse engineering is required because using this technique the dirty, ambiguous code can be converted to clear and unambiguous specification. This specification helps in understanding the source code.

There are 3 important issues in reverse engineering:

1. **Abstraction Level:** This level helps in obtaining the design information from the source code. It is expected that abstraction level should be high in reverse engineering.
2. **Completeness Level:** The completeness means detailing of abstract level. The completeness decreases as abstraction level increases.
3. **Directionality Level:** Directionality means extracting the information from source code and gives it to software engineer. The directionality can be one way or two way. The one way directionality means extracting all the information from source code and gives it to software engineer. The two way directionality means the information taken from source code is fed to re-engineering tool that attempts to restructure or regenerate old programs.

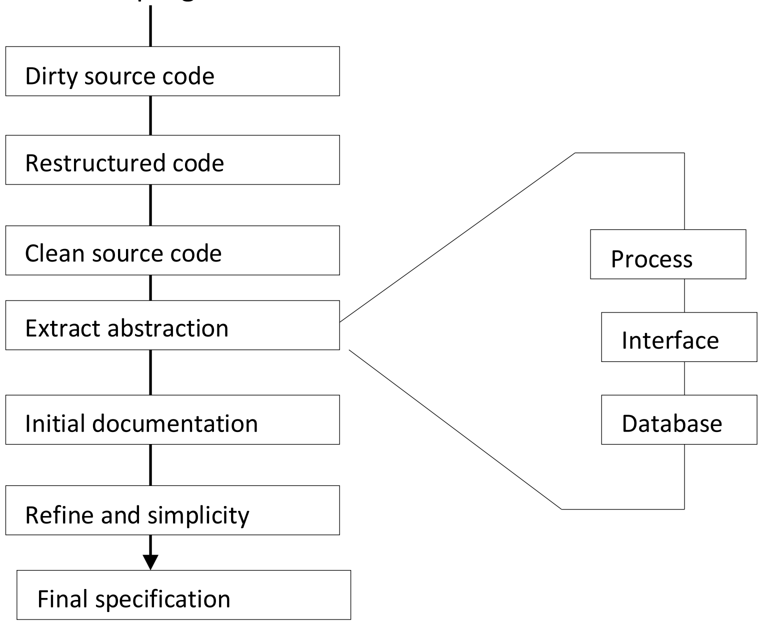

Figure 5.5: Reverse Engineering Process

Difference between reverse and forward and engineering:

S.No.	Reverse Engineering	Forward Engineering
1.	It performs transformation from a lower abstraction to higher one.	The traditional process of moving from high-level abstractions and logical, implementation independent designs to the physical implementation of a system.
2.	Usually done when docs are not appropriate or is missing.	Modification of the system is done. E.g. 1) Use of different programming language. 2) introduction of new DBMS 3) Transfer of s/w to new h/w platform.
3.	Reverse engineering is a process in which the dirty or unstructured code is taken, processed and it is restructured.	Forward-engineering is a process in which theories, methods and tools are applied to develop a professional software product.
4.	Reverse Engineering is trying to recreate the source code from the compiled code. That is trying to figure out how a piece of software works given only the final system.	Forward engineering is normal engineering. It builds devices that can do certain useful things for us:
5.	It is complex because cleaning the dirty or unstructured code requires more efforts.	It is simple and straight forward approach.
6.	Documentation or specification of the product is useful to the developer.	Documentation or specification of the product is useful to the end user.
7.	This process starts by understanding the existing unstructured code.	This process starts by understanding user requirements.

Table 5.1: Forward and Reverse Engineering

TOOL SUPPORT:

CASE Tool Support:

CASE tools are set of software application programs, which are used to automate SDLC activities. CASE tools are used by software project managers, analysts and engineers to develop software system.

There are number of CASE tools available to simplify various stages of Software Development Life Cycle such as Analysis tools, Design tools, Project management tools, Database Management tools, Documentation tools are to name a few.

Use of CASE tools accelerates the development of project to produce desired result and helps to uncover flaws before moving ahead with next stage in software development.

Components of CASE Tools

CASE tools can be broadly divided into the following parts based on their use at a particular SDLC stage:

- **Central Repository** - CASE tools require a central repository, which can serve as a source of common, integrated and consistent information. Central repository is a central place of storage where product specifications, requirement documents, related reports and diagrams, other useful information regarding management are stored. Central repository also serves as data dictionary.
- **Upper Case Tools** - Upper CASE tools are used in planning, analysis and design stages ofSDLC.
- **Lower Case Tools** - Lower CASE tools are used in implementation, testing and maintenance.
- **Integrated Case Tools** - Integrated CASE tools are helpful in all the stages of SDLC, from Requirement gathering to Testing and documentation.

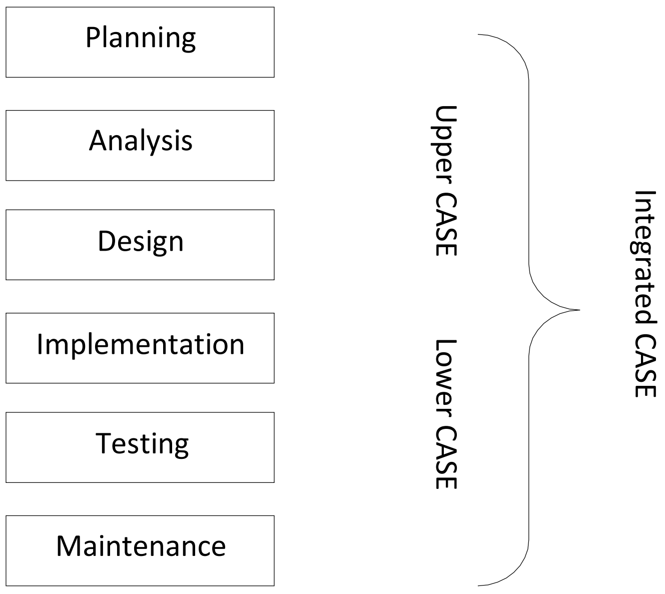

Figure 5.6: CASE Component

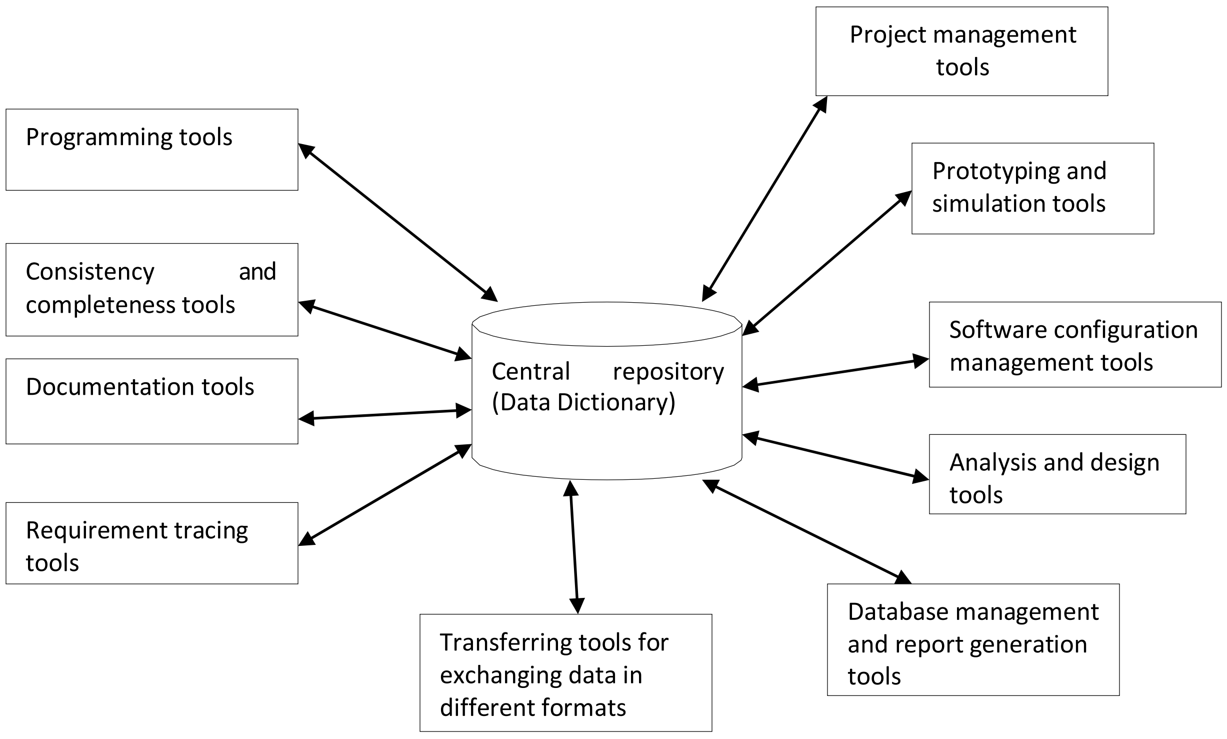
Project management tools
Programming tools
Prototyping and simulation tools
Consistency and completeness tools
Software configuration management tools
Central repository (Data Dictionary)
Documentation tools
Analysis and design tools
Requirement tracing tools
Database management and report generation tools
Transferring tools for exchanging data in different formats

Figure 5.7: Block Diagram for CASE Environment

<u>**PROJECT MANAGEMENT CONCEPTS:**</u>

Software project management is an activity of organizing, planning and scheduling the software projects. The goal of software project management is to deliver the software product in given time and within the budget. It is also necessary that the software project should be developed in accordance with the requirements of the organization. The project management is the application of knowledge, skill, tools and techniques to project activities to meet the project requirements.

Objectives of project management:

- The objective of the project planning and management is to provide a framework for the project.
- Using the project framework, the project manger decides the estimates for the schedule, cost and resources.
- Another objective of the project planning and management is that- it should be possible to get the best case and worst case outcomes of the project.
- There should be sufficient information discovery through the project so that reasonable project estimate can be made.

FEASIBILITY ANALYSIS:

When the client approaches the organization for getting the desired product developed, it comes up with a rough idea about what all functions of the software must perform and which all features are expected from the software.
Referencing to this information, the analysts do a detailed study about whether the desired system and its functionality are feasible to develop or not.
This feasibility study is focused towards goal of the organization. This study analyses whether the software product can be practically materialized in terms of implementation, contribution of project to organization, cost constraints, and as per values and objectives of the organization. It explores technical aspects of the project and product such as usability, maintainability, productivity, and integration ability.
The output of this phase should be a feasibility study report that should contain adequate comments and recommendations for management about whether or not the project should be undertaken.

PROJECT AND PROCESS PLANNING:

Project planning is part of project management, in which project manager should recognize the future problems in advance and should be ready with the tentative solution to those problems. A project plan must be prepared in advance from the available information. The project planning is an iterative process and it gets completed only on the completion of the project. This process is iterative because new information gets available at each phase of the project development. Hence the plan needs to be modified on regular basis for accommodation new requirements of the project.
Project planning is inherently uncertain as it must be done before the project is actually started. Therefore the duration of the tasks is often estimated through a weighted average of optimistic, normal, and pessimistic cases.
The main purpose of this phase is to plan time, cost, and resources adequately to estimate the work needed and to effectively manage risk. Initial planning generally consists of:

- Developing the scope statement
- Selecting the planning team
- Identifying deliverables
- Creating the work breakdown structure
- Identifying the activities needed to complete those deliverables
- Sequencing the activities in a logical way
- Estimating the resources needed
- Estimating the time needed
- Estimating the costs
- Developing the schedule
- Developing the budget
- Gaining formal approval to begin

RESOURCE ALLOCATIONS:

Once the objectives of the project management are achieved, the project management is to estimate the resources for the project. Various recourses of the project are:

- Human or people
- Reusable software components
- Hardware or software components

The resources are available in limited quantity and stay in the organization as a pool of assets. The shortage of resources hampers development of the project and it can lag behind the schedule. Allocating extra resources increases development cost in the end. It is therefore necessary to estimate and allocate adequate resources for the project.

Resource management includes:

- Defining proper organization project by creating a project team and allocating responsibilities to each team member.
- Determining resources required at a particular stage and their availability.
- Manage Resources by generating resource request when they are required and de-allocating them when they are no more needed.

SOFTWARE EFFORTS:

Project Estimation

For an effective management, accurate estimation of various measures is a must. With the correct estimation, managers can manage and control the project more efficiently and effectively.

Project estimation may involve the following:

- **Software size estimation:** Software size may be estimated either in terms of KLOC (Kilo Line of Code) or by calculating number of function points in the software. Lines of code depend upon coding practices. Function points vary according to the user or software requirement.
- **Effort estimation:** The manager estimates efforts in terms of personnel requirement and man-hour required to produce the software. For effort estimation software size should be known. This can either be derived by manager's experience, historical data of organization, or software size can be converted into efforts by using some standard formula.
- **Time estimation:** Once size and efforts are estimated, the time required to produce the software can be estimated. An effort required is segregated into sub categories as per the requirement specifications and interdependency of various components of software. Software tasks are divided into smaller tasks, activities or events by Work Breakthrough Structure (WBS). The tasks are scheduled on day-to-day basis or in calendar months. The sum of time required to complete all tasks in hours or days is the total time invested to complete the project.
- **Cost estimation:** This might be considered as the most difficult of all because it depends on more elements than any of the previous ones. For estimating project cost, it is required to consider –
    - Size of the software
    - Software quality
    - Hardware
    - Additional software or tools, licenses etc.
    - Skilled personnel with task-specific skills
    - Travel involved
    - Communication
    - Training and support

PROJECT SCHEDULING:

Project Scheduling in a project refers to roadmap of all activities to be done with specified order and within time slot allotted to each activity. Project managers tend to define various tasks and project milestones and then arrange them keeping various factors in mind. They look for tasks like in critical path in the schedule, which are necessary to complete in specific manner (because of task interdependency) and strictly within the time allocated. During the project scheduling the total work is separated into various small activities.

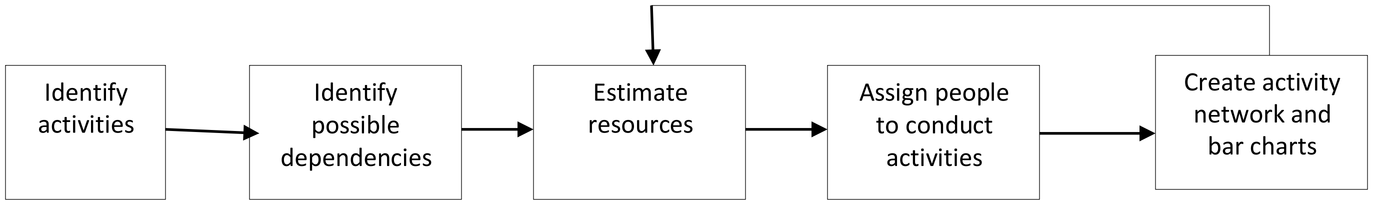
Identify activities
Identify possible dependencies
Estimate resources
Assign people to conduct activities
Create activity network and bar charts

Figure 5.8: Project scheduling process

For scheduling a project, it is necessary to:

- Break down the project tasks into smaller, manageable form
- Find out various tasks and correlate them
- Estimate time frame required for each task
- Divide time into work-units
- Assign adequate number of work-units for each task
- Calculate total time required for the project from start to finish

COST ESTIMATIONS:

Cost estimation can be defined as the approximate judgments of the costs for project. Cost estimation is usually measured in terms of effort. The effort is the amount of time for one person to work for a certain period of time. COCOMO is one the most widely used software estimation models in the world. The Constructive Cost Model (COCOMO) is a procedural software cost estimation model. COCOMO is used to estimate size, effort and duration based on the cost of the software.

COCOMO predicts the effort and schedule for a software product development based on inputs relating to the size of the software and a number of cost drivers that affect productivity.

COCOMO has three different models that reflect the complexities:

- **Basic Model:** this model would be applied early in a projects development. It will provide a rough estimate early on that should be refined later on with one of the other models.
- **Intermediate Model:** this model would be used after you have more detailed requirements for a project.
- **Detailed Model:** when design of the project is complete you can apply this model to further refine your estimate.

Within each of these models there are also three different modes. The mode you choose will depend on your work environment, and the size and constraints of the project itself. The modes are:

- **Organic:** this mode is used for “relativity small software teams developing software in a highly familiar, in-house environment”.
- **Embedded:** operating within tight constraints where the product is strongly tied to a “complex of hardware, software, regulations and operational procedures”.
- **Semi-detached:** an intermediate stage somewhere in between organic and embedded. Projects are usually of moderate size of up to 300,000 lines of code.

Basic Model: The basic COCOMO model estimates the software development effort using only Lines Of Code (LOC). Various equations in this model are:

$$\text{Effort Applied (E)} = a_b(\text{KLOC})^{b_b}[\text{man-months}]$$
$$\text{Development Time (D)} = c_b(\text{Effort Applied})^{d_b}[\text{months}]$$
$$\text{People required (P)} = \text{Effort Applied / Development Time [count]}$$

Where, KLOC is the estimated number of delivered lines (expressed in thousands) of code for project. The coefficients a_b, b_b, c_b and d_b are given in the following table:

Software Projects	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Table 5.2: List of Constants Based on Mode for Basic COCOMO

Intermediate Model: This is an extension of basic COCOMO model. This estimation model makes use of set of "cost driver attributes"to compute the cost of software.
The formula for effort calculation is:

$$E=a_i(KLOC)^{(b_i)}(EAF)$$

Where E is the effort applied in person-months, **KLOC** is the estimated number of thousands of delivered lines of code for the project, and **EAF** is the factor calculated above. The coefficient **a_i** and the exponent **b_i** are given in the next table.

Software Projects	a_i	b_i
Organic	3.2	1.05
Semi-detached	3.0	1.12
Embedded	2.8	1.20

Table 5.3: List of Constants Based on Mode for Intermediate Model

The Development time **D** calculation uses **E** in the same way as in the Basic COCOMO.
Detailed Model: Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step (analysis, design, etc.) of the software engineering process. The detailed model uses different effort multipliers for each cost driver attribute. These Phase Sensitive effort multipliers are each to determine the amount of effort required to complete each phase. In detailed COCOMO, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.
The effort is calculated as a function of program size and a set of cost drivers are given according to each phase of the software life cycle.

PROJECT SCHEDULING AND TRACKING:

Project schedule is the most important factor for software project manager. It is the duty of project manager to decide the project schedule and track the schedule.
Tracking the schedule means determine the tasks and milestone in the project as it proceeds.
Following are the various ways by which tracking of the project can be achieved:

- Conduct periodic meetings.
- Evaluate results of all the project reviews.
- Compare actual start date and scheduled start date of each of the project task.
- Determine if milestones of the project are achieved on scheduled date or not.
- Meet informally to the software practitioners.
- Assess the progress of the project quantitatively.

RISK ASSESSMENT AND MITIGATION:

Risk management involves all activities pertaining to identification, analyzing and making provision for predictable and non-predictable risks in the project. Risk may include the following:

- Experienced staff leaving the project and new staff coming in.
- Change in organizational management.
- Requirement change or misinterpreting requirement.
- Under-estimation of required time and resources.
- Technological changes, environmental changes, business competition.

Process of Risk Management:
Risk management performed in following stages:

1. **Risk identification:** In this phase all possible risks are anticipated and a list of potential risks are prepared.
2. **Risk analysis:** After risk identification, a list is prepared in which risks are prioritized.
3. **Risk planning:** The risk avoidance or risk minimization plan is prepared in this phase.
4. **Risk monitoring:** Identified risks must be mitigated. Hence risk mitigation plan must be prepared once the risks are discovered.

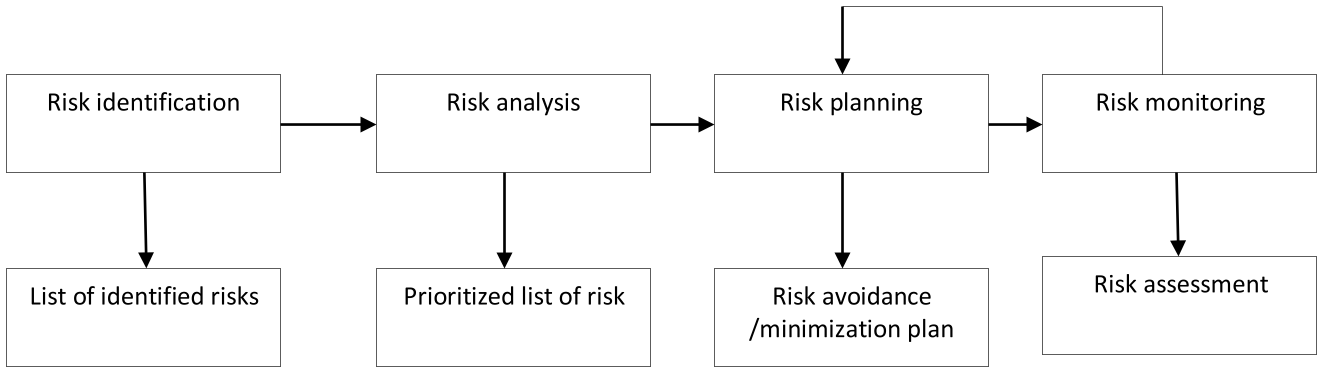

Figure 5.9: Risk management process

Management (RMMM):
Risk analysis supports the project team in constructing a strategy to deal with risks.
There are three important issues considered in developing an effective strategy:

- **Risk avoidance or mitigation -** It is the primary strategy which is fulfilled through a plan.
- **Risk monitoring -** The project manager monitors the factors and gives an indication whether the risk is becoming more or less.
- **Risk management and planning -** It assumes that the mitigation effort failed and the risk is a reality.

RMMM Plan:

- It is a part of the software development plan or a separate document.
- The RMMM plan documents all work executed as a part of risk analysis and used by the project manager as a part of the overall project plan.
- The risk mitigation and monitoring starts after the project is started and the documentation of RMMM is completed.

<u>SOFTWARE QUALITY ASSURANCE (SQA):</u>
Software Quality:
In the context of software engineering, software quality measures how well software is designed (quality of design), and how well the software conforms to that design (quality of conformance).

Quality Control:
Quality control (QC) is a procedure or set of procedures intended to ensure that a manufactured product or performed service adheres to a defined set of quality criteria or meets the requirements of the client or customer.

Quality Assurance:
It is planned and systematic pattern of activities necessary to provide a high degree of confidence in the quality of a product. It provides quality assessment of the quality control activities and determines the validity of the data or procedures for determining quality.

SQA Activities to Assure the Software Quality

The Software Quality Assurance of the software is analyzed and ensured by performing a series of activities. The activities are performed as step by step process and the result analysis is reported for the final evaluation process. The activities are performed as step by step process and the result analysis is reported for the final evaluation process.

- A Quality Management Plan is prepared
- Application of Technical Methods (Employing proper methods and tools for developing software)
- Conduct of Formal Technical Review (FTR)
- Testing of Software
- Enforcement of Standards (Customer imposed standards or management imposed standards)
- Control of Change (Assess the need for change, document the change)
- Measurement (Software Metrics to measure the quality, quantifiable)
- Records Keeping and Recording (Documentation, reviewed, change control etc. i.e. benefits of docs).

PROJECT PLAN:
A project plan is a formal document designed to guide the control and execution of a project. A project plan is the key to a successful project and is the most important document that needs to be created when starting any business project.
A project plan is used for the following purposes:

- To document and communicate stakeholder products and project expectations
- To control schedule and delivery
- To calculate and manage associated risks

PROJECT METRICS:
Metrics is a quantitative measure of the degree to which a system, component, or process possesses a given attribute.
Project metrics are quantitative measures that enable software engineers to gain insight into the efficiency of the software process and the projects conducted using the process framework. Project metrics are used by a project manager and a software team to adapt project work flow and technical activities.

Size Oriented Metrics:

- Size oriented measure is derived by considering the size of software that has been produced.
- The organization builds a simple record of size measure for the software projects. It is built on past experiences of organizations.
- It is a direct measure of software
- A simple set of size measure that can be developed is as given below:
    - Size= KLOC
    - Effort = Person/month

- Productivity=KLOC/ Person-month
- Quality= number of faults/KLOC
- Cost=$/KLOC
- Documentation= Pages of documentation/KLOC

Function Oriented Metrics:

- Use a measure of the functionality delivered by the application as a normalization value.
- Functionality cannot be measured directly; it must be derived indirectly using other direct measures.
- A measure called the function point.
- Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity.

How to calculate Function Point?

Domain Characteristics	**Count**		**Weighting factor**			**Count**
			Simple	**Average**	**Complex**	
Number of user input		x	3	4	6	
Number of user output		x	4	5	7	
Number of user inquiries		x	3	4	6	
Number of files		x	7	10	15	
Number of external interfaces		x	5	7	10	
Count Total						

Table 5.4: Function Point calculation table

Number of user inputs. Each user input that provides distinct application oriented data to the software is counted. Inputs should be distinguished from inquiries, which are counted separately.

Number of user outputs. Each user output that provides application oriented information to the user is counted. In this context output refers to reports, screens, error messages, etc. Individual data items within a report are not counted separately.

Number of user inquiries. An inquiry is defined as an on-line input that results in the generation of some immediate software response in the form of an on-line output. Each distinct inquiry is counted.

Number of files. Each logical master file (i.e., a logical grouping of data that may be one part of a large database or a separate file) is counted.

Number of external interfaces. All machine readable interfaces (e.g., data files on storage media) that are used to transmit information to another system are counted.

To compute function points (FP), the following relationship is used:

FP = count total [0.65 + 0.01 $\Sigma(F_i)$] where count total is the sum of all FP entries .

The Fi (i = 1 to 14) are "complexity adjustment values" based on responses to the following questions:

1. Does the system require reliable backup and recovery?
2. Are data communications required?

3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require on-line data entry?
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?
8. Are the master files updated on-line?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

Object Oriented Metrics:
Following are the set of metrics for object oriented projects:

1. **Weighted Methods per Class**
   Number of methods defined in class
   - Complex methods weigh more
   - Higher numbers = more faults
   - Classes are more specific = hard to reuse
   - Changes have more impact on subclasses
2. **Depth of Inheritance Tree**
   Number of super classes
   - Measures depth of hierarchy
   - Deep trees imply more complexity
   - Inheritance should reduce complexity not increase it
   - Deep trees promote reuse
   - Bugs are found in middle of tree
3. **Number of Children**
   Number of immediate subclasses
   - Measures breadth of hierarchy
   - Depth (DIT) implies reuse in a way breath (NOC) does not
   - Large numbers mean high reuse of base class; test it!
   - High NOC related to lower faults.
   - BUT, perhaps improper use of base class
4. **Coupling between Object Classes**
   Number of classes to which I am coupled
   - Coupling → We use each other’s data or methods
   - High CBO is undesirable
   - Prevents reuse
   - Sensitivity to changes in others increases maintenance
   - > 14 is too high

5. **Response For a Class**
   Total number of methods executed
   - Large RFC = more faults
   - If most methods have a small RFC but one method has a
   - Hugh RFC
   - you have a structured (not OO) application

6. **Lack of Cohesion of Methods**
   How well the methods of a class are are related to each other?
   - A cohesive class performs one function
   - Highly cohesive classes promotes encapsulation
   - but have highly coupled methods
   - Low cohesion = more errors