Module 6

Project Scheduling and Tracking & Software Configuration Management

PROJECT SCHEDULING AND TRACKING

Ques 1) What do you understand by project scheduling and tracking?

Ans: Project Scheduling and Tracking
Software project scheduling is an action that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks.

Software project scheduling is an activity that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks. It is important to note, however, that the schedule evolves over time.

During early stages of project planning, a macroscopic schedule is developed. This type of schedule identifies all major process framework activities and the product functions to which they are applied. As the project gets under way, each entry on the macroscopic schedule is refined into a detailed schedule. Here, specific software tasks (required to accomplish an activity) are identified and scheduled.

Project scheduling involves separating the total work involved in a project in separate activities and judging the time required to complete these activities as shown in **figure 6.1**.

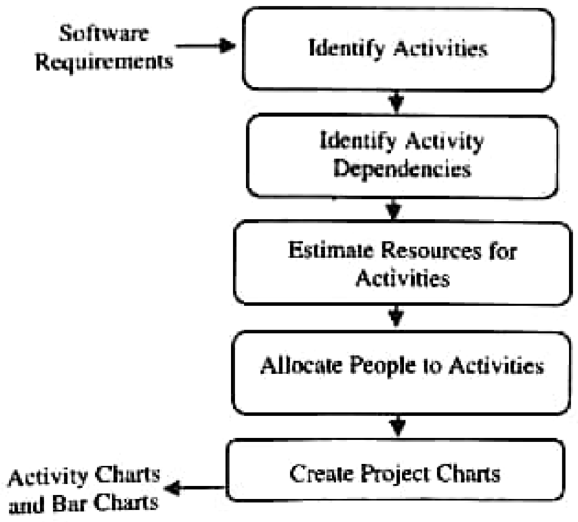

Software Requirements
Identify Activities
Identify Activity Dependencies
Estimate Resources for Activities
Allocate People to Activities
Create Project Charts
Activity Charts and Bar Charts

Figure 6.1: Project Scheduling Process

One of the important projects planning activity is project scheduling. It involves activities deciding which tasks should be done when:
1) Scheduling is an important activity for the project managers.
2) To determine project schedule:
   i) Identify tasks needed to complete the project.
   ii) Determine the dependency among different tasks.
   iii) Determine the most likely estimates for the duration of the identified tasks.
   iv) Plan the starting and ending dates for various tasks.

Ques 2) Discuss about the basic concepts of project scheduling?

Ans: Basic Concepts

Although there are many reasons why software is delivered late, most can be traced to one or more of the following root causes:

1) An unrealistic deadline established by someone outside the software development group and forced on managers and practitioner's within the group.
2) Changing customer requirements that are not reflected in schedule changes.
3) An honest underestimate of the amount of effort and/or the number of resources that will be required to do the job.
4) Predictable and/or unpredictable risks that were not considered when the project commenced.
5) Technical difficulties that could not have been foreseen in advance.
6) Human difficulties that could not have been foreseen in advance.
7) Miscommunication among project staff that results in delays.
8) A failure by project management to recognize that the project is falling behind schedule and a lack of action to correct the problem.

Aggressive (read "unrealistic") deadlines are a fact of life in the software business. Sometimes such deadlines are demanded for reasons that are legitimate, from the point of view of the person who sets the deadline. But the legitimacy must also be perceived by the people doing the work.

The following **steps** are recommended to handle lateness:

1) Perform a detailed estimate using historical data from past projects. Determine the estimated effort and duration for the project.
2) Using an incremental process model, develop a software engineering strategy that will deliver critical functionality by the imposed deadline, but delay other functionality until later. Document the plan.
3) Meet with the customer and (using the detailed estimate),
4) Offer the incremental development strategy as an alternative

Ques 3) What are the basic principles of software project scheduling?

Ans: Basic Principles of Software Project Scheduling

Each of these principles is applied as the project schedule evolves. Like all other areas of software engineering, a number of basic principles guide software project scheduling:

1) **Compartmentalization:** The project must be compartmentalized into a number of manageable activities, actions, and tasks. To accomplish compartmentalization, both the product and the process are decomposed.
2) **Interdependency:** The interdependency of each compartmentalized activity, action, or task must be determined. Some tasks must occur in sequence while others can occur in parallel. Some actions or activities cannot commence until the work product produced by another is available. Other actions or activities can occur independently.
3) **Time Allocation:** Each task to be scheduled must be allocated some number of work units (e.g., person-days of effort). In addition, each task must be assigned a start date and a completion date that are a function of the interdependencies and whether work will be conducted on a full-time or part-time basis.
4) **Effort Validation:** Every project has a defined number of people on the software team. As time allocation occurs, the project manager must ensure that not more than the allocated numbers of people have been scheduled at any given time. **For example,** consider a project that has three assigned software engineers (e.g., three person-days are available per day of assigned effort). On a given day, seven concurrent tasks must be accomplished. Each task requires 0.50 person days of effort. More effort has been allocated than there are people to do the work.
5) **Defined Responsibilities:** Every task that is scheduled should be assigned to a specific team member.
6) **Defined Outcomes:** Every task that is scheduled should have a defined outcome? For software projects, the outcome is normally a work product (e.g., the design of a module) or a part of a work product. Work products are often combined in deliverables.
7) **Defined Milestones:** Every task or group of tasks should be associated with a project milestone. A milestone is accomplished when one or more work products has been reviewed for quality and has been approved.

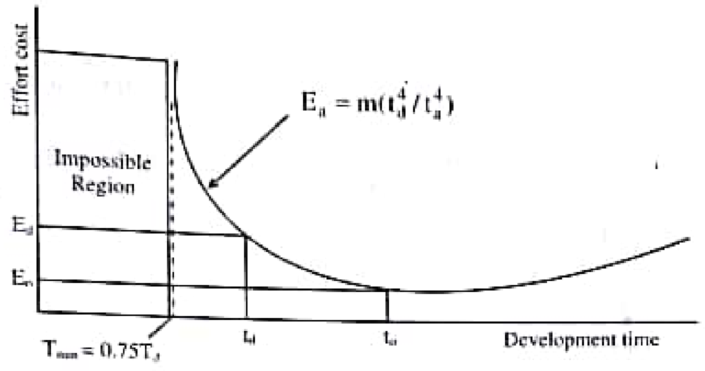
Ques 4) What is the relationship between people and effort?

Or

Discuss about the Putnam-Norden-Rayleigh (PNR) curve.

Ans: Relationship between People and Effort

The **Putnam-Norden-Rayleigh** (PNR) curve provides an indication of the relationship between effort applied and delivery time for a software project. A version of the curve, representing project effort as a function of delivery time, is shown in **figure 6.2**:



E_a = effort in person-months
t_d = nominal delivery time for schedule
t_o= optimal development time (in terms of cost)
t_a = actual delivery time desired

Figure 6.2: Relationship between Effort and Delivery Time

The curve indicates a minimum value, t_o, that indicates the least cost time for delivery (i.e., the delivery time that will result in the least effort expended). As we move left of t_o (i.e., as we try to accelerate delivery), the curve rises non-linearly.

For example, we assume that a project team has estimated a level of effort, E_a, will be required to achieve a nominal delivery time, t_d, that is optimal in terms of schedule and available resources. Although, it is possible to accelerate delivery, the curve rises very sharply to the left of t_d. In fact, the PNR curve indicates that the project delivery time cannot be compressed much beyond $0.75t_d$. If we attempt further compression, the project moves into "the impossible region" and risk of failure becomes very high. The PNR curve also indicates that the lowest cost delivery option, $t_o = 2t_d$. The implication here is that delaying project delivery can reduce costs significantly. This must be weighed against the business cost associated with the delay.

Ques 5) What is effort distribution?

Ans: Effort Distribution

A recommended distribution of effort across the software process is often referred to as the 40-20-40 rule. Forty per cent of all effort is allocated to front-end analysis and design.

A similar percentage is applied to back-end testing. You can correctly infer that coding (20 percent of effort) is de-emphasized.

This effort distribution should be used as a guideline only. The characteristics of each project must dictate the distribution of effort. Work expended on project planning rarely accounts for more than 2-3 per cent of effort, unless the plan commits an organization to large expenditures with high risk. Requirements analysis may comprise 10-25 per cent of project effort. Effort expended on analysis or prototyping should increase in direct proportion with project size and complexity. A range of 20 to 25 per cent of effort is normally applied to software design. Time expended for design review and subsequent iteration must also be considered.

Because of the effort applied to software design, code should follow with relatively little difficulty. A range of 15-20 per cent of overall effort can be achieved. Testing and subsequent debugging can account for 30-40 per cent of software development effort. The criticality of the software often dictates the amount of testing that is required. If software is human rated (i.e., software failure can result in loss of life), even higher percentages are typical.

Ques 6) Discuss how to define a task set for the software project?

Or

What is degree of rigor? What are the different adaptation criteria?

Ans: Defining a Task Set for the Software Project

A task set is a collection of software engineering work tasks, milestones, and work products that must be accomplished to complete a particular project. The task set should provide enough discipline to achieve high software quality. But, at the same time, it must not burden the project team with unnecessary work.

To develop a project schedule, a task set must be distributed on the project time line. The task set will vary depending upon the project type and the degree of rigor with which the software team decides to do its work. Although it is difficult to develop a comprehensive taxonomy of software project types, most software organizations encounter the following projects:

1) **Concept Development Projects:** These are initiated to explore some new business concept or application of some new technology.
2) **New Application Development Projects:** These are undertaken as a consequence of a specific customer request.
3) **Application Enhancement Projects:** These occur when existing software undergoes major modifications to function, performance, or interfaces that are observable by the end-user.
4) **Application Maintenance Projects:** These correct, adapt, or extend existing software in ways that may not be immediately obvious to the end-user.
5) **Re-Engineering Projects:** These are undertaken with the intent of re-building an existing (legacy) system in whole or in part.

Degree of Rigor

Degree of rigor is a function of many project characteristics. Four different degrees of rigor can be defined:

1) **Casual:** All process framework activities are applied, but only a minimum task set is required. In general, umbrella tasks will be minimised and documentation requirements will be reduced. All basic principles of software engineering are still applicable.
2) **Structured:** The process framework will be applied for this project. Framework activities and related tasks appropriate to the project type will be applied and umbrella activities necessary to ensure high quality will be applied. SQA, SCM, documentation, and measurement tasks will be conducted in a streamlined manner.
3) **Strict:** The full process will be applied for this project with a degree of discipline that will ensure high quality. All umbrella activities will be applied and robust work products will be produced.
4) **Quick Reaction:** The process framework will be applied for this project, but because of an emergency situation only those tasks essential to maintaining good quality will be applied. "Back-filling" (i.e., developing a complete set of documentation, conducting additional reviews) will be accomplished after the application/product is delivered to the customer.

Defining Adaptation Criteria

Project manager must develop a systematic approach for selecting the degree of rigor that is appropriate for a particular project. To accomplish this, project adaptation criteria are defined and a task set selector value is computed. Adaptation criteria are used to determine the recommended degree of rigor with which the software process should be applied on a project.

Eleven adaptation criteria are defined for software projects:

1) Size of the project
2) Number of potential users
3) Mission criticality
4) Application longevity
5) Stability of requirements
6) Ease of customer/developer communication
7) Maturity of applicable technology
8) Performance constraints
9) Embedded and non-embedded characteristics
10) Project staff
11) Reengineering factors

Ques 7) Explain how to select software engineering tasks.

Ans: Selecting Software Engineering Tasks

In order to develop a project schedule, a task set must be distributed on the project time line. The task set will vary depending upon the project type and the degree. Each of the project types may be approached using a process model that is linear sequential, iterative (e.g., the prototyping or incremental models), or evolutionary (e.g., the spiral model).

In some cases, one project type flows smoothly into the next. One considers the software engineering tasks for a concept development project. Concept development projects are initiated when the potential for some new technology must be explored. There is no certainty that the technology will be applicable, but a customer (e.g., marketing) believes that potential benefit exists. Concept development projects are approached by applying the following major tasks:

1) **Concept Scoping:** It determines the overall scope of the project.
2) **Preliminary Concept Planning:** It establishes the organization's ability to undertake the work implied by the project scope.
3) **Technology Risk Assessment:** It evaluates the risk associated with the technology to be implemented as part of project scope.
4) **Proof of Concept:** It demonstrates the viability of a new technology in the software context.
5) **Concept Implementation:** It implements the concept representation in a manner that can be reviewed by a customer and is used for "marketing" purposes when a concept must be sold to other customers or management.
6) **Customer Reaction:** Customer reaction to the concept solicits feedback on a new technology concept and targets specific customer applications.

Ques 8) What are the different scheduling techniques?

Or

Write short notes on the following

1) **Work Breakdown Structure (WBS)**
2) **Activity Charts**
3) **Gantt Chart**

Ans: Scheduling Techniques

Scheduling of a software project does not differ greatly from scheduling of any multitask engineering effort. Therefore, generalized project scheduling tools and techniques can be applied with modification for software projects.

Some types of scheduling techniques are as below:

1) **Work Breakdown Structure (WBS):** A work breakdown structure (WBS) is a hierarchical decomposition or breakdown of a project or major activity into successive levels, in which each level is a finer breakdown of the preceding one.

   In final form, a WBS is very similar in structure and layout to a document outline. Each item at a specific level of a WBS is numbered consecutively (**for example,** 10, 20, 30, 40, 50). Each item at the next level is numbered within the number of its parent item (**for example,** 10.1, 10.2, 10.3, 10.4).

   Most project control techniques are based on splitting the goal of the project into several intermediate goals. Each intermediate goal can in turn be split further. This process can be repeated until each goal is small enough to be well understood. Then each goal can be individually planned for its:

   i) Resource requirements
   ii) Assignment of responsibility
   iii) Scheduling, etc.

   A four level WBS diagram based on project phases is shown in **figure 6.3.**

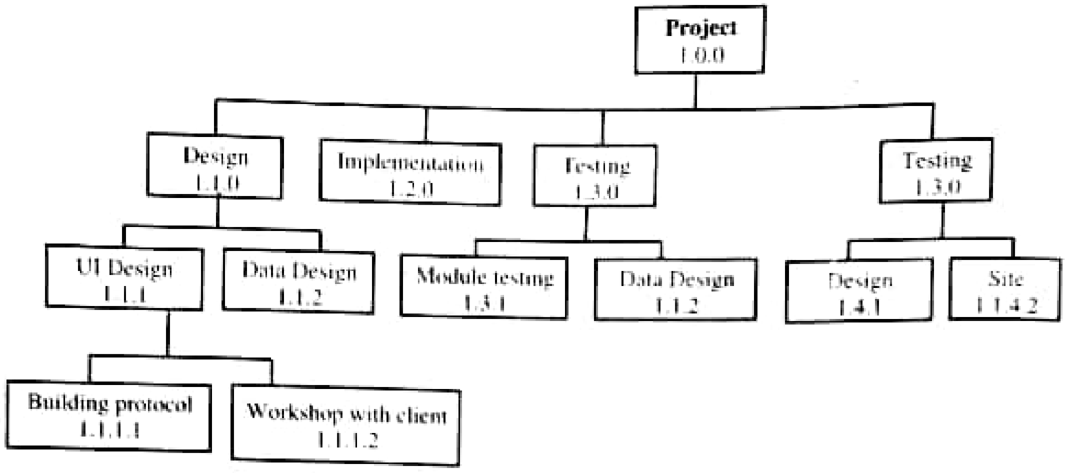


Figure 6.3: Four Level WBS Diagram based on Project Phases

Features of WBS

The major features of work breakdown structure (WBS) are described below:

i) **Structure**

a) WBS diagram is drawn like the organization chart.

b) Different desktop applications offer functionalities to easily create this kind of diagrams.

c) Work Breakdown Structure (WBS) provides a notation for representing task structure:

- Activities are represented as nodes of a tree.
- The root of the tree is labeled by the problem name.
- Each task is broken down into smaller tasks and represented as children nodes.

d) It is not useful to subdivide tasks into units, which take less than a week or two to execute.

e) Finer subdivisions mean that a large amount of time must be spent on estimating and chart revision.

ii) **Description**

a) Each WBS element should be described with a title.

b) The meaning of each title should be clear.

c) On the other hand, to restrict the size of the diagram (and to keep WBS effective method for communication) the length of the title might have to be restricted.

iii) **Coding**

a) One of the main features of WBS is the ability to code the different elements of work.

b) The coding system can be alphabetic, numeric or alphanumeric

iv) **Depth**

a) The recommended depth of a WBS diagram is three to four levels.

b) If deeper hierarchies are required the division into subprojects can be used and one element would then present one subproject.

c) This can be useful, e.g., in a situation where a subcontractor handles some part of the project.

d) Each project manager could then have his/her project presented in one diagram.

e) The downside of adding too many levels is firstly the readability of the diagram and secondly the fact that the larger the diagram, the more troublesome it is to update when major changes occur in the project.

v) **Level of Detail:** A rule of thumb in creating a WBS diagram is to make the lowest level element (often called work package) small enough to be considered a separate work element when estimating the amount of work in a project

2) **Activity Charts:** WBS structure can be refined into an activity network representation:

i) Network of boxes and arrows

ii) Shows different tasks making up a project.

iii) Represents the ordering among the tasks.

Development of WBS and activity network requires a thorough understanding of the tasks involved.

The activity network of tasks needed to complete a project, showing the order in which the tasks need to be completed and the dependencies between them. This is represented graphically, which is shown in **figure 6.4**

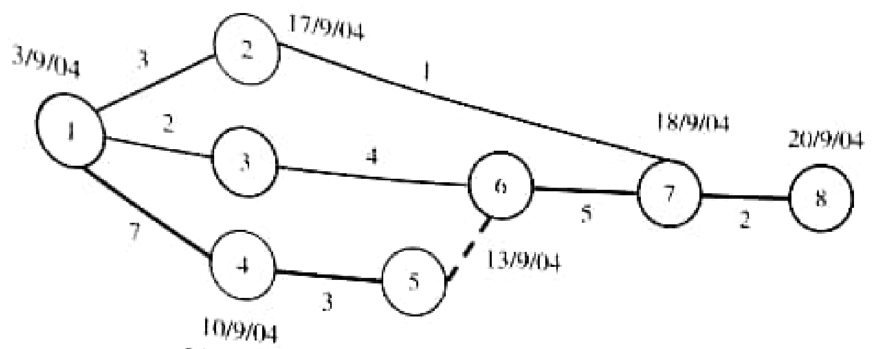


Figure 6.4: Example of an Activity Network

The diagram consists of a number of circles, representing events within the development lifecycle, such as the start or completion of a task, and lines, which represent the tasks themselves. Each task is additionally labeled by its time duration.

3) **Gantt Charts:** A Gantt chart is a horizontal bar chart developed as a production control tool named after Henry L. Gantt, an American engineer and social scientist. It is frequently used in project management.

A Gantt chart is useful for tracking and reporting progress, as well as for graphically displaying a schedule. Gantt charts are often used to report progress because they represent an easily understood picture of project status. However, Gantt Charts are not an ideal tool for project control.

Gantt chart provides a graphical illustration of a schedule that helps to plan, coordinate, and track specific in a project. Gantt charts may be simple versions created on graph paper or more complicated, automated versions created using project management tools. Project management tools incorporating Gantt Charts include PRINCE, MacProject, Microsoft Project and Microsoft Excel.

Purpose of a Gantt chart is to present a project schedule that shown the relationship of activities over time. Gantt charts are a Project planning tool that can be used to represent the timing of tasks required to complete a project. Because Gantt charts are simple to understand and easy to construct, they are used by most project managers for all but the most complex projects.

Gantt Chart Representation

i) Each bar represents an activity.
ii) Bars are drawn against a time line.
iii) Length of each bar is proportional to the length of time planned for the activity.
iv) Are not specific to software engineering.
v) Used in software project management are:
   a) Enhanced version of standard Gantt charts.
   b) Colored part of a bar shows the length of time a task is estimated to take.
   c) White part shows the slack time.
   d) The latest time by which a task must be finished.

For example, consider the following Gantt Chart of Software Development Process.

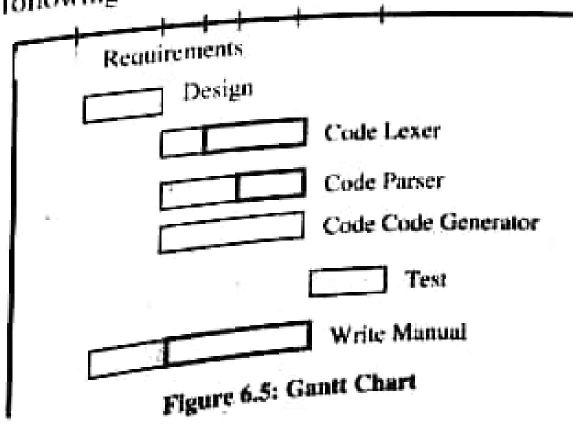


Figure 6.5: Gantt Chart

Ques 9) Write short notes on the following
1) Project Evaluation Review Technique (PERT):
2) Critical Path Method (CPM)

Ans: Project Evaluation Review Technique (PERT)
A Project (or Program) Evaluation and Review Technique (PERT) chart is a project management tool used to schedule, organize, and coordinate tasks within a project. PERT is a methodology developed by the U.S. Navy in the 1950s to manage the Polaris submarine missile program.

A similar methodology, the Critical Path Method (CPM), which was developed, for project management in the private sector at about the same time, has become synonymous with PERT, so that the technique is known by any variation on the names: PERT, CPM, or PERT/CPM.

Unlike Gantt charts, PERT can be both a cost and a time management systems. PERT is organized by events and activities or tasks.

PERT is designed for research and development type projects when activity completion times are uncertain. The heart of any PERT chart is a network of tasks needed to complete between them.

PERT charts depict task, duration, and dependency information. Each chart starts with an initiation node from which the first task, or tasks, originates. If multiple tasks begin at the same time, they are all started from the node or branch, or fork out from the starting point. Each task is represented by a line, which states its name or other identifier, its duration, the number of people assigned to it, and in some cases the initials of the personnel assigned. The other end of the task line is terminated by another node, which identifies the start of another task, or the beginning of any slack time, that is, waiting time between tasks.

Steps in Drawing a PERT Chart
Step 1: Make a list of the project tasks
Step 2: Assign a task identification letter to each task
Step 3: Determine the duration time for each task
Step 4: Draw the PERT network, number each node, label each task with its task identification letter, connect each node from start to finish, and put each task's duration on the network.
Step 5: Determine the need for any dummy tasks
Step 6: Determine the earliest completion time for each task node
Step 7: Determine the latest completion time for each task node
Step 8: Verify the PERT network for correctness

Critical Path Method (CPM)
Critical Path Method (CPM) charts are similar to PERT charts and are sometimes known as PERT/CPM. CPM acts
as the basis both for preparation of a schedule, and of resource planning. They were developed in the 1950s to control
large defense projects, and have been used routinely since then. During management of a project, it allows to monitor
the achievements of project goals. It also helps to see where remedial action needs to be taken to get a project back
on course.

In a CPM chart, the critical path is indicated. Critical Path is the path of longest duration as determined on a project
network diagram. The critical path determines the total duration of the project. If a task on the critical path is
delayed, the final completion of the project will likely be delayed. Critical Path Management (CPM) is a technique
for:
1) Identifying critical paths
2) Managing project.

The CPM technique is not specific to software engineering has a much wider use.

CPM can assist in answering questions like:
1) What are the critical paths in the project?
2) What is the shortest time in which the project can be completed?
3) What is the earliest (or latest) time a task can be started (or finished) without delaying

The critical path is "critical" because tasks that follow a critical task cannot be started until all of the previous tasks on the critical path are completed. Thus, if a task on the critical path is delayed, all tasks following the delayed critical task will be pushed out in time.

The critical tasks will have starting and finishing times that are fixed relative to the start of the project. Tasks not on the critical path will usually have some flexibility relative to when they can start and finish. This flexibility is called "float", or sometimes "slack". Float is the difference between the time available for performing a task and time required to complete a task.

Ques 10)What are the different ways to track the schedule.

Ans: Tracking the Schedule

The project schedule provides a road map for a software project manager. If it has been properly developed, the project schedule defines the tasks and milestones that must be tracked and controlled as the project proceeds.

Tracking can be accomplished in a number of different ways:

1) Conducting periodic project status meetings in which each team member reports progress and problems.
2) Evaluating the results of all reviews conducted throughout the software engineering process.
3) Determining whether formal project milestones have been accomplished by the scheduled date.
4) Comparing actual start-date to planned start-date for each project task listed in the resource table.
5) Meeting informally with practitioners to obtain their subjective assessment of progress to date and problems on the horizon.
6) Using earned value analysis to assess progress quantitatively.

Ques 11)What is earned value analysis?

Ans: Earned Value Analysis

The qualitative approach to project tracking provides the project manager with an indication of progress, but an assessment of the information provided is somewhat subjective. It is reasonable to ask whether there is a quantitative technique for assessing progress as the software team moves through the work tasks allocated to the project schedule. In fact, a technique for performing quantitative analysis of progress does exist. It is called Earned Value Analysis (EVA).

The earned value system provides a common value scale for every (software project) task, regardless of the type of work being performed. The total hours to do the whole project are estimated, and every task is given an earned value based on its estimated percentage of the total.

To determine the earned value, the following steps are performed:

1) The Budgeted Cost of Work Scheduled (BCWS) is determined for each work task represented in the schedule. During estimation, the work (in person-hours or person-days) of each software engineering task is planned. Hence, BCWS, is the effort planned for work task i. To determine progress at a given point along the project schedule, the value of BCWS is the sum of the $BCWS_i$ values for all work tasks that should have been completed by that point in time on the project schedule.
2) The BCWS values for all work tasks are summed to derive the budget at completion, BAC. Hence,

   $BAC = \Sigma(BCWS_k)$ for all tasks k
3) Next, the value for Budgeted Cost of Work Performed (BCWP) is computed. The value for BCWP is the sum of the BCWS values for all work tasks that have actually been completed by a point in time on the project schedule.

SOFTWARE CONFIGURATION MANAGEMENT (SCM)

Ques 12)What is software configuration management (SCM)?

Ans: Software Configuration Management (SCM)

Software configuration management (SCM) is an umbrella activity that is applied throughout the software process. Because change can occur at any time, SCM activities are developed to:

1) Identify change,
2) Control change,
3) Ensure that change is being properly implemented, and
4) Report changes to others who may have an interest.

It is important to make a clear distinction between software support and software configuration management. Support is set of software engineering activities that occur after software has been delivered to the customer and put into operation. Software configuration management is a set of tracking and control activities that begin when a software engineering project begins and terminate only when the software is taken out of operation.

Configuration management is the art of identifying, organizing, and controlling modifications to the software being built by a programming team. The goal is to maximize productivity by minimizing mistakes.

Software configuration management (SCM) is a set of activities designed to control change by identifying the work products that are likely to change, establishing relationships among them, defining mechanisms for managing different versions of these work products, controlling the changes imposed, and auditing and reporting on the changes made.

The output of the software process is information that may be divided into three broad categories:
1) Computer programs (both source level and executable forms)
2) Documents that describe the computer programs (targeted at both technical practitioners and users).
3) Data (contained within the program or external to it)

The items that comprise all information produced as part of the software process are collectively called **software configuration** as shown in **figure 6.6**:

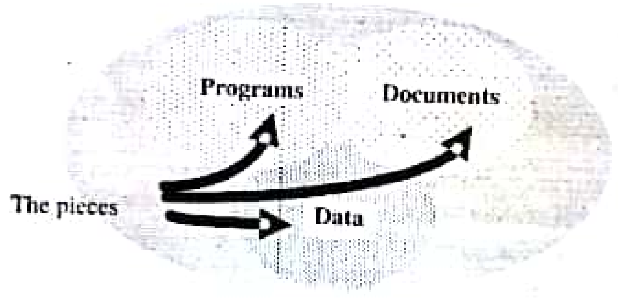


Figure 6.6: Software Configuration

As the software process progresses, the number of software configuration items (SCIs) grows rapidly. A system specification spawns a software project plan and software requirement specification (as well as hardware related documents).

These in turn spawn other documents to create a hierarchy of information. If each SCI simply spawned other SCIs, little confusion would result. Unfortunately, another variable enters the process-change.

Change may occur at any time, for any reason. There are four fundamental sources of change:
1) New business or market conditions.
2) New customer needs demand modification.
3) Reorganization or business growth/downsizing.
4) Budgetary or scheduling constraints.

Thus, software configuration management is a set of activities that have been developed to manage change throughout the life cycle of computer software. SCM can be viewed as a software quality assurance activity that is applied throughout the software process. **Figure 6.7** shows layers of SCM process.

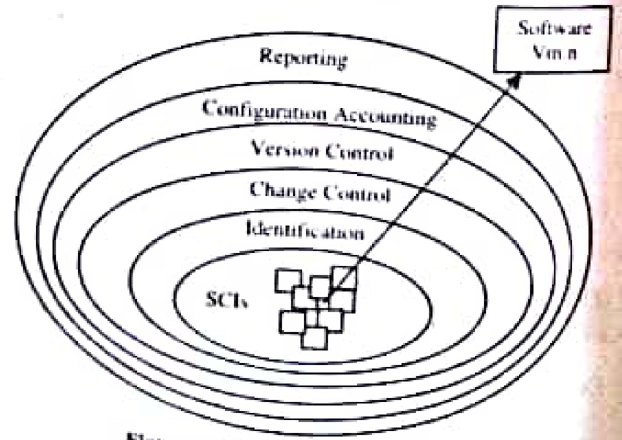


Figure 6.7: Layers of the SCM Process

Ques 13) What are the basic concepts of software configuration management?

Ans: Basic Concept of SCM
1) **Configuration Item:** A piece of software or work product which is subject to change is a configuration item.

2) **Change Request:** It is a formal report that contains the request for modification in a configuration item.
3) **Versions and Configurations:** A version identifies the state of a particular configuration item or a configuration at a well-defined point in time. An initial release or re-release of a configuration item associated with a complete compilation or recompilation of the item. Different versions have different functionality.
4) **Promotion:** A promotion is a version of a configuration item/CM aggregate that has been available to other developers in a project.
5) **Revision:** Change to a version that corrects only errors in the design/code, but does not affect the documented functionality.
6) **Release:** A release is a version that has been available to the user or the client.
7) **Repository:** It stores the various releases of a CM item/aggregate.
8) **Workspace:** It is a library of promotions.

Ques 14)What is the need of configuration management?
Or
What are the elements of configuration management system?

Ans: Need of Configuration Management
1) To be able to identify the exact state of different deliverables at any time.
2) To avoid the problems associated with having replicated objects accessible by multiple engineers.
3) Controlling concurrent work on a module by different engineers. (Overwriting one engineer's work by another).
4) Letting different engineers work on related modules at the same time.
5) Keeping track of variants (versions) and helping fix bugs in them.
6) Storing versions and revisions efficiently.
7) Maintaining revision history (accounting).

Elements of Configuration Management System
Susan Dart identifies four important elements that should exist when a configuration management system is developed:
1) **Component Elements:** These are a set of tools coupled within a file management system (e.g., a database) that enable access to and management of each software configuration item.
2) **Process Elements:** These are a collection of procedures and tasks that define an effective approach to change management (and related activities) for all constituencies involved in the management, engineering, and use of computer software.
3) **Construction Elements:** These are a set of tools that automate the construction of software by ensuring that the proper set of validated components (i.e., the correct version) has been assembled.
4) **Human Elements:** These are used to implement effective SCM, the software team uses a set of tools and process features (encompassing other CM elements).

Ques 15)What is the Software configuration management plan?

Ans: Software Configuration Management Plan (SCMP)
Software configuration management plan starts during the early phases of a project. The outcome of the SCM planning phase is the Software Configuration Management Plan (SCMP) which might be extended or revised during the rest of the project.

The SCMP can either follow a public standard like the IEEE 828, or an internal (e.g. company specific) standard.
The Software Configuration Management Plan:
1) Defines the types of documents to be managed and a document naming scheme.
2) Defines who takes responsibility for the CM procedures and creation of baselines.
3) Defines policies for change control and version management.
4) Describes the tools which should be used to assist the CM process and any limitations on their use.
5) Defines the configuration management database used to record configuration information.

Outline of a Software Configuration Management Plan (SCMP, IEEE 828-1990) is shown below:

1) **Introduction:** Describes purpose, scope of application, key terms and references.
2) **Management (WHO?):** Identifies the responsibilities and authorities for accomplishing the planned configuration management activities.
3) **Activities (WHAT?):** Identifies the activities to be performed in applying to the project.
4) **Schedule (WHEN?):** Establishes the sequence and coordination of the SCM activities with project mile stones.
5) **Resources (HOW?):** Identifies tools and techniques required for the implementation of the SCMP.
6) **Maintenance:** Identifies activities and responsibilities on how the SCMP will be kept current during the lifecycle of the project.

Ques 16)What are the software configuration management activities?

Ans: Software Configuration Management Activities

Project manager performs the software configuration activities that comprise of few others. An automated tool can be used for configuration management.

A variety of automated SCM tools has been developed to aid in identification (and other SCM) of tasks. In some cases, a tool is designed to maintain full copies of only the most recent version. To achieve earlier versions (of documents or programs) changes (cataloged by the tool) are "subtracted" from the most recent version.

Principal Activities of Configuration Management

1) **Configuration Identification:** Configuration identification is the process of deciding what things needs to be placed under configuration control and what will be the relationship between them. An elementary unit during configuration identification is the **Software Configuration Item (SCI).**

   SCI (Software Configuration Item)

   An SCI is a document or artifact that is explicitly placed under configuration control and is regarded a basic unit for modification. The result (deliverables) of any large software development effort consists of a large number of objects:

   i) Source code,
   ii) Design document,
   iii) SRS document,
   iv) Test document,
   v) Software Project Management Plan (SPMP) document, etc.

   SCIs are organized to form configuration objects that may be cataloged in the project database with a single name.

2) **Configuration Control:** Configuration control is the process of managing changes. It is the part of configuration management that most directly affects day-to-day operations of the developers. Once an object goes under configuration control, any further changes require approval from a change control board (CCB). The CCB is a group of people responsible for configuration management. The CCB evaluates the request based on its effect on the project, and the benefit due to change.

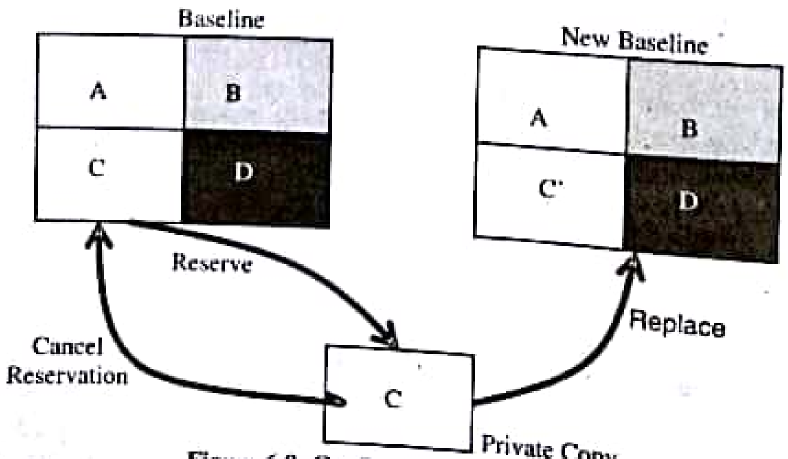


Figure 6.8: Configuration Control

An important reason for configuration control is people need a stable environment to develop a software product. **For example,** anyone is trying to integrate module A, with the modules B and C, he/she cannot make progress if developer of module C keeps changing it; this is especially frustrating if a change to module C forces you to recompile A. As soon as a document under configuration control is updated, the updated version is frozen and is called a baseline, as shown in **figure 6.8.**

3) **Configuration Accounting and Reporting:** Configuration accounting can be explained by two concepts:

i) **Status Accounting:** Once the changes in baselines occur, some mechanisms must be used to record how the system evolves and what is its current state. This task is accomplished by status accounting. Its basic function is to record the activities related to the other SCM functions.

Configuration status reporting (sometimes called status accounting) is an SCM task that answers the following questions:

a) What happened?

b) Who did it?

c) When did it happen?

d) What else will be affected?

The flow of information for configuration status reporting (CSR) is illustrated in figure of change control process above. Each time an SCI is assigned new or updated identification, a CSR entry is made. Each time a change is approved by the CCB (i.e, an ECO is issued), a CSR entry is made. Each time a configuration audit is conducted, the results are reported as part of the CSR task. The key elements under status accounting are shown in **figure 6.9.**

ii) **Configuration Audit:** A software configuration audit complements the formal technical review by assessing a configuration object for characteristics that are generally not considered during review. The audit asks and answers the following questions:

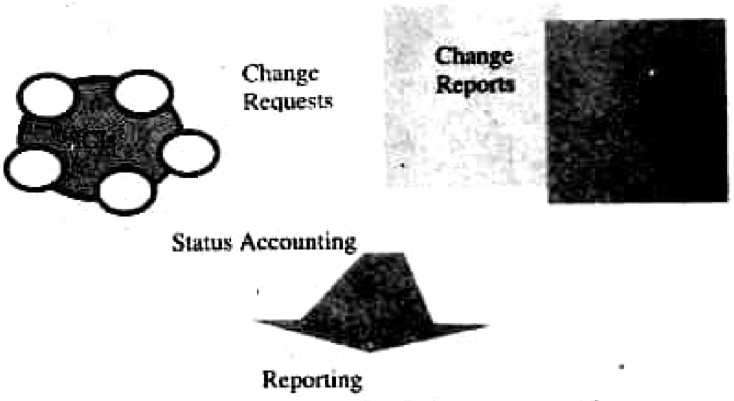


Figure 6.9: Status Accounting

a) Has the change specified in the ECO been made? Have any additional modifications been incorporated?

b) Has a formal technical review been conducted to assess technical correctness?

c) Has the software process been followed and have software engineering standard been properly applied?

d) Has the change been "highlighted" in the SCI? Have the change date and change author been specified? Do the attributes of the configuration object reflect the change?

e) Have SCM procedures for noting the change, recording it, and reporting it been followed?

f) Have all related SCIs been properly updated?

SCM Audit revolves around the software configuration items, change requests and software quality assurance plan.

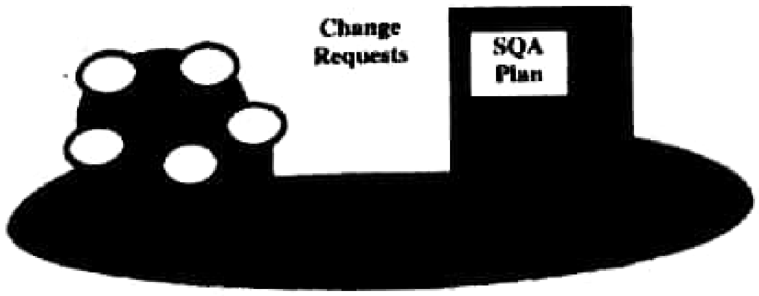


Figure 6.10: Configuration Audit

Ques 17) What do you mean by change control process and software version control?

Ans: Change Control Process

At any moment, the configuration-management team must know the state of any component or document in the system. Consequently, configuration management should emphasise communication among those whose actions affect the system. **Cashman** and **Holt** (1980) suggest that one should always know the answers to the following questions:

1) **Synchronisation:** When was the change made?
2) **Identification:** Who made the change?
3) **Naming:** What components of the system were changed?
4) **Authentication:** Was the change made correctly?
5) **Authorisation:** Who authorised that the change be made?
6) **Routing:** Who was notified of the change?
7) **Cancellation:** Who can cancel the request for a change?
8) **Delegation:** Who is responsible for the change?
9) **Valuation:** What is the priority of the change?

Notice that these questions are management questions, not technical ones. One must use procedures to manage change carefully.

Change control combines human procedures and automated tools to provide a mechanism for the control of change. The change control process is illustrated schematically in **figure 6.11.**

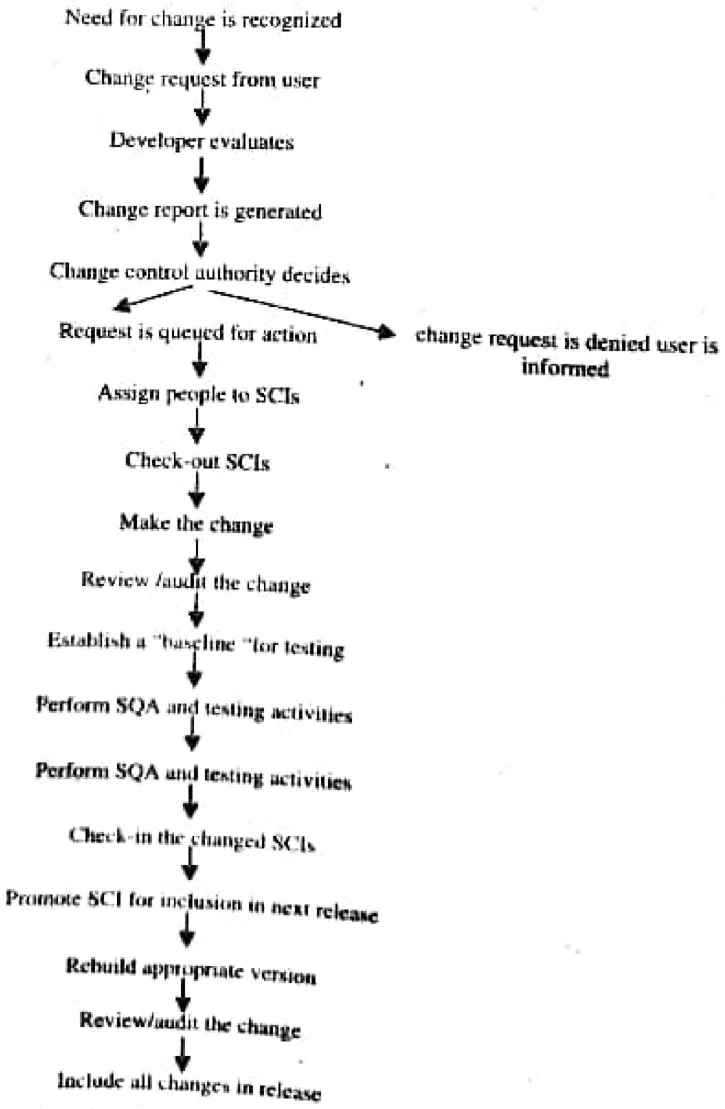

Need for change is recognized
Change request from user
Developer evaluates
Change report is generated
Change control authority decides
Request is queued for action
change request is denied user is informed
Assign people to SCIs
Check-out SCIs
Make the change
Review /audit the change
Establish a "baseline "for testing
Perform SQA and testing activities
Perform SQA and testing activities
Check-in the changed SCIs
Promote SCI for inclusion in next release
Rebuild appropriate version
Review/audit the change
Include all changes in release

Figure 6.11: Change Control Process

Software Version Control

Version control combines procedures and tools to manage different versions of configuration objects that are created during the software process.

"Configuration management allows a user to specify alternative configurations of the software system through the selection of appropriate versions. This is supported by associating attributes with each software version, and then allowing a configuration to be specified [and constructed] by describing the set of desired attributes

During the process of software evolution, many objects are produced. **For example,** files, electronic documents, paper documents, source code, executable code, and bitmap graphics. A version control is the first stage towards being able to manage multiple versions. Once it is in place, a detailed record of every version of the software must be kept. This report includes:

1) The name of each source-code component, including the variations and revisions.
2) The version of the various compilers and linkers used.
3) The name of the software staff who constructed the component.
4) The data and the time at which it was constructed.

The following evolutionary graph (**figure 6.12**) shows the evolution of a configuration item during the development life-cycle. The initial version of the item is given version number Ver 1.0. Subsequent changes to the item which could be mostly fixing bugs or adding minor functionality is given as Ver 1.1 and Ver 1.2.

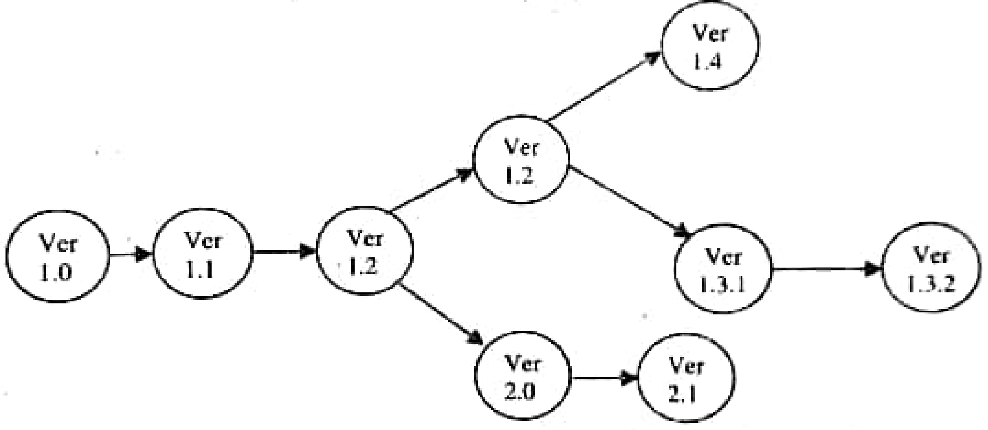


Figure 6.12: An Evolutionary Graph for a Different Version of an Item

After that, a major modification to Ver 1.2 is given a number Ver 2.0, and at the same time, a parallel version of the same item without the major modification is maintained and given a version number 1.3.

Commercial tools are available for version control, which perform one or more of the following tasks:

1) Source-code control
2) Revision control
3) Concurrent-version control

There are many commercial tools, such as Rational Clear Case, Microsoft Visual Source Safe, and a number of other tools to help version control.

Ques 18) Write short note on SCM Standards?

Ans: SCM Standards

Over the past two decades a number of software configuration management standards have been proposed. Many early SCM standards, such as MIL-STD-483, DODSTD- 480A and MIL-STD-1521A, focused on software developed for military applications.

However, more recent ANSI/IEEE standards, such as ANSI/IEEE Standards. No. 828-1983, No. 1042-1987, and Std. No. 1028-1988 [IEE94], are applicable for non-military software and are recommended for both large and small software engineering organizations.

USER INTERFACE DESIGN

Ques 19) What do you understand by User Interface Design?

Ans: User Interface Design
The communication interface between a computer and a human using the User Interface Design (UID) is created. This has been shown in **figure 6.13**. A screen layout that forms the base of a user interface prototype starts after the interface actions and objects are identified keeping in mind the set of interface design principles.

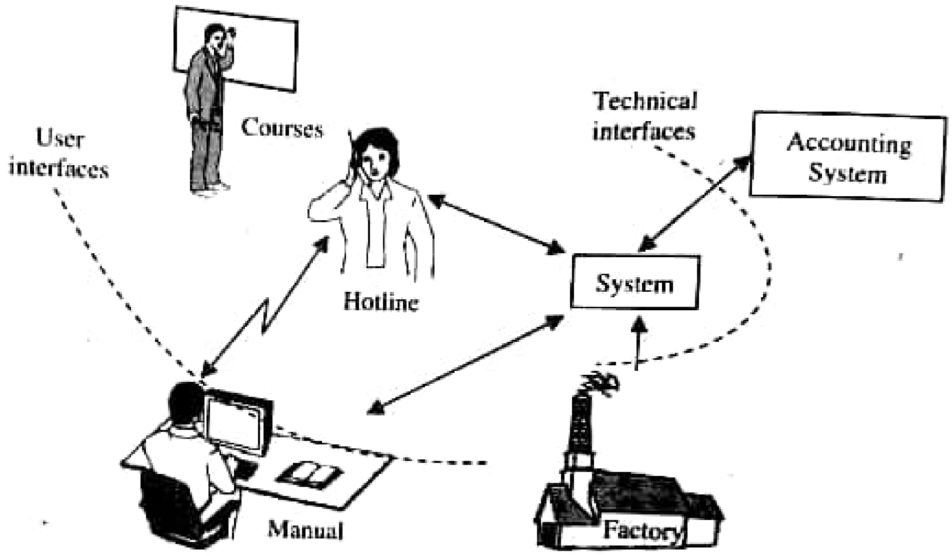


Figure 6.13

The method by which the user will interact with the software is determined by the user interface. An effective communication medium is created between computing machine and the human by the user interface design. With its assistance it is possible to provide a spontaneous and simple access to information. The software functionality can also be easily and efficiently controlled. For implementing this, the designer must have knowledge about the requirements and desires of the user from the user interface.

The part of the system that can be seen, felt or heard is known as the user interface. The part of the system like the database that accumulates the information is secreted from the user. The users can visualise whatever is happening behind the screen, however they are unable to see the secreted parts. This type of imagination is of great significance in those situations when the tasks are not performed in the usual manner by the system.

The user interface serves as the medium through which user interacts with the computer. Some of the common user interfaces for a standard PC are screen, loudspeaker, mouse and keyboard. In case the system is advanced then it will include interfaces such as special buttons, eye sensors with ability to detect where the person is looking at the screen, microphone to input voice, displays and lights, electronic gloves to track the movement of the fingers, etc.

Ques 20) Discuss about the user interface design process.

Ans: User Interface Design Process
Same as other software design elements, the user interface design process is also iterative in nature. Every stage of the user interface design process is done numerous times by refining and elaborating it by taking information from the last step.

Though various models have been proposed to design the user interface, the following steps are common for all of them:

1) Make use of the information that the requirements document contain. Definition of actions and each task are also taken into account.
2) Description of all the user actions or events that cause variation in the user interface is given.
3) Iteration is allotted to every user action.

4) The representation of the look of each user interface is given.
5) An indication of the user's understanding about the system's state is given with the help of the information obtained from the interface.

From the above text we can conclude that the initial steps of the interface design activity commences with user, task and environmental requirements identification. Then a set of actions and interface objects are described by generating and examining user states. Using these objects, elements such as icons, menus, screen layouts, etc., can be created.

Ques 21) What are the golden rules of user interface design?

Ans: Golden Rules of User Interface Design

Golden Rules consist of eight principles that can be applied to systems that are interactive. These principles must be certified and adjusted according to the design domain as they have been sophisticated since couple of decades and are derived from experience. Though the list is not complete but it is useful in providing guidance to designers and students. Following are the eight Golden Rules:

1) **Strive for Consistency:** Alike circumstances must have consistent series of actions; the prompts must use identical terminology; fonts, layout, help screens, menus, colour, capitalisation must be consistent throughout the system. Neither echoing of passwords nor exceptions like validation essential for executing the delete command should be inadequate in number and logical.

2) Cater to Universal Usability: The needs of the diverse users must be recognised and the design must be done in such a manner that transformation of content can be facilitated and plasticity can be ensured. The spectrum of requirements that guide the design is enriched by difference in designers such as expert or novice, technological diversity, range of age and other disabilities. The system quality can be improved and interface design can be enriched by adding explanations to help the novices and shortcuts to achieve faster pace.

3) **Offer Informative Feedback:** A feedback must be provided by the system for each user action. The response must be considerable in case the actions are major and occasional, whereas modest response can be provided for negligible and common actions. The variations can be presented unambiguously by providing a visual presentation of the objects.

4) **Design Dialogs to Yield Closure:** Clusters with a beginning, middle and end must be used to establish the series of actions. If informative feedback is provided at the end of the group containing actions then the operator gets a feeling of achievement and sense of relief. He is able to leave the contingency plans that are in his mind. He is also able to prepare himself for the next set of actions that he is going to perform.

   For example, in case a user is moving through an e-commerce website he starts with selecting products and comes out of the process by coming across a confirmation page which indicates the completion of the transaction.

5) **Prevent Errors:** The system must be designed in such a way that users are prevented from making serious errors. **For example,** users should not be permitted to enter alphabetic characters in numeric fields and inappropriate menu items should be declared obsolete.

   The interface must offer simple and constructive instructions in case it detects any error made by the user. **For example,** in case an incorrect zip code is entered by the user then he must be guided to repair the faulty part only rather than compelling him to enter the entire name-address form again. If there is any erroneous action then the systems state should be left unchanged or instructions should be offered by the interface to restore the previous state.

6) **Permit Easy Reversal of Actions:** The actions have to be reversible in nature as much as possible. This inspires the user to discover those options that are not acquainted to him as he knows that he can undo the errors. The units of reversibility vary from a single action to data entry task and group actions like entering a name-address block.

7) **Support Internal Locus of Control:** Users who have good amount of experience want to get a sense that they can control the interface and get response from the interface as per their action. If the data entry sequences are boring in nature, they get uninterested and don't want sudden changes or surprises but are more comfortable with familiar behaviour. They get anxious if the system is unable to produce the preferred outcome and face trouble in getting the essential information.

8) **Reduce Short-Term Memory Load:** As humans' have inadequate capacity to recall and memory is short-term in nature, the designer must take care at the time of designing the interface so that the user don't have to recall information from screen while navigating to another screen.

   For example, in a cell phone, websites should remain visible, phone numbers need not be entered again and again, adequate training must be provided in case the action series are difficult and multiple displays should be combined.

Interpretation and refinement of these underlying principles is necessary for every environment. Though they have limitations, mobile, web and desktop designers are highly benefited at the beginning. In order to gain control over the system, give rapid feedback to increase feeling of mastery and competence, simplify the data-entry procedures and make the displays comprehensible. The principles coming up in the following sections enhance users' productivity.

COMPUTER-AIDED SOFTWARE ENGINEERING (CASE)

Ques 22)What is Computer-Aided Software Engineering (CASE)? What are the main characteristics of CASE Tools?

Ans: Computer-Aided Software Engineering (CASE)
Computer Aided Software Engineering (CASE) tool refer to automated tools that are used for software development activities. CASE helps in development, verification, maintenance and generation of processes and artifacts. These tools have great importance when complexity of processes and artifacts become high.

A specific activity is supported by a tool. A collection of related tools forms an environment. CASE can be used to automate some functions of software engineering.

As market demands are changing at a much faster pace than before, development of new products needs to be faster so as to replace old products with new ones. So, in order to speed up the software system building process, CASE is introduced.

CASE tools (software programs) support human programmers with the difficulty of processes and artifacts of software engineering. They help in synthesis, analysis, modeling, documentation, coding, etc.

CASE may assist various processes:
1) Converting user requirements into software specifications.
2) Changing software specifications into design specifications.
3) Transformation of design into code.
4) Code testing for operational use.
5) Documentation.

Characteristics of CASE Tools
A CASE tool should have the following characteristics:
1) **Standard Methodology:** A CASE tool should support standard software development methodologies and modelling techniques. Presently, CASE tools use UML.
2) **Flexibility:** A CASE tool must provide flexibility and options to the user for editors and other tools.
3) **Strong Integration:** CASE tools must be integrated with all stages of software development. This means that if a change is made in a model, it must reflect in the code documentation and all related design. Hence, this offers an organised environment for software generation.
4) **Integration with Testing Software:** CASE tools should provide interfaces for automatic testing tools. This helps in regression and other testing software under changing conditions.
5) **Support for Reverse Engineering:** CASE tools should be as that it can create complex models from existing code.
6) **Online Help:** CASE tools offer online tutorials.

Ques 23)Discuss the building blocks of CASE.

Ans: CASE Building Blocks
Computer Aided Software Engineering (CASE) can imply a simple tool that supports a single activity of the software development process or it can provide tools that facilitate all the activities of the development process. It can be made up of hardware, standards, operating system, people, database and network. The base for the next block is formed whenever one block is built. To construct successful software engineering environments, both hardware and systems software are used to comprise the environment architecture.

Figure 6.14 demonstrates the several CASE building blocks that are used:

1) The **Environment architecture** is made up of hardware platform. System support amenities such as object-oriented management services, database management and networking software prepares the base for CASE.

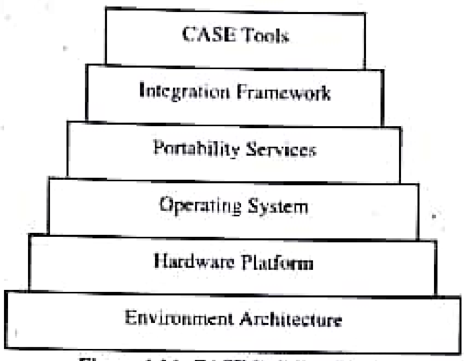


Figure 6.14: CASE Building Blocks

2) To create a bridge between the CASE tools, environment architecture and the framework where they will be integrated a set of portability services are used.

3) With the help of the integration framework which is made up of numerous specialised programs, the individual case can communicate. Similar look can be exhibited to the end user and a project database can be created.

4) The software engineering activities like analysis modelling and code generation, etc., can be greatly supported with the help of the CASE tools. The tools can be also used as point solutions, create project database or communicate with extra tools.

Ques 24)Discuss the taxonomy of CASE tools.

Ans: Taxonomy of CASE Tools
Table 6.1 depicts classification of CASE tools based on their working (function):

Table 6.1: Taxonomy of CASE Tools

Tools	Functions
Process Management Tools	Capture and model development processes.
Project Management Tools	Plan and schedule projects as well as monitor progress,
Risk Analysis Tools	Record, classify and evaluate risks.
Requirements Management Tools	Capture, evaluate and trace needs in the entire development process.
Metrics Tools	Capture particular software metrics and offer measures of quality.
Modelling Tools	Support modelling within analysis and design activities.
Programming Tools	Help in development of code.
Interface Design Tools	Support designing of Graphical User Interfaces (GUI).
Test Management Tools	Manage and organise testing.

Ques 25) Explain the architecture of CASE Environment?

Ans: Architecture of CASE Environment

The below mentioned architectural components are part of a CASE tool:

1) A database for storing information.
2) An object management system for managing variations made to the information.
3) A tools control mechanism for coordinating the use of the CASE tools.
4) A user interface for establishing a consistent pathway between the tools and user actions.

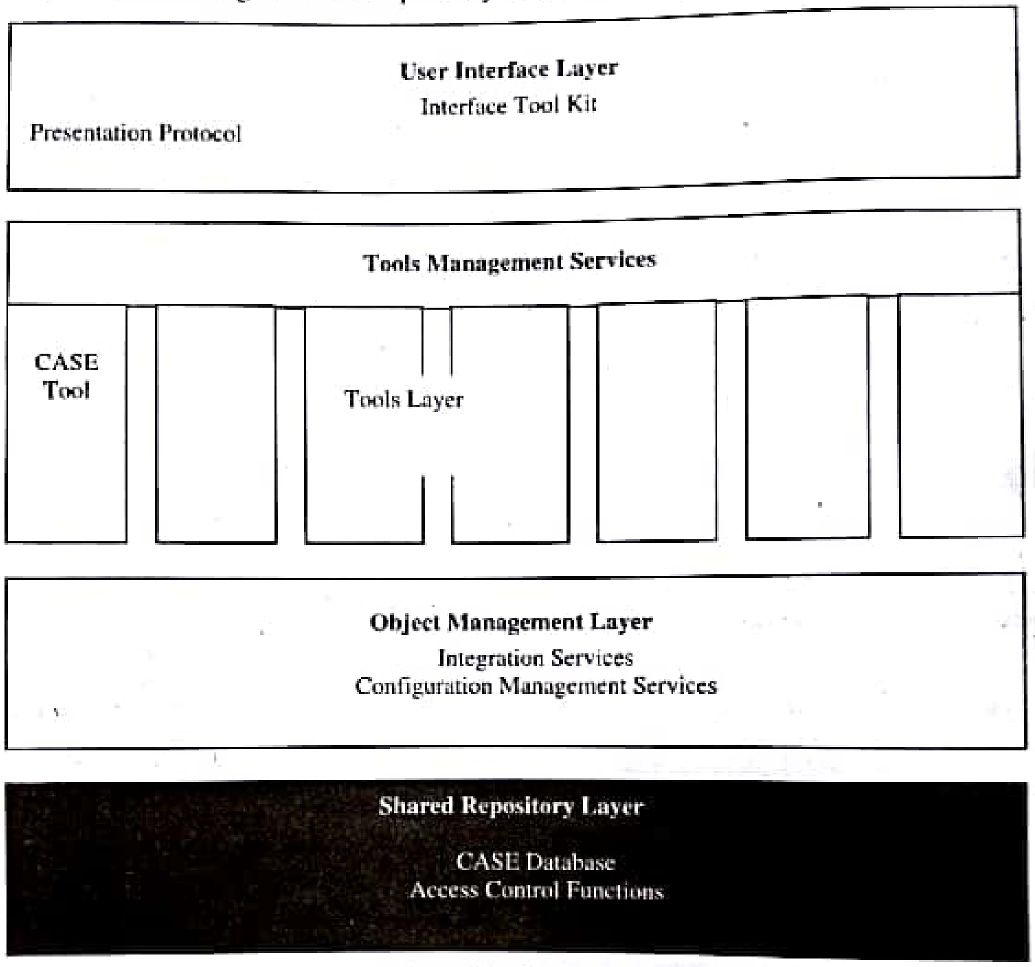


Figure 6.15: CASE Environment

Figure 6.15 shows the architecture:

1) **User Interface Layer:** It is made up of an interface toolkit that is standardised. It implements a common protocol which is used for presentation. The components of the interface are a library which contains display objects and software that facilitates management of interface between human and computer. With the help of these two components the individual CASE tools and components can communicate with each other consistently. The tool access mechanism, use of mouse and keyboard, menu names, object names, icons and screen layout are described in the presentation tool.

2) **Tools Layer:** Alongwith the CASE tools a set of services to manage the tools are also included. The behaviour of the tools within the environment is controlled by the Tools Management Services (TMS). TMS carries out multitask communication and synchronisation in case multitasking is performed by executing multiple tools at a time. This is done by gathering metrics on the usage of the tools, coordinating the information flow between the repository and management system to the tools. The auditing and security functions are also completed by it.

3) **Object Management Layer (OML):** The configuration management tasks are performed by it. A mechanism that enables in tool incorporation is the essence of the software that is present in this layer of the framework architecture. Each case tool has been plugged into the object management layer. The OML and the CASE

repository works in unison to provide integration services. The tools are coupled with the repository by this set of standard modules. Apart from all these functions, the OML also support change control, status accounting and audits. Configuration management services such as the task of classifying those configuration objects that perform version control are also executed by it.

4) **Shared Repository Layer:** It is made up of those access control functions with the help of which the object management layer interacts with the CASE database that is present in this layer. Shared repository layers and object management helps in attaining data integration.

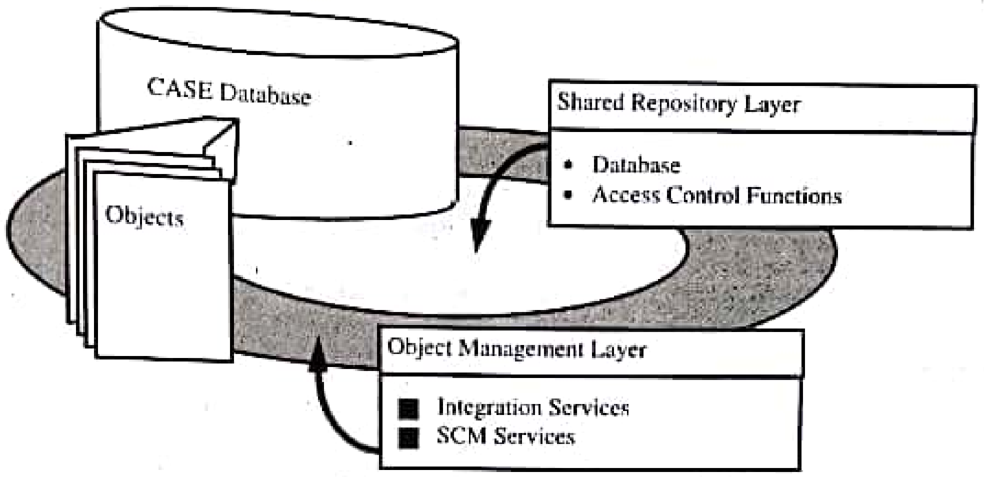


Figure 6.16: Case Repository

Ques 26)What are the advantages and disadvantages of CASE tools?

Ans: Advantages of CASE Tools
Though it is possible to achieve benefits if CASE tools are isolated, a great number of benefits can be obtained from the CASE environment. Following are the advantages of CASE tool:
1) The process system has an effective and longer operational life,
2) The produced system fulfils the user's requirements more efficiently,
3) The documentation of the produced system is tremendous,
4) Requirement of the support in the produced system is very low, and
5) More flexible systems are produced.

Disadvantages of CASE Tools
CASE has the following disadvantages:
1) The systems that are produced primarily require great cost to maintain and build,
2) It requires more perfect and extensive definition of the user's needs,
3) Customisation is difficult,
4) Trained maintenance staffs are needed, and
5) Difficulty is encountered while using with present systems.

Ques 27)Discuss about integrated CASE Environment.

Ans: Integrated CASE Environment
Although benefits can be derived from individual CASE tools that address separate software engineering activities, the real power of CASE can be achieved only through integration. The benefits of integrated CASE (I-CASE) include:
1) Smooth transfer of information (models, programs, documents, data) from one tool to another and one software engineering step to the next.
2) A reduction in the effort required to perform umbrella activities such as software configuration management, quality assurance, and document production.
3) An increase in project control that is achieved through better planning, monitoring, and communication, and
4) Improved coordination among staff members who are working on a large software project.

But I-CASE also poses significant challenges. Integration demands consistent representations of software engineering information, standardized interfaces between tools, a homogeneous mechanism for communication between the software engineer and each tool, and an effective approach that will enable I-CASE to move among various hardware platforms and operating systems. Comprehensive I-CASE environments have emerged more slowly than originally expected. However, integrated environments do exist and are becoming more powerful as the years pass.

I-CASE combines a variety of different tools and a spectrum of information in a way that enables closure of communication among tools, between people, and across the software process. Tools are integrated so that software engineering information is available to each tool that needs it; usage is integrated so that a common look and feel is provided for all tools; a development philosophy is integrated, implying a standardized software engineering approach that applies modern practice and proven methods.

To define integration in the context of the software engineering process, it is necessary to establish a set of requirements for I-CASE: An integrated CASE environment should:

1) Provide a mechanism for sharing software engineering information among all tools contained in the environment.
2) Enable a change to one item of information to be tracked to other related information items.
3) Provide version control and overall configuration management for all software engineering information.
4) Allow direct, non-sequential access to any tool contained in the environment.
5) Establish automated support for the software process model that has been chosen, integrating CASE tools and software configuration items (SCIs) into a standard work breakdown structure.
6) Enable the users of each tool to experience a consistent look and feel at the human/computer interface.
7) Support communication among software engineers.
8) Collect both management and technical metrics that can be used to improve the process and the product.

To achieve these requirements, each of the building blocks of a CASE architecture must fit together in a seamless fashion. The foundation building blocks – environment architecture, hardware platform, and operating system – must be "joined" through a set of portability services to an integration framework that achieves these requirements.