Table of Contents

Module No.	Chapter No.	Chapter Name	Page Nos.
I	1	Introduction to Software Engineering	1-1 to 1-33
II	2	Requirements Analysis and Cost Estimation	2-1 to 2-26
III	3	Design Engineering	3-1 to 3-29
IV	4	Software Risk, Configuration Management	4-1 to 4-18
V	5	Software Testing and Maintenance	5-1 to 5-40
VI	6	IT Project Management and Project Scheduling	6-1 to 6-79

□□□

MODULE I

CHAPTER 1 Introduction to Software Engineering

Syllabus

Nature of Software, Software Engineering, Software Process, Capability Maturity Model (CMM) Generic Process Model, Prescriptive Process Models: The Waterfall Model, V-model, Incremental Process Models, Evolutionary Process Models, Concurrent Models, Agile process, Agility Principles, Extreme Programming (XP), Scrum, Kanban model.

1.1 Software 1-3
1.2 Nature of Software 1-3
1.2.1 Defining Software 1-4
1.2.2 Software Application Domain 1-5
1.3 Software Engineering 1-6
1.3.1 Layered Approach 1-7
UQ. Explain Software Process Framework. MU - May 17, 5 Marks 1-7
1.3.2 Activities of Software Engineering 1-7
1.3.3 Need of Software Engineering 1-7
1.4 Generic View of Process 1-8
1.4.1 Comparison with Conventional Engineering Process 1-9
1.5 Capability Maturity Model (CMM) 1-10
UQ. Explain CMM Level 4 and 5. MU - Dec. 15, 5 Marks 1-10
UQ. What are the different levels of Capability Maturity Model (CMM) ? MU - May 16, Dec 17, May 18, 5 Marks 1-10
UQ. Explain CMM. MU - Dec. 16, Dec 17, 5 Marks 1-10
1.5.1 Capability Maturity Model Integration 1-11
UQ. Write note on Capability Maturity Model Integration (CMMI). MU - May 18, 5 Marks 1-11
1.6 Generic Process Model 1-12
1.7 Prescriptive Models 1-14
UQ. What is Prescriptive Model ? MU - Dec. 17, 5 Marks 1-14
1.7.1 Waterfall Model 1-14
UQ. Explain waterfall model give its advantage and disadvantages. MU - May 16, Dec. 17, 10 Marks 1-14
1.8 Incremental Process Models 1-15
UQ. Describe incremental model. MU - Dec. 17, 5 Marks 1-15
1.8.1 Difference between Waterfall Model and Incremental Model 1-16
UQ. Differentiate between waterfall model and incremental model. MU - Dec. 17, 5 Marks 1-16
1.8.2 RAD Model 1-16

1.8.2(A) Advantages of RAD Model 1-17
1.8.2(B) Disadvantages of RAD Model 1-17
1.9 Evolutionary Process Models 1-17
UQ. What is evolutionary model? List evolutionary models. MU - May 18, 5 Marks. 1-17
1.9.1 Spiral Model 1-18
UQ. Explain in detail Spiral model. (MU - May 15, Dec. 15, 5 Marks) 1-18
UQ. Explain any evolutionary model. MU -May 18, 5 Marks 1-18
1.9.2 Prototyping Model 1-18
1.9.2(A) Difference between Waterfall and Prototype Model 1-20
UQ. What is difference between Prescriptive and Evolutionary models. MU - May 15, May 18, 5 Marks 1-20
UQ. Give difference between waterfall and prototype model. MU - Dec.16, 5 Marks 1-20
1.10 Concurrent Development Model 1-21
1.11 Specialized Models : Component Based, Aspect Oriented Development 1-22
1.11.1 Component–Based Development 1-22
UQ. Write note on component based development MU - May 16, 10 Marks 1-22
UQ. Explain component based development. MU - Dec. 17, 5 Marks 1-22
1.11.2 The Formal Methods Model 1-22
UQ. Explain Formal Methods Model. MU - Dec. 17, 5 Marks 1-22
1.11.3 Aspect-Oriented Development 1-23
UQ. Write note on - Aspect Oriented Model. MU - May 15, 10 Marks 1-23
UQ. Explain Aspect Oriented Development. MU - May 17, 5 Marks 1-23
1.11.4 The Unified Process 1-23
.12 Agile Methodology 1-23
UQ. What is Agility ? MU - Dec. 16, Dec. 17, 5 Marks. 1-23
UQ. What is Agile Development ? MU - Dec. 17, May 18, 5 Marks 1-23
1.12.1 Agility Principles 1-24
UQ. Explain Agile principles. MU - May 17, Dec. 17, 5 Marks 1-24
1.12.2 Advantages of Agile Process 1-24
UQ. What are Advantages of Agile Processes? MU - Dec. 17, 5 Marks 1-24
1.12.3 Disadvantages of Agile Process 1-25
UQ. Is agile process suitable for large scale projects ? Mention few issues. MU - Dec. 15, 5 Marks 1-25
1.12.4 Difference between Prescriptive Process Model and Agile Process Model 1-25
13 Scrum 1-25
UQ. Explain SCRUM. MU - May 15, 5 Marks 1-25
14 Extreme Programming (XP) 1-27
UQ. Explain XP. MU - Dec. 16, Dec. 17, 5 Marks 1-27
UQ. Explain XP Agile Development Process. MU - May 17, May 18, 5 Marks 1-27
1.14.1 XP Values 1-28
5 Kanban Model 1-29
1.15.1 Kanban for Software Teams 1-30
1.15.2 Kanban Boards 1-30
1.15.3 Kanban Cards 1-30
1.15.4 Advantages of Kanban 1-30
• **Chapter Ends** 1-33

1.1 SOFTWARE

GQ. What is software? **(2 Marks)**

Definition

Computer **software**, or simply software, is a generic erm that refers to a collection of data or computer nstructions that tell the computer how to work, in contrast o the physical hardware from which the system is built, that ıctually performs the work.

- In computer science and software engineering, computer software is; all the information processed by computer systems, programs and data.
- Computer software includes computer programs, libraries and related non-executable data, such as online documentation or digital media.
- Computer hardware and software require each other and neither can be realistically used on its own.
- At the lowest level, executable code consists of machine language instructions specific to an individual processor-typically a Central Processing Unit (CPU).

A machine language consists of groups of binary values signifying processor instructions that change the state of the computer from its preceding state.

For example, an instruction may change the value stored in a particular storage location in the computer - an effect that is not directly observable to the user.

An instruction may also (indirectly) causes something to appear on a display of the computer system - a state change which should be visible to the user.

The processor carries out the instructions in the order they are provided, unless it is instructed to "jump" to a different instruction, or is interrupted by the operating system.

The majority of software is written in high-level programming languages that are easier and more efficient for programmers to use because they are closer than machine languages to natural languages.

High-level languages are translated into machine language using a compiler or an interpreter or combination of both.

Software may also be written in a low-level assembly language, which has strong correspondence to the computer's machine language instructions and is translated into machine language using an assembler.

- Fig. 1.1.1 shows how the user interacts with application software on a typical desktop computer. The application software layer interfaces with the operating system, which in turn communicates with the hardware. The arrows indicate information flow.

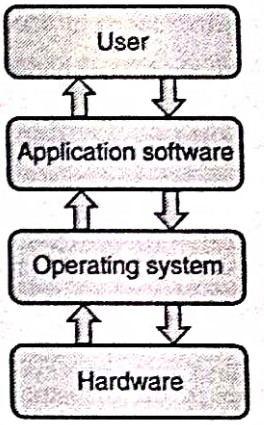


Fig. 1.1.1 : User interaction with application

1.2 NATURE OF SOFTWARE

GQ. Explain changing nature of software. **(5 Marks)**

- Now-a-days the software is not only remains a product, it also becomes an interface for delivering a product.
- As a product, software is able to deliver the computing potential which is basically embodied by computer hardware or precisely we can say, by a network of computers which one can access through local hardware.
- Software is considered as an information transformer or producer irrespective of whether it resides within a smart phone or functions inside a mainframe computer.
- Just like any transport vehicle, software can work as the basis for the control of the computer (operating systems), the transport of information (networks) and the generation and management of other programs (such as various software tools).
- Information delivery is the most important aspect of software. To make any data more useful in a local context, software can transform the data. It also helps in improving the competitiveness of a business by managing the business information.

- A gateway is provided by software to worldwide information networks (e.g., the Internet), and offer the way to gather information in several different forms.
- Over the last half-century, significant changes are occurred in the role of computer software.
- Now-a-days the software becomes more sophisticated as well as complex because of various aspects such as spectacular improvements in hardware performance, tremendous up-gradations in computing architectures, huge enhancements in memory and storage capacity, and extensive diversity of exotic input and output options.
- Sophistication as well as complexity leads to amazing results on success of systems, but they may also lead to huge problems for those who want to develop complex systems.
- In the economies of the industrialized world, the big software industry is considered as a dominant factor.
- In an earlier era, software was developed individually. Now-a-days an entire team works for it in which every person is responsible for one part of the technology which is necessary to deliver a complex application.

1.2.1 Defining Software

GQ. Define software. **(2 Marks)**

Software can be defined in different ways :

Definitions

1. **Software** is set of instructions (computer programs) that when executed provide desired features, function, and performance.
2. ***Software*** *is data structures that enable the programs to adequately manipulate information.*
3. ***Software*** *is descriptive information in both hardcopy and virtual forms that describes the operationand use of the programs.*

- *Now to understand the software exactly we have to examine some characteristics of it which make it different from remaining human made things.*
- *Software is considered as logical rather than a physical system element. Hence, there are characteristics of software which are significantly different than those of hardware.*

Characteristics of software

GQ. State any four attributes of good software. **(5 Marks)**

GQ. Describe the characteristics of software. **(5 Marks)**

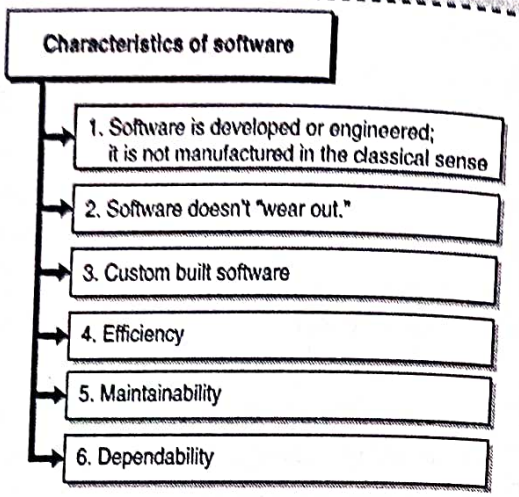


Fig. 1.2.1 : Characteristics of software

1. Software is developed or engineered; it is not manufactured in the classical sense

- Even though there are some similarities found in software development and hardware manufacturing, both the activities are considered as fundamentally different.
- In both of the activities, good design is base for high quality, but in the process of manufacturing there may be quality problems in hardware which may not present in case of software.
- In both of the activities there is prominent dependency on people but there is vast difference in the relationship between people and work accomplished.
- The main aim of both the activities is construction of a "product", but the perspectives are not same.
- Software costs are concentrated in engineering which indicates that it is difficult to manage software projects similar to manufacturing projects.

2. Software doesn't "wear out"

GQ. *Elaborate the software characteristic "Software does not wear out".* **(5 Marks)**

- *In case of hardware, the failure rates are high as compared to software early in its life. Such failures are mainly concerned with design or manufacturing defects. It is possible to correct the defects and drop the failure rate to a steady-state level for some time period.*

- However after some time period, there is again increase in the failure rate as there are cumulative effects of dust, vibration, mishandling, tremendous high or low temperature, and number of other environmental maladies on components of hardware. In simple words, the hardware begins to *wear out.*
- Software does not have any risks of such environmental maladies which lead to wear out of hardware.
- In early life of software there may be high failure rates because of undiscovered defects. However, these can be corrected.
- Now it is clear that there is no possibility of software wear out but it does deteriorate.
- In it life change is an integral part of software. Whenever any change is made, there is possibility of introduction of errors which increase the failure rate.
- Before the software gets consistent state by corrections, any new change get introduced which again increases the failure rate.
- Gradually, the lowest failure rate level starts to increase which indicates that the software is deteriorating due to change.
- There is one important difference in hardware and software regarding wear aspect is that in case of hardware, a failed component can be replaced by spare parts. This facility is not available in software as there are no such spare parts.
- All the software failures point out that there is an error in design or in the method by which the respective design was transformed into machine executable code.
- Hence the tasks of software maintenance which handles requests for change has significantly more complexity as compared to hardware maintenance.

3. Custom built software

- Component reusability is an important aspect in software industry.
- It is responsibility of software engineer to design and implement a software component in such a way that it should be reused easily in many different programs.
- Latest reusable components summarize both data as well as the processing which is applied to the data, which helps the software engineer to develop new applications from existing components.
- For example, reusable components are used in the development of modern interactive user interfaces that enable the generation of graphics windows, pull-down menus, as well as wide range of interaction mechanisms.

4. Efficiency

Software is said to be efficient if it uses the available resources in the most efficient manner and produce desired result in timely manner.

5. Maintainability

- If customer requirements changes, programmers need to modify the software to fulfill those requirements.
- Software engineering provides the ability to maintain the software through modifying the software rather than changing whole product like hardware.

6. Dependability

- Dependability is the ability of the software that provides services which can be trusted by users.
- Dependability provides services to fulfill the customer's requirements, so that it is helpful to achieve trust of customers on the system.

1.2.2 Software Application Domain

Virtually on all computer platforms, software can be grouped into few broad categories.

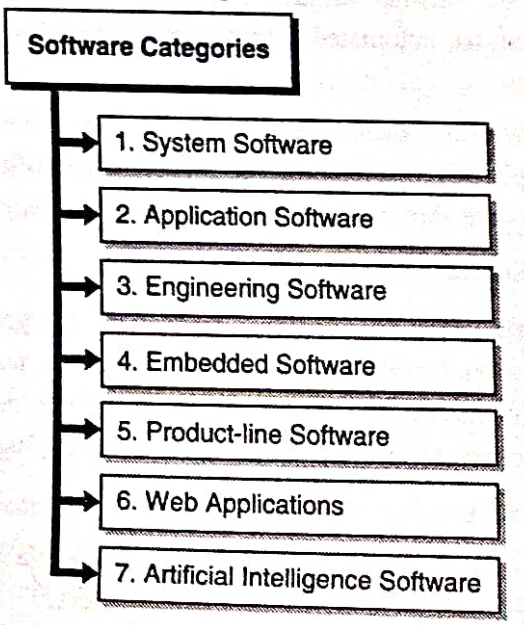


Fig. 1.2.2 : Software Categories

1. System software

- This is a set of programs used to provide service to other programs. Few of them such as compilers, editors, and file management utilities are used to process complex but determinate information structures.
- Some other system software such as components of OS, several drivers, networking related software, and telecommunications processors are generally used to process heavily indeterminate data.
- In all the situations, the system software area is broadly depends upon the interaction with computer hardware.

2. Application software

- These are the stand-alone programs which are specially developed for specific business needs.
- The Application Software helps to process business or technical data in such a way that it should be useful to manage business operations as well technical decision making.
- With traditional processing applications, one can also use application software to keep control on business functions in real time.

3. Engineering/scientific software

- These types of software are characterized through the "number crunching" algorithms. There are number of engineering software in various fields like astronomy, space shuttle orbital dynamics, automotive stress analysis, automated manufacturing, molecular biology etc.
- However, there are drastic changes in modern applications within the engineering/scientific area as they are shifting from traditional numerical algorithms.

4. Embedded software

These are the software which are merged within electronic devices and are used for the purpose of implementing and controlling the features as well as functions for the end user as well as system itself.

5. Product-line software

- *These software are developed to provide a particular functionality for use by several customers.*
- The focus of product-line software may be on a restricted marketplace like inventory control applications or can focus on mass consumer markets such as word processing, spreadsheets, graphics based applications, multimedia, database management, and personal as well as business financial products.

6. Web applications

- These applications are also called as "WebApps". This is a network-centric software category which covers large number of applications.
- In shortest form, WebApps are same as of the group of linked hypertext files which provide information through text and limited graphics.
- Now-a-days the modern WebApps are evolving into environments of sophisticated computing which offer stand-alone features, computing functions, as well as content to the end user.
- It is also possible to integrate WebApps with corporate databases and business applications.

7. Artificial intelligence software

- In these applications non-numerical algorithms are generally used for the purpose of solving complex problems which cannot be handled by computation or straightforward analysis.
- There are various applications in this category such as robotics, games, expert systems, artificial neural networks, proving theorem, pattern matching like image and voice.

1.3 SOFTWARE ENGINEERING

GQ. Explain the term 'Software Engineering'. **(5 Marks)**

Software Engineering combines two concepts, Software and Engineering.

Software

- *As we have already seen Software is set of executed programs related to specific functionality. Set of executable instructions is called as a program.*
- *Software is more than just a program code; it also includes data structures which allow programs to manipulate information, and documentation related to software product which defines the functionality and*

guidance for the user to use this software. Software engineers design and build the software product.

Engineering

Engineering is an application of knowledge and well defined principles to develop products.

Software Engineering

❒ **Definition 1**

Software Engineering is the method of applying scientific and technological knowledge, procedures, and rules to design, develop, and maintain the software product.

OR

❒ **Definition 2**

Applying technological, scientific, and administrative approach to designing, developing, testing and maintaining the software product in order to meet customers' requirements with best quality of product referred as **software engineering**.

1.3.1 Layered Approach

UQ. Explain Software Process Framework. **MU - May 17, 5 Marks**

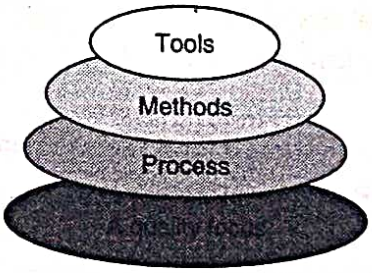


Fig. 1.3.1 : Layered Approach of software Technology

This approach is divided into 4 layers :

1. **A quality Process**

- Any engineering approach must rest on the quality.
- The most important aspect in software Engineering is Quality Focus.

2. **Process**

- Foundation for SE is the Process Layer.
- SE process is the GLUE that holds all the technology layers together and enables the timely development of computer software.
- It forms the base for management control of software project.

3. **Methods**

- SE methods provide the "Technical Questions" for building Software.
- Methods contain a broad array of tasks that include communication requirement analysis, design modeling, program construction testing and support.

4. **Tools**

- SE tools provide automated or semi-automated support for the "Process" and the "Methods".
- Tools are integrated so that information created by one tool can be used by another

1.3.2 Activities of Software Engineering

GQ. Enlist Activities of Software Engineering. **(2 Marks)**

Software Engineering includes many activities as follows :

- Understanding the user requirements is the first most important activity in software engineering.
- According to the requirements, the process of designing the system and taking decisions is implemented, which further leads to successful completion of development of product.
- Constructing the software product based upon designs and decisions made earlier is the next activity in software engineering.
- To verify the performance and quality of software, testing the software product is essential after it is constructed by software engineers.
- The next step is to provide maintenance for software product which is deployed to the customer.

1.3.3 Need of Software Engineering

GQ. Explain need of Software Engineering. **(2 Marks)**

- As the technology changes, the user requirements and environment on which software is working also changes. So every organization is ranked based on the software engineering principles used by that organization.

- Implementing and managing large size of software, programmer requires a specific method to modularize the tasks so that size of software can't harm the software quality. Software engineering provides methodology for implementing complex software systems with high quality.
- Without any standard method or management, it is difficult to address defects in the product and correct them as early as possible. Software Engineering provides this functionality.
- Extending the previous software to add new functionality requires more cost in terms of time to develop and efforts taken by people, as compare to the process of developing new software to provide that functionality. Software engineering provides a way in which software system can be able to scale as needed in future.

Syllabus Topic : Generic View of Process

1.4 GENERIC VIEW OF PROCESS

GQ. Explain in brief Software Process Framework. (4 Marks)

- A software engineering process is the model selected for managing the creation of software from customer initiation to the release of the finished product. The selected process typically involves methods such as
- The software life cycle follows the sequential flow in order to develop software.
- We will see each and every stage of this life cycle in detail in subsequent points :

(i) Requirement gathering and analysis

- In this phase stakeholders communicate with customer and system users to gather the business requirements like who will use the system? How user will interact with the system? What should be the system input and output? etc.

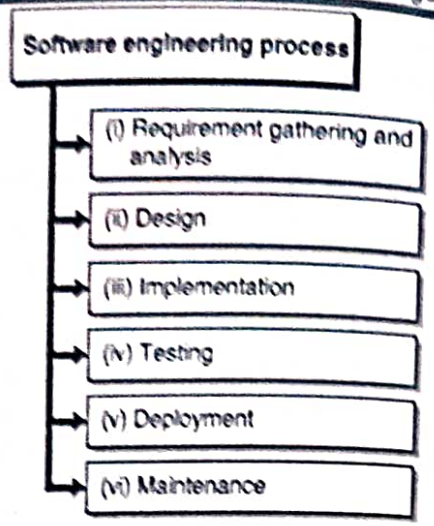


Fig. 1.4.1 : Software engineering process

- Depending on these requirements, Requirement Specification Document (SRS) is prepared.

(ii) Design

- This phase makes use of output of requirement gathering phase i.e. requirement specification document as an input.
- Based on requirement specifications software design is prepared.
- Output of this phase is design specification which specifies hardware and software requirements of system.
- Data Flow Diagram, flowcharts etc. are included by design specification document.
- Design specification acts as an input for implementation phase.

(iii) Implementation phase

- Implementation phase of development process decomposes the system work into various smaller parts called **modules**.
- These modules are assigned to development team members and actual coding is started.
- This is longest phase of system development.
- Output of this phase is actual code using specific programming language.

(iv) Testing

- Testing phase involves testing of actual code of system against requirements of user in order to ensure that system will satisfy all needs of users.

Various testing strategies used are unit testing, system testing, integration testing etc.

Output of testing phase is corrected and modified software code.

) **Deployment**

In this phase the system is deployed at users' site for their use.

In this phase customer uses the software and give feedback to development team for any changes or modifications in system if any.

i) **Maintenance**

Once the software is deployed actual problems from user's perspective will come up. It is responsibility of development team to solve user's problems time to time. This process is called as maintenance of system.

In this phase some additional functionality may need to add in the application as per user's requirement.

1.4.1 Comparison with Conventional Engineering Process

GQ. Compare Software engineering with Conventional Engineering Process. (5 Marks)

- Software engineering has some similar things as the principles of conventional engineering.
- It is assumed that software engineering is associated to the design of software or data processing products and belongs to its problem solving domain, including the class of problems related to software and data processing.
- This idea is expanded by drawing similarity from the methods that are generally used in engineering.
- It is observed that, just as the famous scientific method is used in the field of scientific research, the steps of engineering design process are used in the process of problem solving in the field of engineering.
- Following steps are mostly repetitive :
  1. Problem formulation,
  2. Problem analysis,
  3. Search for alternatives,
  4. Decision,
  5. Specification,
  6. Implementation
- It is advised that these steps are applicable to the field of software engineering as well.
- Software engineering is considered as a result of integration of hardware and systems engineering, including a set of 3 key elements - methods, tools and procedures which facilitate the manager to control the process of software development.
- The methods give the technical "how to's " for building software; tools offer automated or semi-automated support for methods; and procedures describe the series of applying methods, the deliverables, the controls, and the objectives.
- Compared to traditional engineering disciplines, software engineering demonstrates some remarkable differences.
- In conventional engineering, one moves from an abstract design to a concrete product. On other hand, in software engineering, one moves from design to coding.

Software Engineering	Abstract Design	→	More Abstract Code
Manufacturing Engineering	Abstract Design	→	Concrete Products

- The problem domains of software engineering can be anything, from word processing to real-time control and from games to robotics.
- In contrast to another engineering discipline, it is therefore much larger in scope and thus provides bigger challenges.
- Traditional manufacturing engineering that normally emphasize mass production is loaded with production features. Thus, it is highly production concentrated. Software engineering, in contrast, is inherently design concentrated.
- Product standardization assists in cost reduction in manufacturing, whereas such a possibility is remote in software engineering.
- The possibility of process standardization, however, is very high in the latter.
- Conventional engineering process makes the use of unlimited number of domain and application-specific

ideas which are proved. Software engineering, in contrast, makes use of limited, but global concepts such as loops, conditions etc.

1.5 CAPABILITY MATURITY MODEL (CMM)

UQ. Explain CMM Level 4 and 5. **MU - Dec. 15, 5 Marks**

UQ. What are the different levels of Capability Maturity Model (CMM) ? **MU - May 16, Dec 17, May 18, 5 Marks**

UQ. Explain CMM. **MU - Dec. 16, Dec 17, 5 Marks**

- The Capability Maturity Model (CMM) invented by Software Engineering Institute (SEI) states a growing sequence of levels of a software development organization.
- The higher the level, the better the software development procedure so reaching every level is a cost consuming and time consuming procedure.
- The Capability Maturity Model (CMM) is a mechanism used to create and refine software creation procedure of organization.
- The CMM is same as ISO 9001, one of the ISO 9000 sequences of standards determined by the International Organization for Standardization (ISO).
- The ISO 9000 standards state an efficient quality system for manufacturing and service industries; ISO 9001 applied particularly to software development and maintenance.
- The major variation among the two systems present in their uses : ISO 9001 states a lowest acceptable quality level for software procedures, while the CMM use a framework for uninterrupted procedure improvement and is more explicit than the ISO standard in defining use of employed to that end.

Following are levels of CMM :

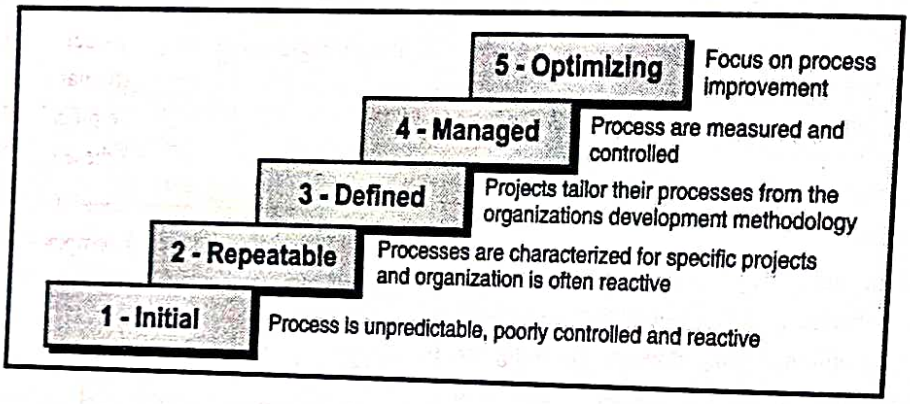


Fig. 1.5.1 : CMM Levels

1. Level 1 : Initial

- This software procedure is stated as inconsistent and occasionally chaotic.
- Stated procedures and standard practices that present are discarded at the time of crisis.
- Success of the association mainly depends on efforts taken by each person, capacity, and heroics.
- The heroes move on to another association taking their knowledge or lessons learnt with them.

2. Level 2 : Repeatable

- This level of Software Development association has a fundamental and consistent project management processes to keep the track of cost, schedule, and functionality.
- The procedure is in region to repeat the latest successes on projects with similar applications.
- Program management is a most important feature of a level two association.

3. Level 3 : Defined

The software procedure for management and engineering actions are documented, standardized, and

incorporated into a standard software procedure for the overall association and each and every project developed by association use an approved, tailored version of the standard software procedure of association for creating, testing and maintaining the software product.

4. Level 4 : Managed

- Management can successfully control the efforts required for software development through precise measurements.
- At this level, association set a quantitative quality aim for both software procedure and software maintenance.
- At this level, the performance of procedures is controlled through statistical and other quantitative mechanisms, and is quantitatively predictable.

5. Level 5 : Optimizing

- The important characteristic of this level is concentrating on repeatedly improving procedure performance with the help of both increasing and innovative technological corrections.
- At this level, modifications to the procedures are done to increase the procedure performance and at the same time maintaining statistical probability to get the recognized quantitative procedure-improvement aims.

1.5.1 Capability Maturity Model Integration

UQ. Write note on Capability Maturity Model Integration (CMMI). **MU - May 18, 5 Marks**

- The CMM model's application in software development has sometimes been problematic. Applying multiple models that are not integrated within and across an organization could be costly in training, appraisals, and improvement activities.
- The Capability Maturity Model Integration (CMMI) project was formed to sort out the problem of using multiple models for software development processes, thus the CMMI model has superseded the CMM model, though the CMM model continues to be a general theoretical process capability model used in the public domain.
- CMMI framework consists of a collection of computer programs based on knowledge, system engineering, software engineering, integrated product and process development and provider sourcing.
- CMMI framework has three groups as :
  1. CMMI for Development (CMMI - DEV)
  2. CMMI for Service (CMMI - SVC)
  3. CMMI for Acquisition (CMMI - ACQ)
- These three groups forms model components which are uniquely designed for particular business process and they may contain some core processes as a part of them which will be same among these groups.
- It is also possible to extend CMMI framework by making addition of model component to it.

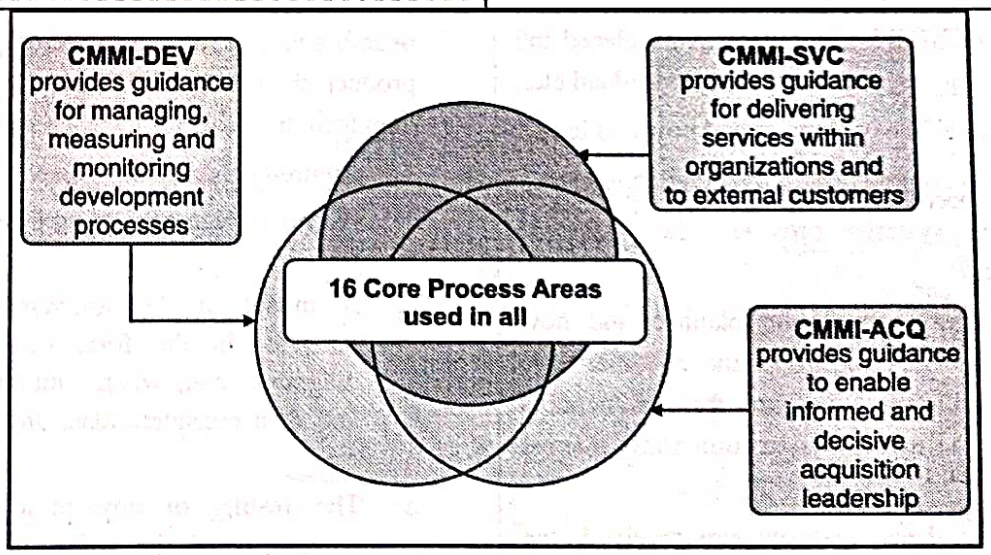


Fig. 1.5.2 : CMM Framework Groups

1.6 GENERIC PROCESS MODEL

GQ. Write note on Generic Process Model. (10 Marks)

- In section 1.4 we have already seen the important phases of a software process which can also be termed as Software Development Life Cycle.
- Now we are going to discuss all these phases in detail for a generic process model with some important aspects.

System Engineering

- As software approximately always makes the modules of a larger system, one can starts work by forming requirements for all components of the system, recognizing not only the role performed by the software but, more importantly, its interface and interaction with the outside world.
- This 'system view' is necessary when software must interface with other elements such as hardware, people, databases and computers outside of, and ahead of the control of the system designer.
- In essence Systems engineering contains discovering some of the following issues :
  - (i) Where does the software solution fit into the overall picture?
  - (ii) What does the system do? Is it necessary to examine some process or activity or is it simply performing analysis of data?
  - (iii) What environment will the system be placed in? That is system is on-line, real-time, embedded etc.
  - (iv) What user interface is required and how is it used?
  - (v) How systems communicate with other computers?
  - (vi) Does the system provide the required performance?
  - (vii) What time period has been planned and how serious it is. What budget does the customer have and is it realistic? What are the cost/benefits tradeoffs to the user in automating some manual procedure?
- *When answers of these questions are received, the developer is in a good position to evaluate the RISK involved in implementing the system.*

(1) Requirements Gathering and Analysis

- Requirements Analysis is the first important step towards creating a specification and a design.
- Trying to design a solution to a problem without totally understanding the nature and requirements of the user will always fails down the solution.
- It has been shown that more than ninety percent of software that are rejected by the customers after delivery is because the software does not meet the customer needs, expectation or requirements so it is important to understand those requirements completely.
- Requirements analysis is an iterative process carried out together by an analyst and the customer and represents an attempt, to discover the customer's needs, even as at the same time assessing the viability of a software solution.
- Analysis gives the designer the demonstration of the information which is processed by the system and the nature of the processing. i.e., what does the system do with the information which it processes?
- Analysis enables the designer to identify the software's function and performance. Also where the customer is not sure about how the system will eventually behave; the analyst may discover the concept of a prototype.

(2) Analysis Summary

- The aim of requirements analysis is to generate a "requirements specification document" or a "target document" that illustrates great extent detail in an unambiguous a manner regarding exactly what the product should do. This requirements document will then form the basis for the succeeding design phase.
- The requirements document may consists of :
  - A Project plan containing details of delivery times and cost estimates.
  - A model of the software's functionality and behavior in the form of 'data flow / UML diagrams' and, where suitable, any performance and data considerations, any input/output details etc.
  - The results of any prototyping so that the emergence of the software and how it should behave can be exposed to the designer. This might contain a user's manual.

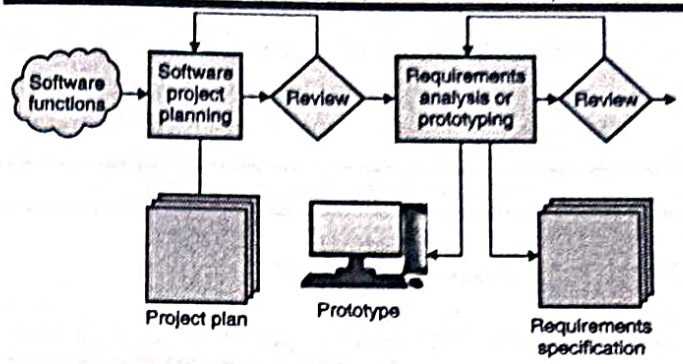


Fig. 1.6.1 : Activities in Analysis

- Any formal Z or VDM specifications for the system requirements.
- Any test data that may be used to find out whether the end product is build according to the agreed specification or not.

- The Fig. 1.6.1 illustrates the activities that are sum-up by the analysis.

(3) Design and Implementation (Coding)

- When the analysis of the system has been completed, design or development can start. This is an attempt to convert a group of requirements and program/data models that are specified in the "requirements document" into a well-organized, designed and engineering software solution.
- Design can be best summarized with the help of following steps :
  - The data flow / UML (Unified modeling Language) diagrams that signify the system model are transformed into appropriate hierarchical, modular program and data structure/architecture.
  - Every program module is transformed into a correct cohesive function subroutine or class that is designed to do an individual task which is well-defined.
  - Design further focus on the implementation of individual module/class. The user interfaces of module and its communication with other modules/objects is also considered and evaluated for good design.
  - The algorithm of module can afterwards converted into a flowchart, which is a step-by-step graphical representation of the actions which are held by the module demonstrated in terms of sequence, selection and repetition.
  - The flowchart can then be converted into Pseudo code, which conveys the same information as a flowchart, but presents it in a way that is more acceptable to translate into program code.
  - At the end, the Pseudo code for every module is converted into a selected programming language and the number of modules entered are compiled and integrated into a system which is ready for testing.
  - The final result of design and coding contains the architecture, data structures, flowcharts, Pseudo code, algorithms and all program source code listings, as depicted in Fig. 1.6.2.

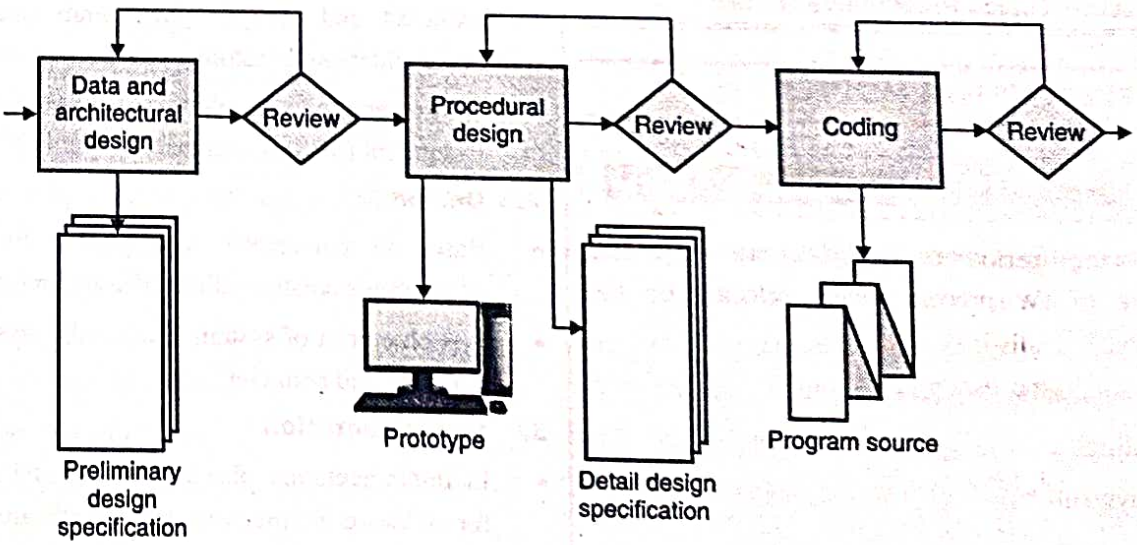


Fig. 1.6.2 : Design and Implementation (Coding)

(4) Software Testing

- When the part of the software components/modules has been written, testing and debugging can be started.
- Following techniques are involved in testing :
  - (i) **Verification and Validation :** This technique is useful for checking that the software meets the agreed specification and verifying that the software is correct in its operation.
  - (ii) **Black and white box testing techniques :** It is helpful for testing the insides of the modules for correct operation and testing the interfaces of the module.
  - (iii) **Integration Testing :** Testing that the modules all working together properly.
  - (iv) **Acceptance Testing :** Allowing the customer to test the product.
- Debugging can be defined as it is an art of recognizing the cause of failure in a part of software and correcting it.

(5) Deployment

After completion of software development, we have to install it at client site. This process is known as deployment.

(6) Software Maintenance

Software maintenance reapplies all of the previous life cycle steps to an existing program instead of a new one in order to correct existing or insert new functionality.

Syllabus Topic : Prescriptive Models

1.7 PRESCRIPTIVE MODELS

UQ. What is Prescriptive Model ? **MU - Dec. 17, 5 Marks**

- The following framework activities are performed irrespective of the process model selected by the organization.
  1. Communication
  2. Planning
  3. Modelings
  4. Construction
  5. Deployment
- The name **'prescriptive'** is given because the model prescribes a set of activities, actions, tasks, quality assurance and change the mechanism for every project.

Syllabus Topic : Waterfall Model

1.7.1 Waterfall Model

UQ. Explain waterfall model give its advantage and disadvantages. **MU - May 16, Dec. 17, 10 Marks**

- Waterfall model is the first approach used in software development process.
- It is also called as classical life cycle model or linear sequential model.
- In waterfall model any phase of development process begins only if previous phase is completed.

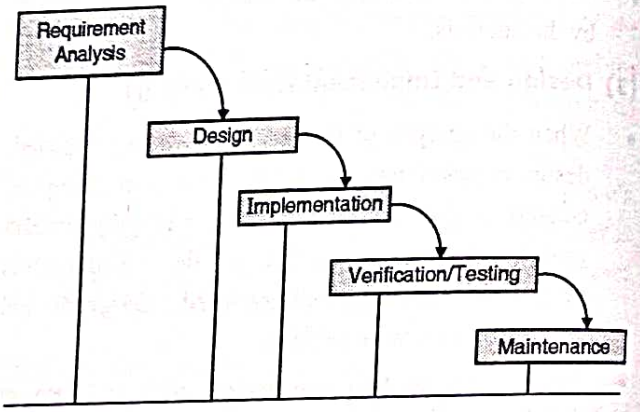


Fig. 1.7.1 : Waterfall Model

1. Requirement Analysis

- In this phase all business requirements of system are gathered and analyzed by communication between stakeholders and customer or user.
- At the end of this phase Requirement Specification Document (SRS) is created.

2. Design

- Based on requirement, specification document design of system is created called software architecture.
- It is blue print of system representing system's internal structure and behavior.

3. Implementation

- In implementation phase actual coding is constructed for software architecture using hardware and software requirements of system.

- It is responsibility of developer.

4. Verification / Testing

- Here coding or job done by developer is verified against requirements of user in order to ensure that software will satisfy all business requirements of user.
- After the successful verification software is deployed at user's site for their use.

5. Maintenance

- While using software if user faces some problems, then those problems must be solved time to time by development team. This task comes under maintenance of software.
- Maintenance also includes adding new functionalities in software as per user requirements.

☞ Advantages of waterfall model

(i) It is very simple to understand and easy to use.

(ii) Phases of waterfall model do not overlap with each other.

(iii) It is useful for small projects in which requirements are clear initially.

(iv) Since development is linear it is easy to manage development process.

☞ Disadvantages of waterfall model

(i) It is not useful for large projects.

(ii) Not suitable for projects in which requirements are not clear initially.

(iii) System or product is available only at the end of development process.

(iv) It is very difficult to modify system requirements in the middle of development process.

Practical situations in which Waterfall model can be used

- This model is used only when the requirements are very well known, clear and fixed.
- Product definition is stable.
- Technology is understood.
- There are no ambiguous requirements.
- Ample resources with required expertise are available freely.
- The project is short.

1.8 INCREMENTAL PROCESS MODELS

UQ. Describe incremental model.
MU - Dec. 17, 5 Marks

- The incremental model applies the waterfall model incrementally.
- The series of releases is referred to as "increments", with each increment providing more functionality to the customers.
- After the first increment, a core product is delivered, which can already be used by the customer.
- Based on customer feedback, a plan is developed for the next increments, and modifications are made accordingly.
- This process continues with increments being delivered until the complete product is delivered. The incremental philosophy is also used in the agile process model.

Following tasks are common to all the models :

1. **Communication :** Helps to understand the objective.
2. **Planning :** Required as many people (software teams) work on the same project but different functions at same time.
3. **Modelling :** Involves business modelling, data modelling, and process modelling.
4. **Construction :** This involves the reuse of software components and automatic code.
5. **Deployment :** Integration of all the increments.

☞ Characteristics of Incremental Model

1. System is broken down into many mini development projects.
2. Partial systems are built to produce the final system.
3. First tackled highest priority requirements.
4. The requirement of a portion is frozen once the incremented portion is developed

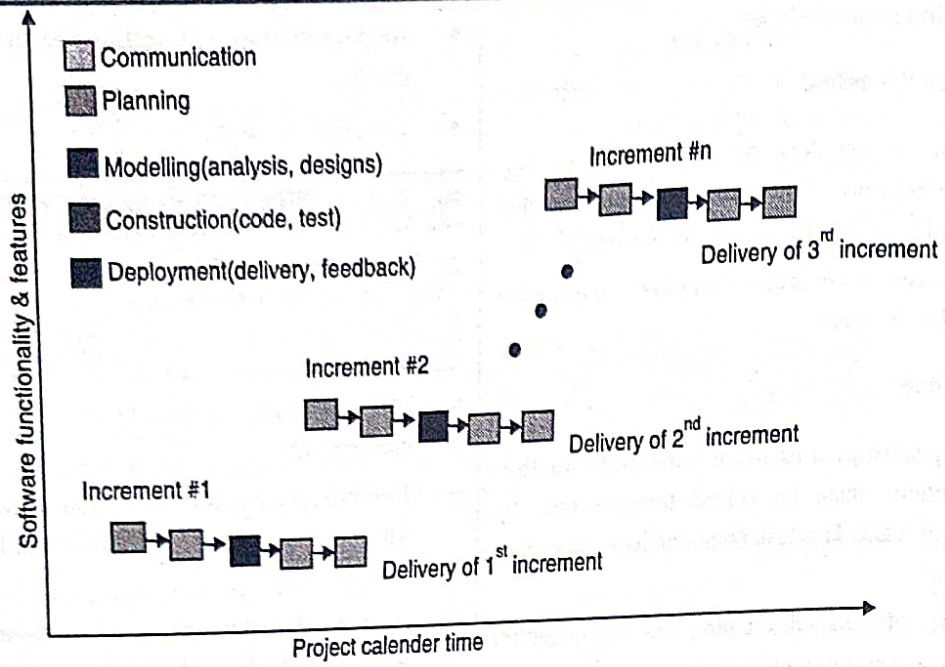


Fig. 1.8.1 : Incremental Model

Advantages of Incremental Model

1. After each iteration, regression testing should be conducted. During this testing, faulty elements of the software can be quickly identified because few changes are made within any single iteration.
2. It is generally easier to test and debug than other methods of software development because relatively smaller changes are made during each iteration. This allows for more targeted and rigorous testing of each element within the overall product.
3. Customer can respond to features and review the product for any needed or useful changes.
4. Initial product delivery is faster and costs less.

Disadvantages of Incremental Model

1. Resulting cost may exceed the cost of the organization.
2. As additional functionality is added to the product, problems may arise related to system architecture which were not evident in earlier prototypes.

1.8.1 Difference between Waterfall Model and Incremental Model

UQ. Differentiate between waterfall model and incremental model. **MU - Dec. 17, 5 Marks**

Parameter	Waterfall Model	Incremental Model
Simplicity	Simple	Intermediate
Risk Involvement	High	Easily manageable
Flexibility to change	Difficult	Easy
User involvement	Only at beginning	Intermediate
Flexibility	Rigid	Less Flexible
Maintenance	Least	Promotes maintainability
Duration	Long	Very Long

Syllabus Topic : RAD Model

1.8.2 RAD Model

- Rapid Application Development is an adaption to linear sequential model with a high speed, following incremental development approach.
- In this model speedy (rapid) development of product is achieved by using iteration based construction.

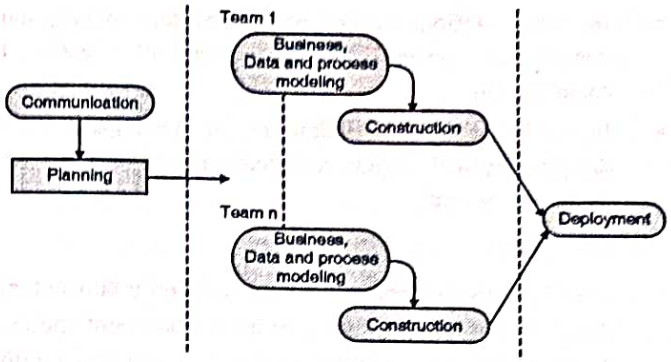


Fig. 1.8.2 : RAD Model

Phases of RAD model are as follow

(i) Communication

Communication between different stakeholders and customer's takes place to identity business requirement and necessary characteristics that the system must satisfy.

(ii) Planning

Planning an important activity of RAD model as various development teams works of many different modules of system which needs to be integrated at the end of development process.

(iii) Modeling

- Modeling phase of RAD includes three major activities. Business modeling, data modeling and process modeling.
- It involves creation of software architecture which acts as a basis for RAD's construction activity.

(iv) Construction

- Construction phase involves constructing actual code for software architecture.
- It makes use of various pre-existing software components and apply various automatic code generation tools.

(v) Deployment

- In this phase all the constructions made by various development teams are integrated and deployed at user's site for their use.
- It is always followed by maintenance of system time to time.

1.8.2(A) Advantages of RAD Model

(i) Faster development as compared to other development models.

(ii) Use of automatic code generation tools results in reduction of manual coding.

(iii) Incremental approach gives better quality product.

(iv) RAD is most suitable for small scale project development.

1.8.2(B) Disadvantages of RAD Model

(i) Not suitable for large projects.

(ii) Cannot be used when technical risks are high.

(iii) Project will fail if customer and developer not agreed upon rapid fire activities.

(iv) Use of automatic code generation tools may increases the cost of project.

Syllabus Topic : Evolutionary Process Models

1.9 EVOLUTIONARY PROCESS MODELS

UQ. What is evolutionary model? List evolutionary models. **MU - May 18, 5 Marks**

- Normally the Evolutionary Development Model is based on the principle that "stages consist of growing increments of an operational software product, with the way of evolution being discovered by operational experience".
- According to the business need and the changing nature of the market there are lot of improvements required in the software product over a time.
- Due to this lot of improvement is required in the product hence evolutionary developments model is iterative in nature.
- There are two types of models in Evolutionary process. They are :
  1. Spiral model
  2. Prototyping model

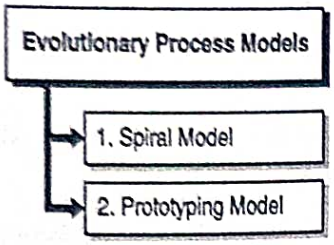


Fig. 1.9.1 : Evolutionary Process Models

Syllabus Topic : Spiral Model

1.9.1 Spiral Model

UQ. Explain in detail Spiral model.
MU - May 15, Dec. 15, 5 Marks

UQ. Explain any evolutionary model.
MU -May 18, 5 Marks

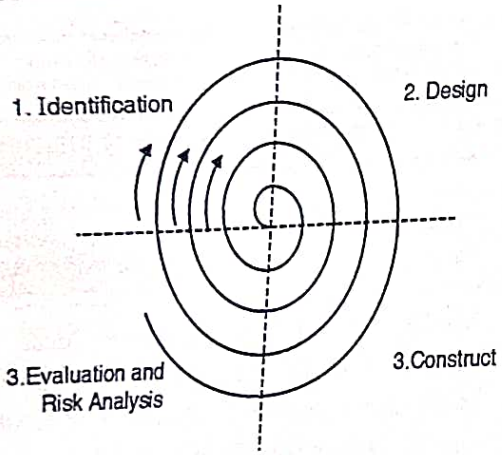


Fig. 1.9.2 : Spiral Model

- Spiral model is a combination of iterative model and waterfall model.
- Spiral model has four phases of development each of these phases is called as spiral.

(i) Identification

- *This phase identifies all business requirements of system at the beginning. It involves clear understanding of requirements by communication between stakeholders and customer.*
- *In the subsequent iterations all subsystem requirements and unit requirements are identified.*

(ii) Design

- In first iteration, design phase develops co design of system based on initially requirements.
- In further spirals or iterations, it develops design, physical design, architectural design ar design of system.

(iii) Construct

Initially construct phase develops a code for con design to get user feedback. In next subsequent detailed working model of software is constructe increment number and are delivered to custom feedback.

(iv) Evaluation and risk analysis

- In this phase management risks like cost overru identified and monitored, technical feasibility of s is also done.
- At end of each spiral customer evaluates softwa potential risks in system and provides feedback.
- Spiral model can be used for projects where budge risk evaluation are important factors.
- If customers are not sure about their requirements spiral model is useful than waterfall model.

Advantages of spiral model

(i) It is more flexible to changing requirements.
(ii) Requirements are achieved more accurately.
(iii) User can see the system from 1st iteration to en development.
(iv) Risk management is easier.

Disadvantages of spiral model

(i) It is difficult to manage development process.
(ii) Not useful for small projects development.
(iii) Spiral can run indefinitely.
(iv) It requires excessive documentation work documentation is prepared for each iteration.

Syllabus Topic : Prototyping Model

1.9.2 Prototyping Model

GQ. Describe Prototype model. **(5 Marks**

- Prototyping model refers to developing software application prototypes (early approximation / version) which displays the behaviour of product under development but may not actually contain the exact logic of the original application.
- Prototyping allows user to evaluate developers' proposal and try it before actual implementation.
- Prototyping model is widely used popular software development model as it helps to understand user requirements in early stage of development process.

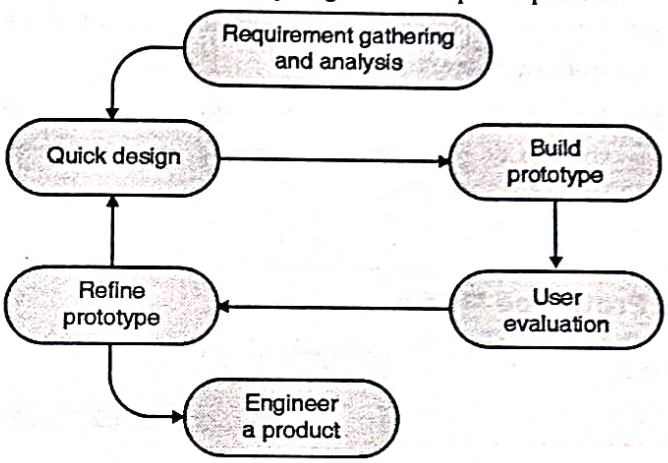


Fig. 1.9.3 : Prototyping Model

Phases of prototyping model are as follow :

(i) Requirement gathering and analysis

- Building of prototype begins with requirement gathering and analysis.
- In this phase various users of system are interviewed in order to know their system requirements or expectations from system.
- Requirement specification report is generated as an output of this phase.

(ii) Quick Design

- After gathering and analysis of user requirements quick design (prototype design) of system is developed.
- Quick design is not detailed design of system; it contains only necessary characteristics of the system which gives basis idea of system to the end users.

(iii) Build prototype

- Based on feedback received from user about quick design of system, the first prototype of system is created.
- Prototype is working model of system under development.

(iv) Use Evaluation

- Once prototype is built, proposed system is presented to end user of system for its evaluation.
- User determines strengths and weaknesses of prototype i.e. what needs to be added or what is to be removed.
- All the comments and suggestions given by user in feedback are passed to developer.

(v) Refining prototype

- Depending on comments and suggestions came from user, developers refine previous prototype to form new prototype of system.
- New prototype is again evaluated just like previous prototype.
- The process of evaluation and refining prototype continues until all user requirements are met by prototype.

(vi) Engineer a product

- Once evaluation and refining of prototype completes i.e. when user accepts final prototype of system the final system is evaluated thoroughly and deployed at user's site.
- Deployment of engineered product is further followed by regular maintenance of system.

When to use Prototype model

- Prototype model should be used when the desired system needs to have a lot of interaction with the end users.
- Typically online systems in which web interfaces have a very high amount of interaction with end users, are best suited for Prototype model.
- It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.
- Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system.

- They are excellent for designing good human computer interface systems.

Advantages of prototyping model

(i) It enables early evaluation of system by providing working model to end users at early stage of development.

(ii) Refining of prototype results in better implementation of system requirements.

(iii) Communication between developer and user reduces ambiguity.

(iv) More involvement of user in development process results in meeting user's requirement at greater extent.

Disadvantages of prototyping model

GQ. What are the potential problems of Prototyping Model ? **(5 Marks)**

(i) Refining of prototype continues until user is completely satisfied, thus it is time consuming and expensive process.

(ii) More involvement of user in development process is not accepted by developer always.

(iii) Refining of prototype again and again may disturb the working of development team.

(iv) Practically prototyping model results in increasing the complexity of system as scope of system extends beyond original plan.

1.9.2(A) Difference between Waterfall and Prototype Model

UQ. What is difference between Prescriptive and Evolutionary models. **MU - May 15, May 18, 5 Marks**

UQ. Give difference between waterfall and prototype model. **MU - Dec.16, 5 Marks**

Parameter	Waterfall Model / Prescriptive Model	Prototype Model / Evolutionary Model
Client preview	Client can preview the system only after the final version of the software is developed because there is no feedback loop.	Client has a preview of the system from the "quick design" and the prototype is developed early of process.
Developer rights	Developers encounter a freezing requirement where they are not allowed to modify the requirements or specification of the previous phase until the next iteration.	Developers can refine or add requirements and specification to the system after the prototype is built.
Complexity of error	The complexity of an error increases because of the nature of the model; each phase is sequential of the other.	The complexity of an error is low because the prototype enables the developer to detect any deficiency early at the process.
Linearity	Linear	Non-linear
Suitable for	The waterfall model is better suited for more conventional software projects where user requirements are clear, right from the start.	Prototype model is well suited for online applications where user interfaces are the most important components and clients are not clear about what they exactly need in the final product.

Syllabus Topic : Concurrent Development Model

1.10 CONCURRENT DEVELOPMENT MODEL

GQ. Write note on Concurrent Models. (10 Marks)

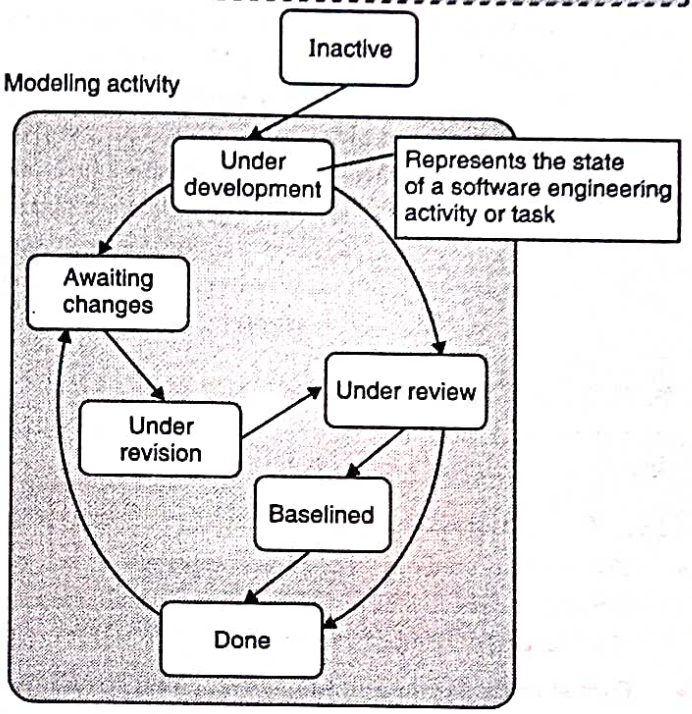


Fig. 1.10.1 : Concurrent Models

- The concurrent development model is also known as concurrent engineering.
- This model helps software team in the representation of iterative as well as concurrent elements of various process models.
- For example, consider the modeling activity which has been defined for the spiral model is carried out by calling one or more software engineering actions such as prototyping, analysis, and design.
- In Fig. 1.10.1 we can observe schematic representation regarding a software engineering activity in the scope of the modeling activity with the help of concurrent modeling approach.
- The state of activity **modeling** may be any out of the states which are noted at any particular time.
- In the same way, it is possible to represent remaining activities, actions, or tasks such as **communication, construction** etc. in an analogous manner.
- All the activities regarding software engineering are executed simultaneously but exist in different states.
- For example, consider in the early phase of a project, first iteration of the communication activity (not displayed in the diagram) has been completed and exists in the **awaiting changes** state.
- The modeling activity whose state is **inactive** at the time of completion of initial communication, now changes its state to **under development.**
- However, if the customer enforces that changes in requirements should be necessarily done, the modeling activity shifts from the **under development** state into the **awaiting changes** state.
- Concurrent modeling describes a sequence of events which will activate transitions in between states for all the software engineering activities, actions, or tasks.
- For example, in the previous phase of design (an important action regarding software engineering which occurs in the process of modeling activity), an inconsistency in the requirements model is uncovered.
- Because of it the event *analysis model correction, is* generated that will activate the requirements analysis action from the done state into the **awaiting changes** state.
- Concurrent modeling is proved to be compatible for all the types of software development and offers an exact picture of the ongoing state of a project.
- Instead of encircling all the software engineering actions, and tasks to a series of events, a process network is defined by the concurrent modeling.
- All the activities, actions, or tasks on the network execute concurrently with other activities, actions, or tasks.
- In the process network, events which are generated at one point activate transitions between the states.

Syllabus Topic : Specialized Models : Component Based, Aspect Oriented Development

1.11 SPECIALIZED MODELS : COMPONENT BASED, ASPECT ORIENTED DEVELOPMENT

Special process models take on many of the characteristics of one or more of the conventional models. However, specialized models tend to be applied when a narrowly defined software engineering approach is chosen.

1.11.1 Component-Based Development

UQ. Write note on component based development.
MU - May 16, 10 Marks

UQ. Explain component based development.
MU - Dec. 17, 5 Marks

- Commercial off-the-shelf (COST) software components, developed by vendors who offer them as products, can be used when software is to be built.
- These components provide targeted functionality with well-defined interfaces that enable the component to be integrated into the software.
- The component-based development model incorporates many of the characteristics of the spiral model. It is evolutionary in nature, demanding an iterative approach to the creation of software.
- However, the model composes applications from prepackaged software components. Modeling and construction activities begin with the identification of candidate components.
- These components can be designed as either conventional software modules or object-oriented classes or packages of classes.
- Regardless of the technology that is used to create the components, the component-based development model incorporates following steps (implemented using an evolutionary approach) :
  - Available component-based products are researched and evaluated for the application domain in question.
  - Component integration issues are considered.
  - Software architecture is designed to accommodate the components.
  - Components are integrated into the architecture.
  - Comprehensive testing is conducted to ensure proper functionality. The component-based development model leads to software reuse, and reusability provides software engineers with a number of measurable benefits. Based on studies of reusability, component-based development can lead to reduction in development cycle time, reduction in project cost and increase in productivity. Although these results are a function of the robustness of the component library, there is little question that whether the component-based development model provides significant advantages for software engineers.

1.11.2 The Formal Methods Model

UQ. Explain Formal Methods Model.
MU - Dec. 17, 5 Marks

- The formal methods model encompasses a set of activities that leads to formal mathematical specification of computer software.
- Formal methods enable a software engineer to specify, develop, and verify a computer-based system by applying rigorous, mathematical notation.
- A variation on this approach is called clean-room software engineering.
- When formal methods are used during development, they provide a mechanism for eliminating many of the problems that are difficult to overcome using other software engineering paradigms.
- Ambiguity, incompleteness, and inconsistency can be discovered and corrected more easily, not through ad hoc review, but through the application of mathematical analysis. When formal methods are used during design, they serve as a basis for program verification and therefore enable the software engineer to discover and correct errors that might otherwise go undetected.
- Although not a mainstream approach, the formal methods model offers the promise of defect-free

software. Yet, concern about its applicability in a business environment has been expressed :

- The development of formal models is currently quite time-consuming and expensive.
- Because only few software developers have the necessary background to apply formal methods, extensive training is required.
- It is difficult to use the model as a communication mechanism for technically unsophisticated customers.

1.11.3 Aspect-Oriented Development

GQ. Write note on - Aspect Oriented Model.
MU - May 15, 10 Marks

GQ. Explain Aspect Oriented Development.
MU - May 17, 5 Marks

- Regardless of the software process that is chosen, the builders of complex software invariably implement a set of localized features, functions and information content.
- These localized software characteristics are modeled as components and then constructed within the context of a system architecture.
- As modern computer-based systems become more sophisticated and complex, certain concerns, customer required properties or areas of technical interest, span the entire architecture.
- Some concerns are high-level properties of a system; others affect functions or are systemic. When concerns cut across multiple system functions, features, and information they are often referred to as crosscutting concerns.
- Aspectual requirements define those crosscutting concerns that have impact across the software architecture.
- Aspects are mechanisms beyond subroutines and inheritance for localizing the expression of a crosscutting concerns.
- Aspect-Oriented Software Development (AOSD), often referred to as Aspect-Oriented Programming (AOP), is a relatively new software engineering paradigm that provides a process and methodological approach for defining, specifying, designing, and constructing aspects.
- A distinct aspect-oriented process has not yet matured. However, it is likely that such a process will adopt characteristics of both the spiral and concurrent process models.
- The evolutionary nature of the spiral is appropriate as aspects are identified and then constructed. The parallel nature of concurrent development is essential because aspects are engineered independently of localized software components and yet aspects have a direct impact on these components.

1.11.4 The Unified Process

- In some ways the unified process (UP) is an attempt to draw on the best features and characteristics of conventional software process models, but characterize them in a way that implements many of the best principles of agile software development.
- The unified process recognizes the importance of customer communication and streamlined methods for describing the customer's view of a system.
- It emphasizes the important role of software architecture and helps the architect focus on the right goals, such as understandability, reliance to future changes, and reuse.
- It suggests a process flow that is iterative and incremental, providing the evolutionary feel that is essential in modern software development.

Syllabus Topic : Agile Methodology

1.12 AGILE METHODOLOGY

UQ. What is Agility ? **MU - Dec. 16, Dec. 17, 5 Marks**

UQ. What is Agile Development ?
MU - Dec. 17, May 18, 5 Marks

- Agile software development describes an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customers (end users).

- It advocates adaptive planning, evolutionary development, early delivery, and continual improvement, and it encourages rapid and flexible response to change.
- The term Agile was popularized, in this context, by the Manifesto for Agile Software Development.
- The values and principles adopted in this manifesto were derived from and underpin a broad range of software development frameworks, including Scrum and Kanban.
- There is significant subjective evidence that adopting agile practices and values improves the agility of software professionals, teams and organizations.

Agile software development values

- Based on their combined experience of developing software and helping others do that, the seventeen signatories to the manifesto proclaimed that they value:
  - Individuals and Interactions over processes and tools.
  - Working Software over comprehensive documentation.
  - Customer Collaboration over contract negotiation.
  - Responding to Change over following a plan.
- As per the view of Scott Ambler (Canadian software engineer, consultant and author) :
  - Tools and processes are important, but it is more important to have competent people working together effectively.
  - Good documentation is useful in helping people to understand how the software is built and how to use it, but the main point of development is to create software, not documentation.
  - A contract is important but is no substitute for working closely with customers to discover what they need.
  - A project plan is important, but it must not be too rigid to accommodate changes in technology or the environment, stakeholders' priorities, and people's understanding of the problem and its solution.

1.12.1 Agility Principles

UQ. Explain Agile principles.

MU - May 17, Dec. 17, 5 Marks

The Manifesto for Agile Software Development is based on twelve principles :

1. Customer satisfaction by early and continuous delivery of valuable software.
2. Welcome changing requirements, even in late development.
3. Working software is delivered frequently (weeks rather than months).
4. Close, daily cooperation between business people and developers.
5. Projects are built around motivated individuals, who should be trusted.
6. Face-to-face conversation is the best form of communication (co-location).
7. Working software is the primary measure of progress.
8. Sustainable development, able to maintain a constant pace.
9. Continuous attention to technical excellence and good design.
10. Simplicity is essential.
11. Best architectures, requirements, and designs emerge from self-organizing teams.
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly.

1.12.2 Advantages of Agile Process

UQ. What are Advantages of Agile Processes?

MU - Dec. 17, 5 Marks

There are number of advantages of Agile Process :

- Stakeholder Engagement,
- Transparency,
- Early and Predictable Delivery,
- Predictable Costs and Schedule,
- Allows for Change,
- Focuses on Business Value,
- Focuses on Users,

- Improves Quality.

1.12.3 Disadvantages of Agile Process

UQ. Is agile process suitable for large scale projects ? Mention few issues. **MU - Dec. 15, 5 Marks**

Agile methodology is not considered as suitable for large projects because of following important issues :

- Lack of clarity
- Continual reliance on legacy methods
- Inadequate experience with agile
- Lack of collaboration in teams composed by different companies
- Lack of a Testing Strategy
- Lack of alignment in other areas of the enterprise
- Larger teams and big pyramid structures
- Not changing the objectives

1.12.4 Difference between Prescriptive Process Model and Agile Process Model

GQ. Differentiate between prescriptive process model and agile process model (any four points). **(5 Marks)**

Parameter	Prescriptive Process Model	Agile Process Model
Basic aim	Developed to bring order and structure to the software development process.	These models satisfy customer through early and continuous delivery.
Functionality	It can accommodate changing requirements	Defines a distinct set of activities, actions, tasks, milestones, and work products that are required to engineer high-quality software.
Popularity	It is more popular.	Comparatively less popular.
Examples	Water fall model, Incremental models	Scrum, extreme Programming (XP), Feature Driven Development (FDD), Dynamic Systems Development Method (DSDM), Adaptive Software Development (ASD).

Syllabus Topic : Scrum

1.13 SCRUM

UQ. Explain SCRUM. **MU - May 15, 5 Marks**

- Scrum is an agile framework for managing knowledge work, with an emphasis on software development. It is designed for teams of three to nine members, who break their work into actions that can be completed within time boxed iterations, called "sprints", no longer than one month and most commonly two weeks, then track progress and re-plan in 15-minute stand-up meetings, called daily scrums.
- Scrum principles are mostly based on the agile manifesto and help to guide activities regarding development inside a process which includes framework activities such as requirements, analysis, design, evolution, and delivery.
- In all the framework activities, work tasks happen within a process pattern called a sprint.

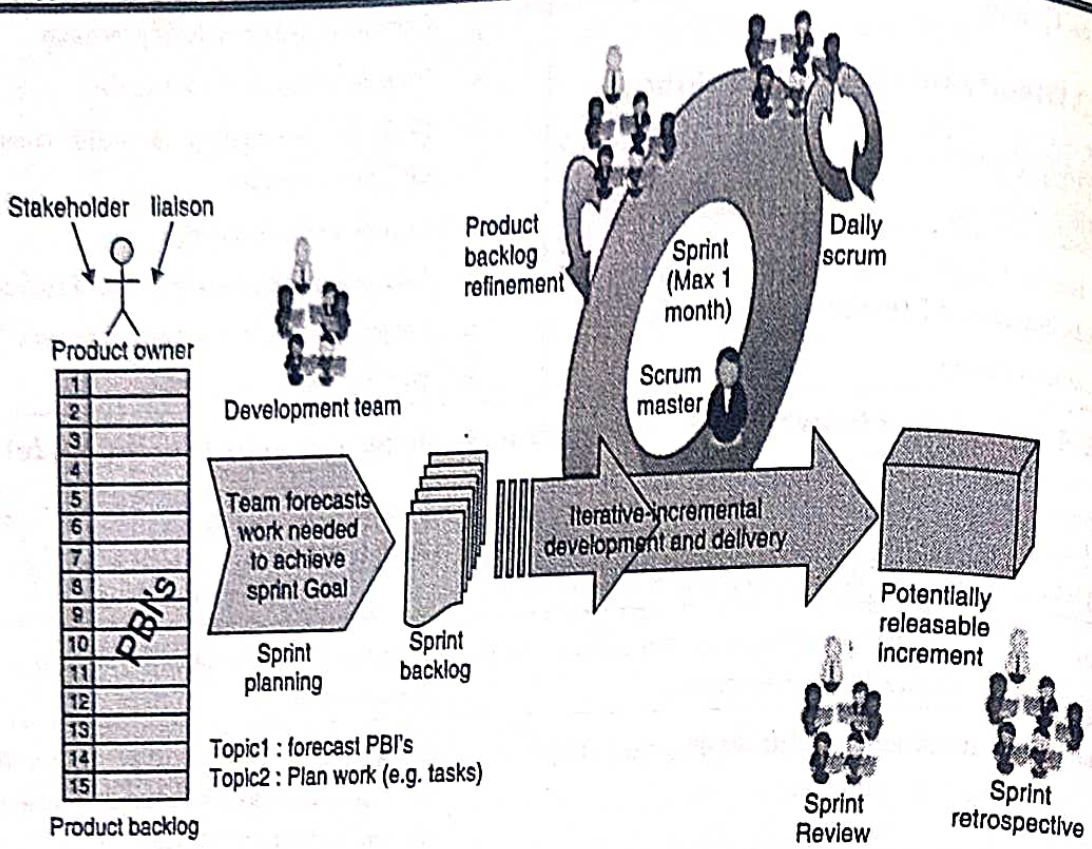


Fig. 1.13.1 : Scrum Framework

- The work implemented in a sprint is tailored to the problem at hand and is defined and usually tailored in real time by the respective Scrum team.

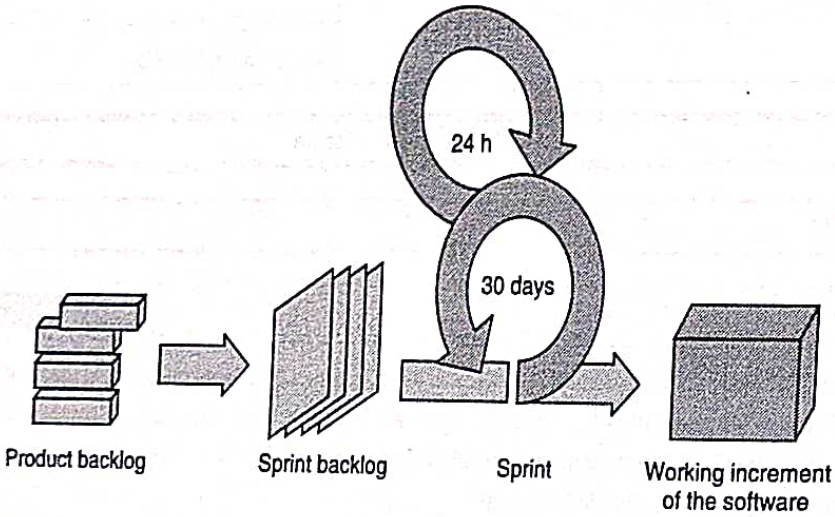


Fig. 1.13.2 : The scrum process

- Scrum mostly focus on the use of a bunch of software process patterns which are proved effective for projects with restrictive timelines, changing requirements, and business criticality. All such process patterns defines several sets of development actions :
- **Backlog :** A prioritized list regarding project needs or features which offer business value for the customer. It is possible to add items to the backlog at any time. The respective product manager assesses the backlog and changes the priorities as per necessity.
- **Sprints :** Includes work units which are necessary to gain a requirement defined in the backlog which

should be incorporated in predefined time-box (usually thirty days).

- Modifications (such as backlog work items) are not involved during the sprint. Therefore, team members are allowed by the sprint to work in a short-term, but stable environment.
- Scrum meetings - are short (usually fifteen minutes) meetings conducted on daily basis by the Scrum team. Three most important key questions which are asked and must be answered by all team members are :
  1. What did you do since the last team meeting ?
  2. What obstacles are you encountering ?
  3. What do you plan to accomplish by the next team meeting ?
- A team leader, who is known as a Scrum master, handles the meeting and evaluates the responses from each person.
- The important use of Scrum meeting to the team is that it can uncover critical issues as early as possible.
- One more benefit of these daily meetings is that they lead to "knowledge socialization" and thus encourage a self-organizing team structure.
- **Demos :** Implement the software increment at the client side (customer) so that the newly developed functionality can be demonstrated as well as evaluated by the customer.
- It is also possible that the demo may not able to contain all planned functionality, but instead such functions which can be delivered within the established time-box.

Syllabus Topic : Extreme Programming

1.14 EXTREME PROGRAMMING (XP)

UQ. Explain XP. **MU - Dec. 16, Dec. 17, 5 Marks**

UQ. Explain XP Agile Development Process. **MU - May 17, May 18, 5 Marks**

- Extreme Programming (XP) is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements.
- As a type of agile software development, it advocates frequent "releases" in short development cycles, which is intended to improve productivity and introduce checkpoints at which new customer requirements can be adopted.

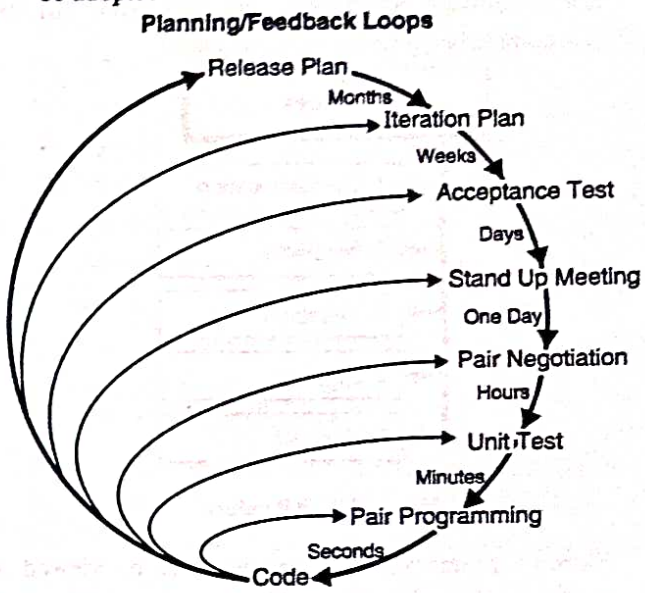


Fig. 1.14.1 : Extreme Programming

- Other elements of extreme programming include: programming in pairs or doing extensive code review, unit testing of all code, avoiding programming of features until they are actually needed, a flat management structure, code simplicity and clarity, expecting changes in the customer's requirements as time passes and the problem is better understood, and frequent communication with the customer and among programmers.
- The methodology takes its name from the idea that the beneficial elements of traditional software engineering practices are taken to "extreme" levels.
- As an example, code reviews are considered a beneficial practice; taken to the extreme, code can be reviewed continuously, i.e. the practice of pair programming.

1.14.1 XP Values

Extreme programming initially recognized four values in 1999 : communication, simplicity, feedback, and courage. A new value, respect, was added in the second edition of Extreme Programming Explained. Those five values are described in Fig. 1.14.2.

1. Communication

- Building software systems requires communicating system requirements to the developers of the system. In formal software development methodologies, this task is accomplished through documentation.

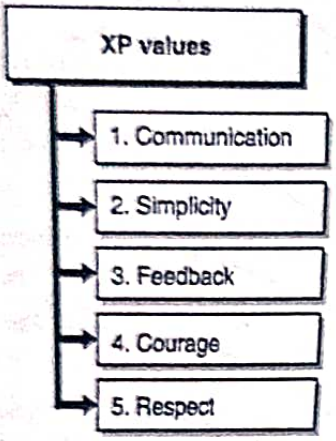


Fig. 1.14.2 : XP values

- Extreme programming techniques can be viewed as methods for rapidly building and disseminating institutional knowledge among members of a development team.
- The goal is to give all developers a shared view of the system which matches the view held by the users of the system.
- To this end, extreme programming favors simple designs, common metaphors, collaboration of users and programmers, frequent verbal communication, and feedback.

2. *Simplicity*

- *Extreme programming encourages starting with the simplest solution. Extra functionality can then be added later.*
- *The difference between this approach and more conventional system development methods is the focus on designing and coding for the needs of today instead of those of tomorrow, next week, or next month.*
- This is sometimes summed up as the "You aren't gonna need it" (YAGNI) approach.
- Proponents of XP acknowledge the disadvantage that this can sometimes entail more effort tomorrow to change the system; their claim is that this is more than compensated for by the advantage of not investing in possible future requirements that might change before they become relevant.
- Coding and designing for uncertain future requirements implies the risk of spending resources on something that might not be needed, while perhaps delaying crucial features.
- Related to the "communication" value, simplicity in design and coding should improve the quality of communication. A simple design with very simple code could be easily understood by most programmers in the team.

3. Feedback

- Within extreme programming, feedback relates to different dimensions of the system development :
  - **Feedback from the system :** By writing unit tests, or running periodic integration tests, the programmers have direct feedback from the state of the system after implementing changes.
  - **Feedback from the customer :** The functional tests (aka acceptance tests) are written by the customer and the testers. They will get concrete feedback about the current state of their system. This review is planned once in every two or three weeks so the customer can easily steer the development.
  - **Feedback from the team :** When customers come up with new requirements in the planning game the team directly gives an estimation of the time that it will take to implement.
- Feedback is closely related to communication and simplicity. Flaws in the system are easily communicated by writing a unit test that proves a certain piece of code will break.
- The direct feedback from the system tells programmers to recode this part.
- A customer is able to test the system periodically according to the functional requirements, known as user stories.

- As per Kent Beck (American software engineer and the creator of extreme programming), "Optimism is an occupational hazard of programming. Feedback is the treatment".

4. Courage

- Several practices represent courage. One is the commandment to always design and code for today and not for tomorrow.
- This is an effort to avoid getting delayed in design and requiring a lot of effort to implement anything else. Courage enables developers to feel comfortable with refactoring their code when necessary.
- This means reviewing the existing system and modifying it so that future changes can be implemented more easily.
- Another example of courage is knowing when to throw code away : courage to remove source code that is obsolete, no matter how much effort was used to create that source code.
- Also, courage means persistence : a programmer might be stuck on a complex problem for an entire day, then solve the problem quickly the next day, but only if they are persistent.

5. Respect

- The respect value includes respect for others as well as self-respect.
- Programmers should never commit changes that break compilation, that make existing unit-tests fail, or that otherwise delay the work of their peers.
- Members respect their own work by always striving for high quality and seeking for the best design for the solution at hand through refactoring.
- Adopting the four earlier values leads to respect gained from others in the team.
- Nobody on the team should feel unappreciated or ignored. This ensures a high level of motivation and encourages loyalty toward the team and toward the goal of the project.
- This value is dependent upon the other values, and is oriented toward teamwork.

1.15 KANBAN MODEL

GQ. Explain the Kanban Model in detail. (5 Marks)

- Kanban is a popular framework used to implement agile software development. It requires real-time communication of capacity and full transparency of work.
- Work items are represented visually on a kanban board, allowing team members to see the state of every piece of work at any time.
- Kanban is enormously prominent among today's agile software teams, but the kanban methodology of work dates back more than 50 years.
- In the late 1940s Toyota began optimizing its engineering processes based on the same model that supermarkets were using to stock their shelves.
- Supermarkets stock just enough products to meet consumer demand, a practice that optimizes the flow between the supermarket and the consumer.
- Because inventory levels match consumption patterns, the supermarket gains significant efficiency in inventory management by decreasing the amount of excess stock it must hold at any given time. Meanwhile, the supermarket can still ensure that the given product a consumer needs is always in stock.
- When Toyota applied this same system to its factory floors, the goal was to better align their massive inventory levels with the actual consumption of materials.
- To communicate capacity levels in real-time on the factory floor (and to suppliers), workers would pass a card, or "kanban", between teams.
- When a bin of materials being used on the production line was emptied, a kanban was passed to the warehouse describing what material was needed, the exact amount of this material, and so on.
- The warehouse would have a new bin of this material waiting, which they would then send to the factory floor, and in turn send their own kanban to the supplier.
- The supplier would also have a bin of this particular material waiting, which it would ship to the warehouse. While the signaling technology of this process has

evolved since the 1940s, this same "just in time" (or JIT) manufacturing process is still at the heart of it.

1.15.1 Kanban for Software Teams

- Agile software development teams today are able to leverage these same JIT principles by matching the amount of work in progress (WIP) to the team's capacity.
- This gives teams more flexible planning options, faster output, clearer focus, and transparency throughout the development cycle.
- While the core principles of the framework are timeless and applicable to almost any industry, software development teams have found particular success with the agile practice.
- In part, this is because software teams can begin practicing with little to no overhead once they understand the basic principles.
- Unlike implementing kanban on a factory floor, which would involve changes to physical processes and the addition of substantial materials, the only physical things a software teams need are a board and cards, and even those can be virtual.

1.15.2 Kanban Boards

- The work of all kanban teams revolves around a kanban board, a tool used to visualize work and optimize the flow of the work among the team.
- While physical boards are popular among some teams, virtual boards are a crucial feature in any agile software development tool for their traceability, easier collaboration, and accessibility from multiple locations.
- Regardless of whether a team's board is physical or digital, their function is to ensure the team's work is visualized, their workflow is standardized, and all blockers and dependencies are immediately identified and resolved.
- A basic kanban board has a three-step workflow: To Do, In Progress, and Done. However, depending on a team's size, structure, and objectives, the workflow can be mapped to meet the unique process of any particular team.
- The kanban methodology relies upon full transparency of work and real-time communication of capacity, therefore the kanban board should be seen as the single source of truth for the team's work.

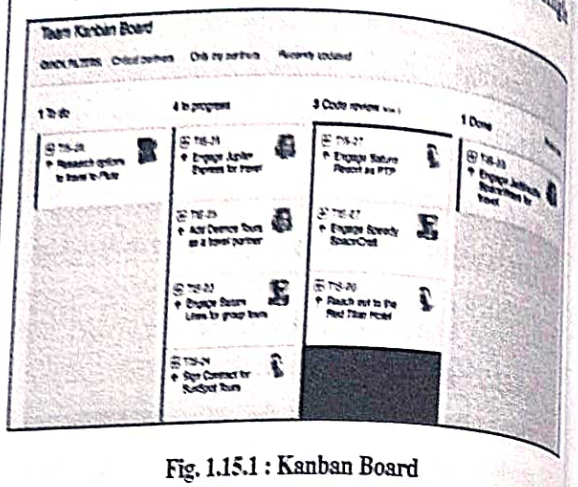


Fig. 1.15.1 : Kanban Board

1.15.3 Kanban Cards

- In Japanese, kanban literally translates to "visual signal." For kanban teams, every work item is represented as a separate card on the board.
- The main purpose of representing work as a card on the kanban board is to allow team members to track the progress of work through its workflow in a highly visual manner.
- Kanban cards feature critical information about that particular work item, giving the entire team full visibility into who is responsible for that item of work, a brief description of the job being done, how long that piece of work is estimated to take, and so on.
- Cards on virtual kanban boards will often also feature screenshots and other technical details that are valuable to the assignee.
- Allowing team members to see the state of every work item at any given point in time, as well as all of the associated details, ensures increased focus, full traceability, and fast identification of blockers and dependencies.

1.15.4 Advantages of Kanban

- Kanban is one of the most popular software development methodologies adopted by agile teams today.

- Kanban offers several additional advantages to task planning and throughput for teams of all sizes :

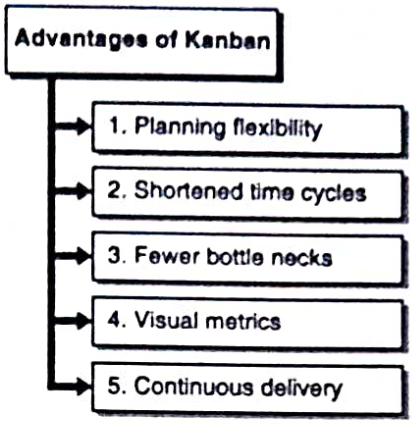


Fig. 1.15.2 : Advantages of Kanban

1. Planning flexibility

- A kanban team is only focused on the work that's actively in progress. Once the team completes a work item, they pluck the next work item off the top of the backlog.
- The product owner is free to reprioritize work in the backlog without disrupting the team, because any changes outside the current work items don't impact the team.
- As long as the product owner keeps the most important work items on top of the backlog, the development team is assured that they are delivering maximum value back to the business. So there's no need for the fixed-length iterations that are found in scrum.

2. Shortened time cycles

- Cycle time is a key metric for Kanban teams. Cycle time is the amount of time it takes for a unit of work to travel through the team's workflow - from the moment work starts to the moment it ships.
- By optimizing cycle time, the team can confidently forecast the delivery of future work.
- Overlapping skill sets lead to smaller cycle times. When only one person holds a skill set, that person becomes a bottleneck in the workflow. So teams employ basic best practices like code review and mentoring help to spread knowledge.
- Shared skills mean that team members can take on heterogeneous work, which further optimizes cycle time. It also means that if there is a backup of work, the entire team can swarm on it to get the process flowing smoothly again. For instance, testing isn't only done by QA engineers. Developers pitch in, too.
- In a Kanban framework, it's the entire team's responsibility to ensure work is moving smoothly through the process.

3. Fewer bottlenecks

- Multitasking kills efficiency. The more work items in flight at any given time, the more context switching, which hinders their path to completion.
- That's why a key tenant of Kanban is to limit the amount of work in progress (WIP). Work-in-progress limits highlight bottlenecks and backups in the team's process due to lack of focus, people, or skill sets.
- For example, a typical software team might have four workflow states: To Do, In Progress, Code Review, and Done.
- They could choose to set a WIP limit of 2 for the code review state. That might seem like a low limit, but there's good reason for it: developers often prefer to write new code, rather than spend time reviewing someone else's work.
- A low limit encourages the team to pay special attention to issues in the review state, and to review others work before raising their own code reviews. This ultimately reduces the overall cycle time.

4. Visual metrics

- One of the core values is a strong focus on continually improving team efficiency and effectiveness with every iteration of work.
- Charts provide a visual mechanism for teams to ensure they're continuing to improve. When the team can see data, it's easier to spot bottlenecks in the process (and remove them).
- Two common reports Kanban teams use are control charts and cumulative flow diagrams.
- A control chart shows the cycle time for each issue as well as a rolling average for the team.

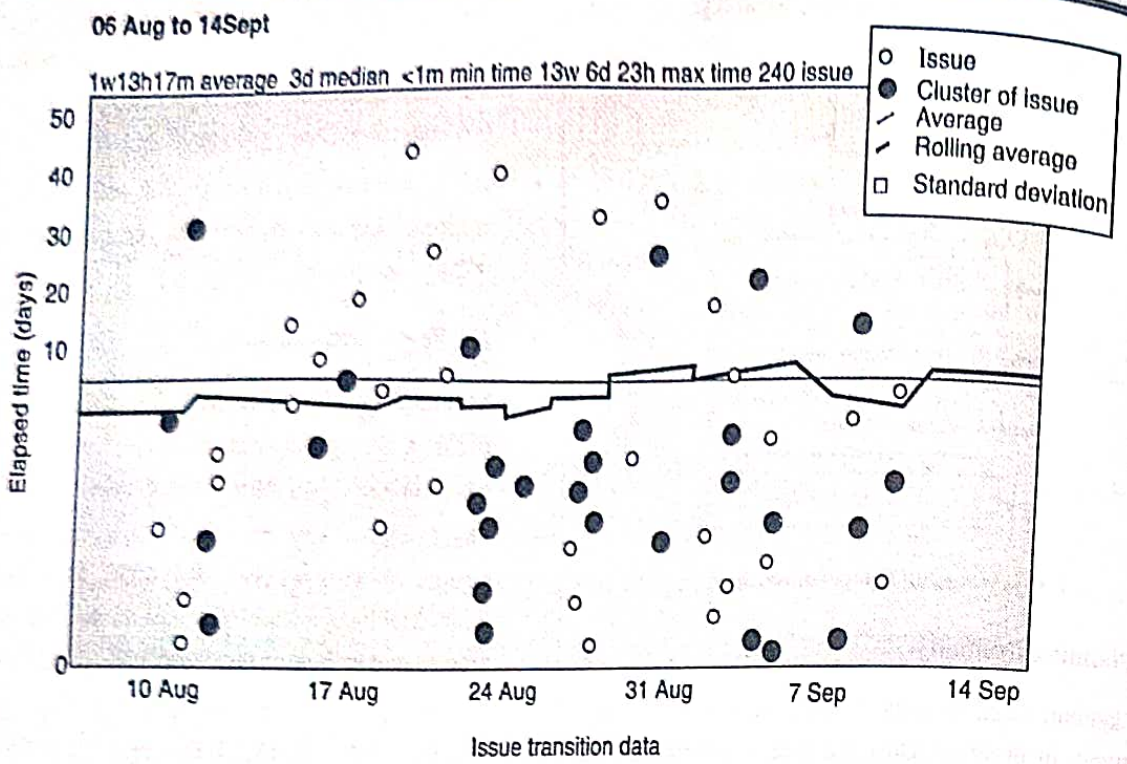


Fig. 1.15.3 : Control Chart

- A cumulative flow diagram shows the number of issues in each state. The team can easily spot blockages by seeing the number of issues increase in any given state. Issues in intermediate states such as "In Progress" or "In Review" are not yet shipped to customers, and a blockage in these states can increase the likelihood of massive integration conflicts when the work does get merged upstream.

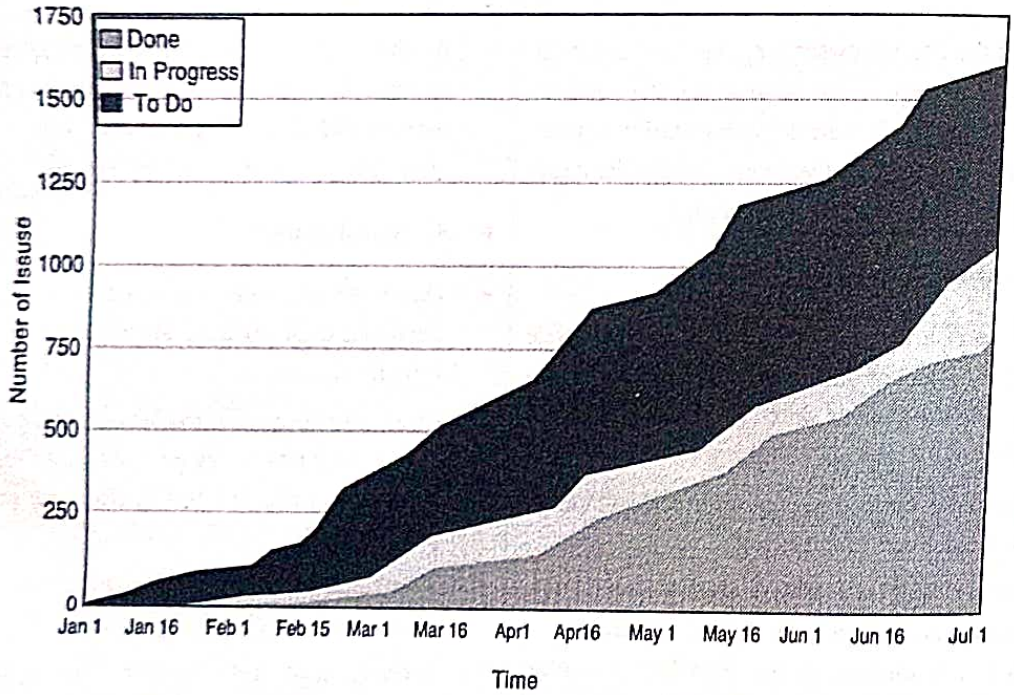


Fig. 1.15.4 : Cumulative Flow Diagram

5. Continuous Delivery

- We know that continuous integration the practice of automatically building and testing code incrementally throughout the day is essential for maintaining quality. Now it's time to meet continuous delivery (CD).
- CD is the practice of releasing work to customers frequently even daily or hourly.
- Kanban and CD beautifully complement each other because both techniques focus on the just-in-time (and one-at-a-time) delivery of value.
- The faster a team can deliver innovation to market, the more competitive their product will be in the marketplace. And Kanban teams focus on exactly that: optimizing the flow of work out to customers.

Chapter Ends...

❑❑❑

Note

MODULE II

CHAPTER 2 Requirements Analysis and Cost Estimation

Syllabus

Software Requirements : Functional & non-functional – user-system requirement engineering process – feasibility studies – elicitation – validation & management – software prototyping – S/W documentation – Analysis and modelling Requirement Elicitation, Software requirement specification (SRS) 3Ps (people, product and process) Process and Project metrics Software Project Estimation: LOC, FP, Empirical Estimation Models - COCOMO II Model.

2.1 Requirement 2-2
2.2 Types of Requirements 2-3
2.2.1 Functional Requirements 2-3
2.2.2 Non Functional Requirements (NFR) 2-4
2.2.2(A) Types of Non-Functional Requirements 2-4
2.2.3 Domain Requirements 2-4
2.3 Requirements Engineering Tasks 2-4
UQ. What are different requirements engineering tasks? **MU - May 16, Dec. 16, May 18, 5 Marks** 2-4
UQ. Explain Requirement Engineering Activities. In which Requirement Engineering Activity you will consider stakeholders. **MU - May 17, 10 Marks** 2-4
2.4 Requirement Gathering / Elicitation 2-6
2.5 Requirement Analysis 2-7
UQ. How important is Requirements analysis? **MU - Dec. 16, 5 Marks** 2-7
UQ. What is the importance of Requirements analysis ? **MU - May 18, 5 Marks** 2-7
2.6 Identifying Stakeholders 2-8
2.7 Multiple Viewpoints 2-8
2.8 SRS (Software Requirement Specification) 2-10
2.8.1 Advantages of SRS 2-10
2.8.2 Characteristics of SRS 2-11
2.8.3 Format of SRS 2-11
2.8.4 SRS Document for Online Student Feedback System 2-13
2.8.5 SRS for Hospital Management System 2-15
2.9 Project Estimation 2-19
2.9.1 Metrics for Size Estimation 2-20
UQ. Explain different metrics for size estimation. **MU - May 18, 5 Marks** 2-20
2.9.2 Line of Code (LOC) 2-20
2.9.3 FP (Function Points) Based Estimation 2-21
2.9.4 Use Case Based Estimation 2-22
• **Chapter Ends** 2-26

Syllabus Topic : Requirement

2.1 REQUIREMENT

GQ. What do you mean by requirement ? (2 Marks)

- Requirement the information which describes the user's expectation about the system performance is called as requirement.
- The requirement must be clear and unambiguous. Some requirement definition has to face problem of lack of clarity.

Characteristics of requirements

There are various characteristic of requirements. They are as follows :

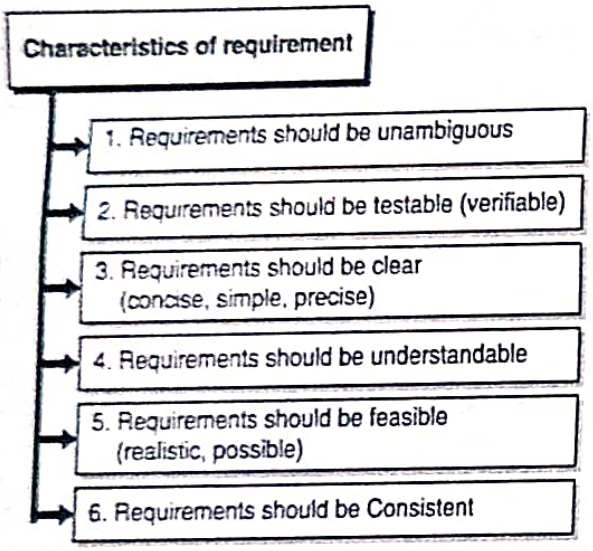


Fig. 2.1.1 : Characteristics of requirements

1. Requirements should be unambiguous

- Ambiguous means the single word or statement has more than one meaning. If requirements contain ambiguity then it is difficult to fulfill the requirements correctly.
- Therefore the requirements should be unambiguous means non confusing.

2. Requirements should be testable (verifiable)

The requirements should be testable means tester should be able to easily test the requirement whether they are implemented successfully or not. For easy test, the requirement should be clear and unambiguous.

3. Requirements should be clear (concise, simple, precise)

- The requirements should be clear. They should not contain any unnecessary information.
- If there is any unnecessary information then it becomes difficult to achieve the appropriate fulfillment of the requirement.

4. Requirements should be understandable

- The requirements should be understandable means when anyone read it then it should be easily understood by that person. To make it understandable proper conventions are used.
- It should be grammatically correct.

5. Requirements should be feasible (realistic, possible)

- The requirement should be completed within the given time and budget.
- Specified requirement said to be feasible if it can be implemented using existing technology, with estimated budget and time.

6. Requirements should be Consistent

- Consistency is an important feature of requirements. It means that all inputs must be processed similarly.
- It should not happen that processes produce different outputs for same inputs coming from different sources.

Requirement Engineering

- The procedure which collects the software requirements from customer, analyze and document them is called as **requirement engineering.**
- The aim of requirement engineering is to create and maintain 'System Requirements Specification' document.
- Requirements engineering is the process of understanding and defining which services are required and identifying the constraints on these services.
- Requirement engineering processes ensures your software will meet the user expectations and ending up with high quality software.
- It's a critical stage of the software process as errors at this stage will reflect later on in the next stages, which definitely will cause you higher costs.

- At the end of this stage, a requirement document that specifies the requirements will be produced and validated with the stakeholders.

Syllabus Topic : Types of Requirements

2.2 TYPES OF REQUIREMENTS

GQ. Explain the types of requirements. (10 Marks)

- The requirements, which are in general assumed, are divided into three categories, namely, functional requirements, non-functional requirements, and domain requirements.

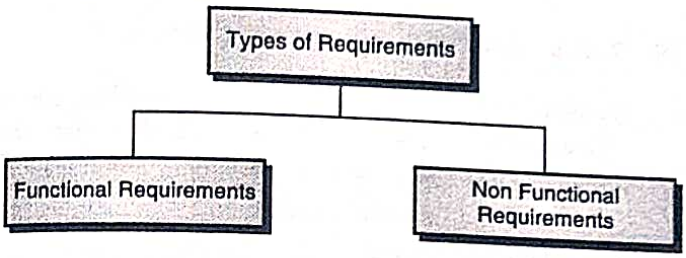


Fig. 2.2.1 : Types of requirements

2.2.1 Functional Requirements

GQ. Explain Functional Requirements. (5 Marks)

- The Functional requirements specification create document regarding the operations as well as activities which a system must be able to carry out.
- In software engineering, a functional requirement is used to define a function regarding a system or its component, in which a function is described as a specification of behavior between outputs and inputs.
- In the functional requirements there may be elements such as calculations, technical details, data manipulation and processing, and other specific functionality which define what a system is supposed to implement.
- As defined in requirements engineering, functional requirements is used to mention specific results of a system.
- This should be contrasted with non-functional requirements that mention overall characteristics like as cost and reliability.
- Functional requirements drive the application architecture of a system, while non-functional requirements drive the technical architecture of a system.

Functional Requirements should include

- Descriptions of data to be entered into the system.
- Descriptions of operations performed by each screen.
- Descriptions of work-flows performed by the system.
- Descriptions of system reports or other outputs.
- Persons who can enter the data into the system.
- Way the system meets applicable regulatory requirements.

Examples of Functional Requirements

Functional requirements must contain functions performed by specific screens, outlines of work-flows performed by the system, and other business or compliance requirements the system must meet.

Interface requirements

- Field 1 accepts numeric data entry.
- Field 2 only accepts dates before the current date.
- Screen 1 can print on-screen data to the printer.

Business Requirements

- Data must be entered before a request can be approved.
- Clicking the Approve button moves the request to the Approval Workflow.
- All personnel using the system will be trained.

Regulatory/Compliance Requirements

- The database will have a functional audit trail.
- The system will limit access to authorized users.
- The spreadsheet can secure data with electronic signatures.

Security Requirements

- Members of the Data Entry group can enter requests but cannot approve or delete requests.
- Members of the Managers group can enter or approve a request but cannot delete requests.
- Members of the Administrators group cannot enter or approve requests but can delete requests.

2.2.2 Non Functional Requirements (NFR)

GQ. Explain Non-Functional Requirements with types. **(5 Marks)**

- These are basically the quality constraints that the system must satisfy according to the project contract.
- The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements.
- They basically deal with issues like :
  - Portability
  - Security
  - Maintainability
  - Reliability
  - Scalability
  - Performance
  - Reusability
  - Flexibility
- NFRs are classified into following types :
  - Interface constraints
  - Performance constraints: response time, security, storage space, etc.
  - Operating constraints
  - Life cycle constraints : maintainability, portability, etc.
  - Economic constraints
- The process of specifying non-functional requirements needs the knowledge of the functionality of the system, and also the knowledge of the context within which the system will operate.

2.2.2(A) Types of Non-Functional Requirements

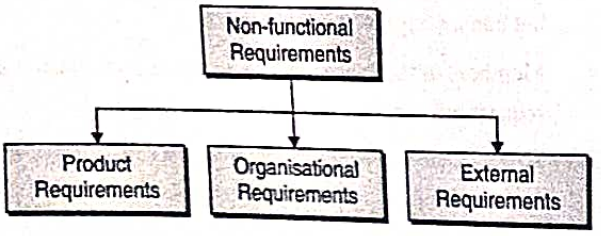


Fig. 2.2.2 : Types of non-functional requirements

1. **Product requirements :** Requirements which menti that the delivered product should have behavior in specific way, e.g. execution speed, reliability etc.
2. **Organizational requirements :** Requirements that a a result of organizational policies as well as procedure such as process standards which has been use implementation necessities etc.
3. **External requirements :** Requirements which ar generated from the factors that are external to th system and its development process, e.g interoperability requirements, legislative requirement.

2.2.3 Domain Requirements

- Domain requirements are used to describe system characteristics as well as features which impact the domain. Those may be new functional requirements, constraints on present requirements or may define particular computations.
- If domain requirements are not satisfied, the system may be unworkable.
- **Example :** Library system

Syllabus Topic : Requirement Engineering Task

2.3 REQUIREMENTS ENGINEERING TASKS

UQ. What are different requirements engineering tasks? **MU - May 16, Dec. 16, May 18, 5 Marks**

UQ. Explain Requirement Engineering Activities. In which Requirement Engineering Activity you will consider stakeholders. **MU - May 17, 10 Marks**

- The **activities/tasks** involved **in requirements engineering** vary widely, depending on the type of system being developed and the specific practices of the organization(s) involved.

- These may include :

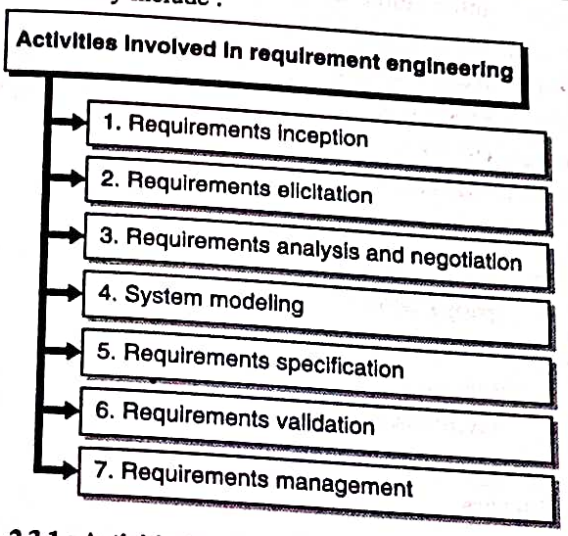


Fig. 2.3.1 : Activities involved in requirements engineering

1. Requirements Inception

- Inception is a task where the requirement engineering asks a set of questions to establish a software process.
- In this task, it understands the problem and evaluates with the proper solution.
- It collaborates with the relationship between the customer and the developer.
- The developer and customer decide the overall scope and the nature of the question.

2. Requirements Elicitation

- In requirements engineering, requirements elicitation is the practice of researching and discovering the requirements of a system from users, customers, and other **stakeholders**. The practice is also sometimes referred to as "requirement gathering".
- The term elicitation is used in books and research to raise the fact that good requirements cannot just be collected from the customer, as would be indicated by the name requirements gathering.
- Requirements elicitation is non-trivial because you can never be sure you get all requirements from the user and customer by just asking them what the system should do OR NOT do (for Safety and Reliability).
- Requirements elicitation practices include interviews, questionnaires, user observation, workshops, brainstorming, use cases, role playing and prototyping.
- Before requirements can be analyzed, modeled, or specified they must be gathered through an elicitation process. Requirements elicitation is a part of the requirements engineering process, usually followed by analysis and specification of the requirements.
- We will see more details regarding requirement gathering in section 2.4.

3. Requirements Analysis and Negotiation

Requirements are identified (including new ones if the development is iterative) and conflicts with stakeholders are solved. Both written and graphical tools (the latter commonly used in the design phase but some find them helpful at this stage, too) are successfully used as aids. Examples of written analysis tools : use cases and user stories. Examples of graphical tools: UML and LML.

4. System modelling

- Some engineering fields (or specific situations) require the product to be completely designed and modeled before its construction or fabrication starts and, therefore, the design phase must be performed in advance.
- For instance, blueprints for a building must be elaborated before any contract can be approved and signed.
- Many fields might derive models of the system with the Lifecycle Modeling Language, whereas others, might use UML.

Note : In many fields, such as software engineering, most modeling activities are classified as design activities and not as requirement engineering activities.

5. Requirements specification

- Requirements are documented in a formal artifact called Requirements Specification (RS).
- Nevertheless, it will become official only after validation. A RS can contain both written and graphical (models) information if necessary. Example : Software Requirements Specification (SRS).

6. Requirements validation

It is the process of checking whether the documented requirements and models are consistent and meet the

needs of the stakeholder. Only if the final draft passes the validation process, the RS becomes official.

7. Requirements management

Managing all the activities related to the requirements since inception, supervising as the system is developed and, even until after it is put into use (e. g., changes, extensions, etc.)

Syllabus Topic : Requirement Gathering / Elicitation

2.4 REQUIREMENT GATHERING / ELICITATION

GQ. Mention any four requirement Elicitation Methods. **(5 Marks)**

- In requirements engineering, requirements elicitation is the practice of researching and discovering the requirements of a system from users, customers, and other stakeholders. The practice is also sometimes referred to as "requirement gathering".
- The term elicitation is used in books and research to raise the fact that good requirements cannot just be collected from the customer, as would be indicated by the name requirements gathering.
- Requirements elicitation is non-trivial because you can never be sure you get all requirements from the user and customer by just asking them what the system should do OR NOT do (for Safety and Reliability).
- Requirements elicitation practices include interviews, questionnaires, user observation, workshops, brainstorming, use cases, role playing and prototyping.
- Before requirements can be analyzed, modeled, or specified they must be gathered through an elicitation process. Requirements elicitation is a part of the requirements engineering process, usually followed by analysis and specification of the requirements.

Different ways to identify customer requirement

1. Interviews

- Interviews are important way for gathering requirements. Organization may take various kinds of interviews such as :
  - Structured or closed interviews in which information to be collected from customer is decided in advance.
  - Non-structured or open interviews in which details needs to be collected from customer are decided in advance.
  - Oral interviews.
  - Written interviews.
  - Face-to-face interviews which are held among two people across the table.
  - Groups of persons are participating in group interviews. They support to cover any missing requirement as number of people participates in this process.

2. Surveys

- Organization may perform surveys of different stakeholders by questioning them about their requirements and expectation from the proposed software system.
- The advantage of this technique is that, collecting requirements is economically beneficial because it collects requirements from a large number of persons at same time.
- Surveys are less effective method of data discovery.
- Frequently, a survey's main finding is that other questions should have been asked instead. Surveys are most useful for capturing clear factual information.

3. Questionnaires

- Questionnaires are a document way which contains in-built set of objectives that is based on questions and their options.
- This document is given to all stakeholders to give answer of those questions which are gathered and compiled.
- Output of this mechanism is, if an answer for some question is not given in the questionnaire, the issue may be remaining unattended.
- The advantage of this technique is that, collecting requirements economically is beneficial because it collects requirements from huge amount of persons at same time.
- Questionnaires are less effective method of data discovery.

4. **Task analysis**

- In this technique, team of software developers and engineers may identify the functional specifications for which the new system is needed to develop.
- If the customer has some software to do the particular operations then it is analyzed by this team to find out requirements for proposed system.

5. **Domain Analysis**

- Each software put into some domain category.
- The experienced persons in that domain can be a great support to study general and specific requirements.

6. **Brainstorming**

- Brainstorming is an informal debate held between different stakeholders and all their suggestions are documented for further requirement analysis.

7. **Prototyping**

- Prototyping is a technique in which, we create user interface without including detail functionality for user to interpret features of desired software product.

It supports providing details about requirements.

If the client does not know its own requirements, in such case the developer develop a prototype based on requirements provided at initial stage.

The prototype is displayed to the client and the feedback is collected from client.

The client feedback is used as an input for requirement gathering.

Observation

Team of experienced persons visits the organization or workplace of client. They observe existing system's work.

They observe the flow of control at client's end and how execution problems are dealt. The team itself draws some conclusions which aid to form requirements expected from the software.

2.5 REQUIREMENT ANALYSIS

Q. How important is Requirements analysis?

MU - Dec. 16, 5 Marks

Q. What is the importance of Requirements analysis ?

MU - May 18, 5 Marks

- Requirements analysis is vital phase of requirement engineering process after Elicitation.
- In this activity, we understand and refine the collected requirements to make them consistent and non-confusing.
- This activity reviews each and every requirement and may also give graphical view of the overall system.
- After the analysis, it is supposed that the understanding of requirments of project are increased significantly.
- In this activity, developer can communicate with customer to clear confusing points and to understand which requirements are more vital than others.
- The significant factors in the requirement analysis activity are :
  1. Identify and solve confusion among requirements at the same level and among different levels.
  2. Identify the boundaries of the proposed software system and the way in which it communicate with environment.
  3. Evaluate customer requirements for overall system requirements and then break down them into individual component level.
- The objective of the Requirement Analysis is generating a solution to implement the requirements.
- Process of the Requirements Analysis includes following steps :
  1. Analysis of the requirements
  2. Description of the solution
  3. Cost estimate and prioritization.
- Here are some examples of how we can consider problems occurred in requirement analysis.
- Stakeholders don't recognize what they really required.
- Stakeholders state the requirements in their own words.
- Various stakeholders may have different and confusing requirements.
- Organizational and political factors may affect the software needs.
- The requirements are modified at the time of requirement analysis process.
- New stockholders may be included and the business environment may change.

Syllabus Topic : Identifying Stakeholders

2.6 IDENTIFYING STAKEHOLDERS

GQ. Write note on Identifying Stakeholders. (5 Marks)

- **Stakeholders :** A group or individual that is affected by or in some way accountable for the outcome of an undertaking. Stakeholders may include project members, suppliers, customers, end users, and others
- Stakeholder identification is very important and comes before any other RE activity: it is necessary to first identify who they are and how much is their importance.
- Consider there is need to elicit and document requirements of a software system.
- For determining the relevant stakeholder roles, we have to search for persons or organizations who :
  - posses an active interest in the system since they are the one who actually use it or are straightforwardly concerned with processes that the system will change;
  - must manage, introduce, operate, as well as maintain the respective system further process of deployment;
  - are involved in the process of system development and plays any of the role such as an architect, developer, tester, quality engineer, or project manager;
  - are responsible for the respective business or process which has been supported or automated by the system;
  - posses a monetary interest (for example, they've given order for the system, paid for it, or are sales of the system);
  - constrain the system as regulators; or are affected in a negative manner by the system (supposed negative stakeholders).
- Not all stakeholders have similar importance, hence there is need to prioritize the roes of identified stakeholder-for example, into critical, major, and minor.
- If there is negligence in selecting proper stakeholders with their exact importance, there may be significant impact on the success of system.
- Finally, there is need to choose representative individuals or groups from the identified and prioritized stakeholder roles with whom it will be convenient to elicit and validate system requirements.

Syllabus Topic : Multiple Viewpoints

2.7 MULTIPLE VIEWPOINTS

GQ. Explain Multiple Viewpoints in requirement engineering. (10 Marks)

- Viewpoint-oriented approach in the context of requirements engineering helps to organise both the elicitation process as well as the requirements themselves with the help of different viewpoints.
- A viewpoint is a mechanism used to organize the requirements for a software system, depending upon some sort of perspectives such as an end-user perspective or a manager's perspective.
- An important advantage of viewpoint-oriented analysis is that it is able to recognize the presence of multiple perspectives and offers a framework for the purpose of discovering conflicts in the requirements collected from different stakeholders.
- It is possible to use viewpoints as a method for the purpose of classifying different types of stakeholder as well as sources of requirements.
- There are three generic types of viewpoint :
  1. **Interactor viewpoints** represent people or other elements which communicate directly with the system. For example in the bank ATM system customers and account database are interactor viewpoints.
  2. **Indirect viewpoints** represent stakeholders who do not have any interaction with the system directly but who have influence on the requirements in one or other way. In the bank ATM system, management of the bank and the bank security staff are indirect viewpoints.

3. **Domain viewpoints** represent domain characteristics as well as constraints which have influence on the system requirements. In the bank ATM system, the standards that have been established for inter-bank communications are domain viewpoints.

- In general, different types of requirements are provided by these viewpoints.
- Interactor viewpoints give system requirements in detail including the system features as well as interfaces.
- Indirect viewpoints are usually provide higher-level organisational requirements and the respective constraints.
- Domain viewpoints generally provide domain constraints which are applicable to the system.
- The initial identification of viewpoints which are appropriate to a system may sometimes becomes very difficult. To resolve this, we should try to identify viewpoints of different types :
  - Providers as well as receivers of system services.
  - Systems which must interact directly with the system being specified.
  - Regulations and standards which has been applied to the system.
  - The sources regarding the system business and non-functional requirements.
  - Engineering viewpoints which impact the requirements of people, who have the responsibility to develop, manage as well as maintain the system.
  - Marketing and external affairs viewpoints which create requirements on the product features which are expected by customers.
- It is necessary that all of the existing organisational systems should inter-operate with each other in the organization.
- When a there is planning of a new system, it is necessary to plan the interactions with other systems.
- The interfaces offered by such other existing systems are designed already. These may put requirements as well as constraints on the new system also. There is need to new systems to conform to existing regulations as well as standards.
- It is must to determine high-level business and non-functional requirements in the early stage of RE process.
- The sources of these requirements may consider as significant viewpoints in a more detailed process. They can expand and develop the respective high-level requirements into more specific system requirements.
- The importance of engineering viewpoints is because of for two reasons :
  - Firstly, the engineers who are developing the respective system may have experience of same kind of system and can suggest requirements from their previous experience.
  - Secondly, the technical staff which has the responsibility of managing and maintaining the system may possess requirements which will be useful to simplify this task.
- System management requirements seem to be very important as system management costs are considered as an increasing proportion of the entire lifetime costs for a system.
- Finally, viewpoints which give requirements may come from the marketing and external affairs departments in an organization. This usually happens for web-based systems, specifically e-commerce systems.
- Web-based systems have responsibility to present a favorable image of the respective organisation which is running the system and delivering functionality to the end user.
- Once viewpoints have been identified and structured, the next step is to determine the most significant viewpoints and begin with them when discovering system requirements.
- For any non-trivial system, there are several possible viewpoints and it is really not viable to elicit requirements from all of them. Hence, there is need to organize and structure the viewpoints into a hierarchical manner.
- Viewpoints which exist in the same branch of the tree share common requirements.
- To understand this, see the viewpoint hierarchy shown in Fig. 2.7.1. This is a fairly simple hierarchy of the viewpoints which may be considered in deriving the requirements for a library system.

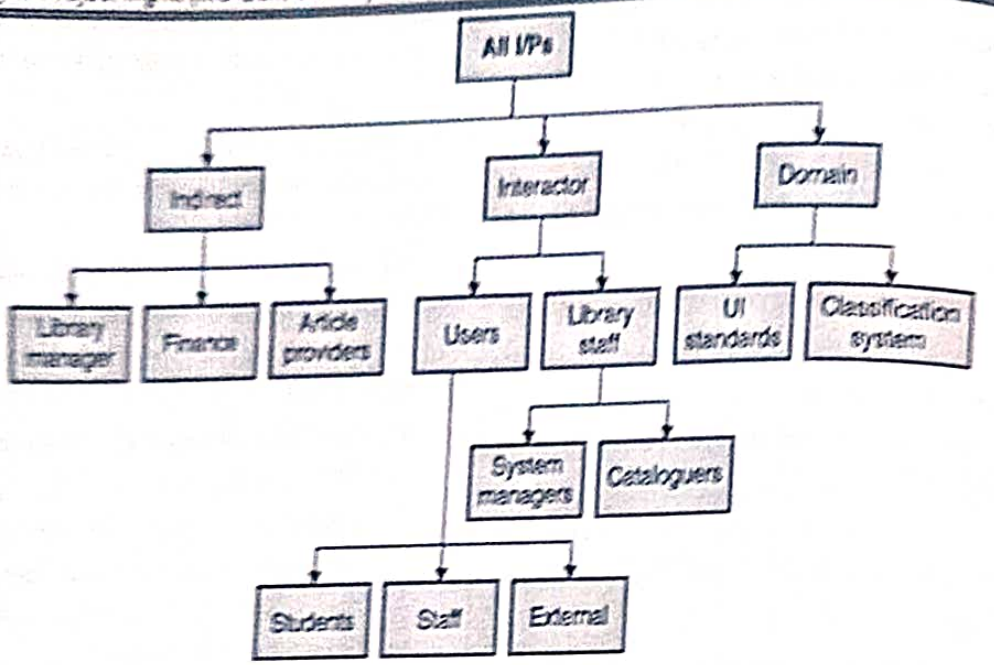


Fig. 2.7.1 : Viewpoint hierarchy for library system

Syllabus Topic : SRS (Software Requirement Specification)

2.8 SRS (SOFTWARE REQUIREMENT SPECIFICATION)

GQ. What is SRS document ? (4 Marks)

- SRS stands for Software/System Requirement Specification. SRS is a special kind of document which contains user requirements for a system which states properties and constraints that must be satisfied by a software system.
- IEEE defines software requirements specification as, 'a document that clearly and precisely describes each of the essential requirements (functions, performance, design constraints and quality attributes) of the software and the external interfaces.
- The outcome of the requirements gathering and analysis phase of the SDLC is **Software Requirements Specification (SRS)**.
- SRS is also called as **requirements document**.
- SRS is base for software engineering actions and is generated when all requirements of software project are gathered and analyzed.
- It is generally signed at the end of requirements engineering phase.
- Following are 6 requirements stated by Heninger, which SRS document should follow :
  1. SRS document should specify external system behaviour.
  2. SRS document should specify implementation constraints.
  3. It should be easily changeable if any changes occur.
  4. It should act as reference tool for maintaining the system.
  5. SRS document record forethought about the lifecycle of the system i.e. predicts changes.
  6. It must include acceptable response to undesired events.

2.8.1 Advantages of SRS

GQ. What are advantages of SRS ? (4 Marks)

1. SRS contains the base for agreement among the client and the company who develop the product as per expectations.
2. A SRS gives a base for verification of the completely developed product.
3. Using high-quality SRS, we can develop high-quality software product.
4. A high-quality SRS decreases the cost of development.

2.8.2 Characteristics of SRS

GQ. Explain any five characteristics of SRS. (10 Marks)

A good software requirement specification must satisfy following characteristics :

1. Correct

- SRS is correct when all requirements of user are described in the requirement document.
- The listed requirements must be matched with desired system.
- This depict that every requirement is analyzed to give assurance that it (SRS) contains user requirements.
- Remember that there is no specific tool or process to ensure the correctness of SRS.
- Correctness gives assurance that all stated requirements are worked as expected.

2. Unambiguous

Fig. 2.8.1 : Characteristics of SRS

- SRS does not contain any confusion when each specified requirement has single interpretation.
- This characteristic state that every requirement is individually interpreted.
- In situation, where one term has number of meanings, then its meaning must be specified in the SRS so that it will be non-confusing and simple to understand.

. Complete

SRS is complete when the requirements undoubtedly specified that which work the software product is required to perform.

This contains each and every requirement associated to performance, design and functionality.

Ranked for importance/stability

Each and every requirement has not same importance, so every requirement is recognized to make differentiation between requirements.

For this purpose, it is needed to undoubtedly recognize every requirement.

Stability refers to the probability of further modifications in the requirement.

5. Modifiable

The requirements given by user are changeable, so requirement document must be generated in such a way that those modifications can be included easily in SRS by preserving consistently the structure and style of the SRS.

6. Traceable

- SRS is observable when the source of every requirement is unambiguous and facilitates the description of every requirement in future.
- Forward tracing and backward tracing methods are used for this purpose.
- Forward tracing state that every requirement must be referencing to design and code of components.
- Backward tracing state every requirement explicitly addressing its source.

7. Verifiable

- SRS is testable when the stated requirements can be tested with a cost-effective procedure to verify whether the final software fulfills those requirements. The requirements are tested through reviews.
- Remember that clear requirement is needed for verifiability.

2.8.3 Format of SRS

GQ. Explain general format of Software Requirement Specification (SRS). (10 Marks)

Structure of SRS document includes following sections with various subsections in it :

1. Introduction

- Introduction contains brief information about the SRS. This part is further divided into following subsections :
  - (i) **Purpose :** This section specifies the main purpose of SRS document with its intended audience for whom SRS is constructed.
  - (ii) **Scope :** This section specifies scope of the system for which SRS is built.

    Scope defines not only benefits, objectives and behaviour of software product but also limitations of software so as to understand the boundary of software.

(iii) **Definition, Acronyms and Abbreviations :** To avoid ambiguity of terms specified in requirements, SRS provides definition, abbreviation and acronyms about these terms.

(iv) **References :** It contains list of all documents which are referred by this document.

(v) **Overview :** It gives overview of the document including its goals and objectives of the system.

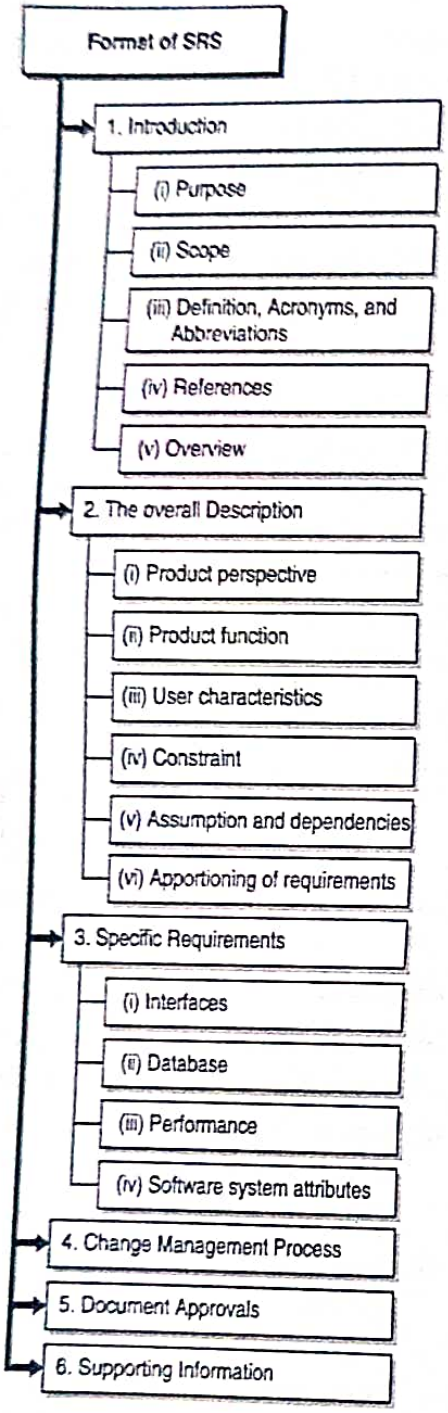


Fig. 2.8.2 : Format of SRS

2. The overall Description

- It provides overall information about the requirements. Customers/users of the system are concerned about this section. It contains following subsections :

(i) **Product perspective :** This section contains the information which states the benefits of current product over other existing products.

(ii) **Product function :** The functionality of software product summarizes in this section of SRS document.

It is possible to use diagrams to summarize the software functionality and logical relationship among variables present in the system. This information is provided in simple way so that users of the system can understand.

(iii) **User characteristics :** To use any system, it is mandatory that users of system should have some basic knowledge i.e. at least the educational qualification or basic knowledge related to the field. This section provides the criteria about the users of the system.

(iv) **Constraint :** It includes limitations of components used in the system for example, hardware limitations.

(v) **Assumption and dependencies :** It contains the list of factors on which SRS is depending. That is if the factor changes it leads to change in the SRS too.

(vi) **Apportioning of requirements :** It states the order in which the specified requirements are fulfilled.

3. Specific Requirements

This section of SRS actually specifies the Requirements briefly by taking help of information stated in above section. This section is divided into following subsections :

(i) **Interfaces**

- This section of SRS contain the information about various interfaces that are needed for product, like hardware/software interface, system interface, user interfaces, and communication interfaces.

- It means that each and every input device, output device and the communication medium used by system is introduced in this section of SRS document.

(ii) Database

To store data required and generated by system, database is required. Information about the database used in the system is given in this section.

(iii) Performance

It describes the system performance which includes the required speed, task completion time, response time and throughput of the system.

(iv) Software system attributes

- System attributes are : reliability, security, maintainability.
- This section of SRS provides the assurance that all system attributes will be provided by product and the way in which terms these attributes are satisfied/ fulfilled.

4. Change Management Process

- Due to additional user requirements, changes in the system are occurred. This section provides a way that will help to handle such kind of changes.
- As the system make any changes according to user requirements, SRS document should be changed accordingly.

5. Document Approvals

- The SRS document should be accepted by both the parties, including the customers for whom the system is developing and the developer who develop the system.
- Document Approvals section include the approval date time and sign of both the parties.

6. Supporting Information

It includes guidelines about how to use SRS, index of SRS, references, etc.

2.8.4 SRS Document for Online Student Feedback System

GQ. Build an SRS document for online student feedback system. **(10 Marks)**

Table of Contents

1. Introduction
   1.1 Purpose 1.2 Scope 1.3 Overview
2. General Description
   2.1 User Manual
3. Functional Requirements
   3.1 Description
4. Interface Requirements
   4.1 GUI 4.2 Hardware Interface
   4.3 Software Interface
5. Performance Requirements
6. Design Constraints
7. Other non-functional Attributes
   7.1 Security 7.2 Reliability
8. Operational Scenarios
9. Preliminary Schedule

1. Introduction

1.1 Purpose

- This document gives detailed functional and non-functional requirements for online student feedback system.
- The purpose of this document is that the requirements mentioned in it should be utilized by software developers to implement the system.

1.2 Scope

- This system allows the students to provide quick feedback which is provided by collage staff.
- The feedback report is generated and which is checked by HOD's. He can view grade and grade obtained to the lecturers.

1.3 Overview

- This system provides an easy solution to collage staff and students for maintaining feedback related to collage staff and infrastructure, facility.

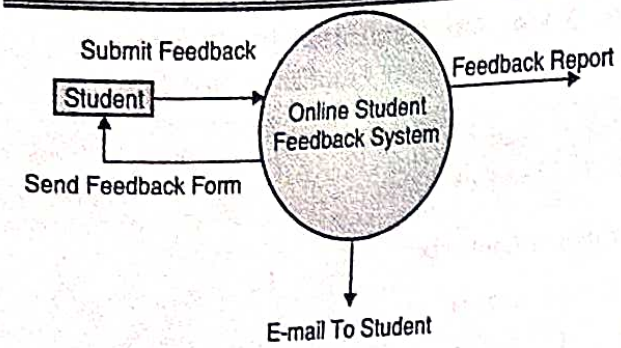


Fig. 2.8.3 : Online student feedback system

2. General Description

- This online student feedback system replaces the traditional, manual feedback system by which lots of paper work will be reduced. The teachers are able to provide feedback regarding facility and students are able to provide feedback easily. This is primary feature of this system.
- Another feature is that feedback form can be provided to student and staff by emailing for filling it.

2.1 User Manual

The system should provide Help option in which how to operate system should be explained. Also hard copy of this document should be given to user in booklet form.

3. Functional Requirement

3.1 Description

- For identity of staff, system should display staff photograph along with their names for that corresponding subject and skills.
- Statistical report accordingly subject or skill should display individual's report whenever required.

4. Interface Requirement

4.1 GUI Student

- Student can give the feedback about the lecturers on the scale of ten. Student can give feedback about the lecturers based on interaction of lecturer in the class rooms with students and facility provided by collage like infrastructure etc.

Staff : The feedback given by students can be viewed by the staff and improve their performance in teaching and other aspects.

Head of Department : The feedback report can be checked by HOD. He can view overall grades and view the grades obtained by the lecturers and give report to principal and he can give counseling to the collage staff.

Principal : Finally, report was referred by principal who can give suggestions to the lecturers to improve their teaching.

4.2 Hardware Interface

: The system should be embedded in all desktops.

4.3 Software Interface

i. Online Student Feedback System

ii. The feedback database transmitted to database server.

iii. Report generator.

5. Performance Requirements

- The system should work concurrently on multiple processors between the collage hours.
- The report should be generated immediately within one hour.

6. Design Constraints

The system should be designed within 6 months.

7. Other non-functional attributes

7.1 Security : The student and staff should be provided password to log on to the system.

7.2 Reliability : Reliability should be guaranteed due to wired connectivity.

7.3 Availability : The system should be available during college hours.

7.4 Maintainability : There should be facility to add and delete feedback form for different purpose.

7.5 Reusability : The same system will be used in each semester.

8. Operational Scenarios

- There should be student database and teacher database. The student database should contain name and feedback information.
- The teacher database should contain name, subject, skills, and other details.

. Preliminary Schedule

The system should be designed within 6 months.

2.8.5 SRS for Hospital Management System

GQ. Develop a software requirement specification (SRS) for developing a software for Hospital Management System.

Create an SRS that contains following:

1. Objective and scope
2. Product Perspective
3. Functional Requirement
4. Non Functional Requirement **(10 Marks)**

tle

System Requirement Specification Document for ospital Management System.

ojective

To get with preparing requirement document, which ll be used to capture and document all the requirements at e start of project. In the assignment we mainly focus on nctional requirements.

Introduction

1.1 Purpose

- The main purpose of our system is to make hospital task easy and is to develop software that replaces the manual hospital system into automated hospital management system. This document serves as the unambiguous guide for the developers of this software system.

1.2 Document Conventions

- HMS - Hospital Management System
- GUI - Graphical User Interface
- PHID - Patient Hospital Identification Number

1.3 Scope of the Project

- The purpose of this specification is to document requirements for a system to manage the hospital.
- The specification identifies what such a system is required to do.
- The Hospital Management System will manage a waiting list of patients requiring different treatments.
- The availability of beds will be determined and if beds are available the next appropriate patient on the list will be notified.
- Nurses will be allocated to wards depending on ward sizes, what type of nursing is needed, operating schedules, etc.
- The current manual method of managing patients, nurses, and beds is time consuming and error prone. It is also difficult to manage the large paper flow involved in this process.
- The Hospital Management System will allow hospital administrative staff to access relevant information efficiently and effectively.
- The goal of HMS is to manage nurses, patients, beds, and patients' medical information in an cost-effective manner.
- All of these sub-systems (managing nurses, beds, patient medical information) need to be designed and implemented so that HMS can run effectively.

2. Overall Description

2.1 Product Perspective

The HMS is designed to help the hospital administrator to handle patient, nurse and bed information. The current design goal is to build an internal system to achieve the functionality outlined in this specification.

2.2 Product Functions

- The HMS will allow the user to manage information about patients, nurses, and beds. Patient management will include the checking-in and checking-out of patients to and from the hospital.
- The HMS will also support the automatic backup and protection of data.

2.3 Operating Environment

Following are the requirements for running the software successfully :

- Processor - Pentium III or Higher.
- Ram - 512 MB or Higher.
- Disk Space - 10 GB or Higher.

- OS - Windows XP or Above.

2.4 Design and Implementation Constraint

- GUI only in English.
- Login and password is used for identification of user and there is no facility for guest.

2.5 Assumption and Dependencies

- It is assumed that one hundred compatible computers will be available before the system is installed and tested.
- It is assumed that Hospital will have enough trained staff to take care of the system.

3. External Interface Requirements

3.1 User Interface

Input from the user will be via keyboard and mouse. The user will navigate through the software by clicking on icons and links. The icons will give appropriate responses to the given input.

3.2 Hardware Interface

These are the minimum hardware interfaces :

- Processor - Pentium III or Higher.
- Ram - 512 MB or Higher.
- Disk Space -10 GB or Higher.

3.3 Software Interface

These are the minimum software interfaces :

- Technologies : C# .Net 2.0
- Database : SQL server (standard edition).
- Operating system : Windows XP or above.

4. System Features

4.1 System Features

- **Work Scheduling :** Assigning nurses to doctors and doctors to patients.
- **Admissions :** Admitting patients, assigning the patients to appropriate wards.
- **Patient Care :** Monitoring patients while they are in the hospital.
- **Surgery Management :** Planning and organizing the work that surgeons and nurses perform in the operating rooms.
- **Ward Management :** Planning and coordinating the management of wards and rooms.
- **Waiting list :** Monitoring to see if there are any patients waiting for available beds, assigning them to doctors and beds once these become available.

5. Other Non-functional Requirements

5.1 Performance Requirements

The performance of our software is at its best when the following regularly are done :

- Password Management
- Regular Database Archiving
- Virus Protection

5.2 Safety Requirements

Humans are error-prone, but the negative effects of common errors should be limited. e.g. users should realize that a given command will delete data, and be asked to confirm their intent or have the option to undo.

5.3 Security Requirements

- Each member is required to enter an individual Username and password when accessing the software. Administrators have the option of increasing the level of password security their members use.
- The data in the database is secured through multiple layers of Protection.
- One of those security layers involves member passwords. For maximum Security of your software, each member must protect their password.

5.4 Software Quality Attributes

The Quality of the system is maintained in such a way that it can be very user-friendly. The software quality attributes are assumed as follows :

- Accurate and hence reliable.
- Secured.
- Fast Speed.
- Compatibility.

3Ps (people, product and process)

GQ. Explain 3Ps.

The key to increasing the chances of success on software projects is to maintain the proper relationships among people, process, and product -- the three "Ps."

People

- The primary element of any project is the people. People gather requirements, people interview users (people), people design software, and people write software for people. No people -- no software.
- The best thing that can happen to any software project is to have people who know what they are doing and have the courage and self-discipline to do it. Knowledgeable people do what is right and avoid what is wrong.
- Courageous people tell the truth when others want to hear something else. Disciplined people work through projects and don't cut corners.

Find people who know the product and can work in the process.

Process

- Process is how we go from the beginning to the end of a project. All projects use a process. Many project managers, however, do not choose a process based on the people and product at hand. They simply use the same process they've always used or misused.
- Let's focus on two points regarding process: (1) process improvement and (2) using the right process for the people and product at hand. Many people believe that process improvement is the key to increasing your ability to produce software. You must have a process before you can improve it.

The CMM has a series of levels through which an organization can progress from the chaotic level 1 (Initial) up through level 5 (Optimizing). It can take an organization 9 to 24 months to move from one level to another.

The CMM provides a long-term program for a quality software process. It lays out key process areas that build on practices from lower levels.

Successfully addressing the key process areas at each CMM level can help an organization establish and improve its software process.

Level 2 of the CMM (Repeatable) requires an organization to have written software process policies and to follow them on every project. It means having a software process. Levels 3 through 5 concentrate on improving the process. The ISO 9000 standards are similar to the CMM in that they concentrate on a quality process.

- If you have process experts on your staff, you can change the order in your organization. If you are like the vast majority of us, however, it is a boon to be given a road map to follow.
- The second point to make about process is that different and useful process models exist. We must recognize them and use the one that suits the people and product in our current project, because no one model is best for every situation.
- The basic (and much-maligned) waterfall model is efficient and effective when you understand the product well. Other process models help when you don't understand the product, because they allow you to explore and learn. But they may be too fancy and a waste of time when your people do understand the product.
- I have used several process models on different successful projects. The basic waterfall model involves defining the problem (requirements) and then selecting a solution (design) and building it (code, integrate, test, maintain). This model works when it fits the people and product. Its steps are included in all the other models, with the differences being the order and number of iterations of the steps.
- The process model used in the follow-on project described earlier involved a series of deliveries. In this model, each delivery has all the required functions.
- Each succeeding delivery improves one or more attributes of the product (size, speed, etc.). This process model allows your people to learn how to optimize certain attributes, and it gives the customer a complete product along the way.
- Another process model that uses a series of deliveries provides the shell of the system, with only a few functions at first. Ensuing deliveries add more functions. This gives the customer growing capability, allows the builder to succeed month by month, and reduces the time between customer and builder interaction.
- The customer has some confidence in the builder, the product, and the project, because it is not waiting for some "big bang" to occur out in the future somewhere.

- Barry Boehm's spiral model concentrates on evaluating and reducing risk. It is most helpful on innovative projects.
- Often the product in a project comprises smaller products. There is no reason to use the same process model for each of the smaller products. Use the process that fits the smaller product and the people who are building it.

Product

- The product is the result of a project. The desired product satisfies the customers and keeps them coming back for more. Sometimes, however, the actual product is something less.
- The product pays the bills and ultimately allows people to work together in a process and build software. Always keep the product in focus. Our current emphasis on process sometimes causes us to forget the product. This results in a poor product, no money, no more business, and no more need for people and process.
- Two other aspects of the product. These are (1) the difficulty of the product and (2) the external and internal quality.
- The difficulty of the product influences the process needed. "Difficult" is subjective and depends on how familiar your people are with the product. For me, a text editor is a very difficult product, while an intelligent image analyzer is simple. Is the product familiar or new to your people? Is it new to everyone in the world? Is the user interface a major portion of the product? The answers to these questions determine the "difficulty" of the product and the type of process needed.
- Difficult products demand process models that allow for experimenting and learning. Easy products call for process models that are simple, straightforward, and efficient (like the waterfall). Difficult products become easier when you bring in people with knowledge of the product.
- Keep an eye on the external and internal quality of the product. External quality is what the customer sees. The customer is happy if the product has all the required functions, is easy to learn and use, runs quickly, and doesn't require much disk space and memory.
- Internal quality is what the builder sees. High internal quality indicates, among other things, that the design and code are easy to understand and the software is modifiable and portable.
- When your customer announces he or she is changing hardware platforms and operating environments, high internal quality lets you change the product quickly and easily.
- These quality factors also influence the people and process. If portability (internal quality) is important, you need Unix and Macintosh people as well as MS-DOS/MS- Windows people on the project. If these people are not available, you must allow for learning and risk in your process.

THE RELATIONSHIPS OF THE THREE Ps

The following scenarios illustrate how to keep the proper relationships among people, process, and product.

Example 1 :

Write a desktop publishing program to run on any Unix platform using the Motif window manager. You are given the product. Your company has a strict methodology using the waterfall model with no iteration. Your company gives you a short deadline and an adequate but not wonderful budget. You are given the process.

Now, what people do you need for this project? The strict process means you must do it right the first time. Therefore you need people with knowledge of this product (some desktop publishing experts and some Unix Motif experts). Attempting this project with rookies would be disastrous, unless your management gives you some freedom with schedule or much more money.

Example 2 :

Write a desktop publishing program to run on any Unix platform using the Motif window manager. You are given the product. You are given the people -- you must use the 10 programmers on hand. They know Unix and X-Windows, but they have never used Motif and know nothing about desktop publishing. What process will you use?

Your people need training, and there are many risks on this project. You need a creative process, such as spiral development or incremental delivery. Petition your management for a long schedule and extra money for training. Schedule extra time for requirements, because your people don't understand the product. Try to hire experts to train your people and help build the most difficult parts of the product.

Example 3 :

Write a desktop publishing program to run on any Unix platform using the Motif window manager. You are given the product. You can choose the people and process.

The easy answer is to hire a company that has done this before. Its people know the product and what process works best. As an alternative, you could use this as an opportunity to build up a new group of programmers in your Company Bring in a mixture of new talent, train these people, and adjust the process to fit them. Your company would then have new capabilities for future projects.

Example 4 :

Management wants you to select a new product from a list of ideas. You can use the people on hand, and you have several process models at your disposal. Which product should you attempt to build? The answer, of course, is the product that will bring the most profit. Included in this profit question are the risk and cost of failure incurred by attempting the wrong product. Which product fits your people's area of expertise? Do you have processes that will help your people build a product outside their area of expertise? Do you have the budget for some consultants and training? Do you have the time for training?

2.9 PROJECT ESTIMATION

GQ. Explain Software Project Estimation in detail. **(10 Marks)**

- Measurements in the physical world can be divided into two categories: direct measures (e.g., the weight of a product) and indirect measures (e.g., the "quality" of product).
- In the same manner, it is possible to categorize software metrics.
- In the direct measures of the **software** process the elements involved are cost and effort applied.
- Whereas in the direct measures of the **product** the elements involved are LOC (lines of code) generated, speed of execution, memory size, and bugs recorded over some particular time span.
- In the indirect measures of the product the elements involved are functionality, quality, complexity, efficiency, reliability, maintainability, and several other capabilities.
- The cost and effort necessary to develop software, the number of lines of code generated, and other direct measures are comparatively simple to gather, as long as particular conventions for measurement are set in advance.
- However there is difficulty in assessing and measuring directly the quality as well as functionality of software or its efficiency or maintainability.
- When the software metrics domain is partitioned into process, project, and product metrics, then it is observed that product metrics which are private to an individual are usually integrated to develop project metrics which are public to the respective software team.
- The consolidation of Project metrics is done to generate process metrics which are public to the software firm as a whole.
- But what will be the way for an organization to combine metrics which are coming from different individuals or projects?
- To understand this we will take a simple example. Individuals on 2 separate project teams record and make categorization of all the errors which are encountered during the software process.
- Those individual measures can be further combined to form team measures.
- Team X encounters hundred errors during the software process before release. Team Y encounters seventy errors.
- Being everything is similar, which team is more efficient in searching errors throughout the process?

Since we do not have knowledge of size or complexity of the projects, we cannot provide answer to this question.

- However, if we do normalization of the measures, it is probable to generate software metrics which enable comparison to broader organizational averages.

2.9.1 Metrics for Size Estimation

UQ. Explain different metrics for size estimation.
MU - May 18, 5 Marks

- Size oriented software metrics are in general derived by the process of normalizing quality and/or productivity measures by assuming the software size which has been produced.
- If a software organization is able to keep simple records, a table regarding size-oriented measures will be as shown in Table 2.9.1.
- In Table 2.9.1, the details shown about all the software projects are developed in last five years.
- **Project alpha :** 12,100 lines is the total size of code with 24 person-months of effort at a price of $168,000.

Table 2.9.1

Project	LOC	Effort	$(000)	Pp doc	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6

- It is important that the effort and cost recorded in the table does not represent only coding, but it also represents various software engineering activities such as analysis, design and testing.
- The further information shows that the size of documentation is of 365 pages.
- Before the release of software the count of errors was 134. Within the first year after release to the customer, the count of defects was twenty nine. The number of people involved in development was three.
- For the purpose of metrics development which can be assimilated with other same types of metrics from various projects, as our normalization value, the Line of code can be selected.
- As per the table information, a set of simple size oriented metrics can be generated for all the projects :
  - Errors per KLOC (Thousands line of code)
  - Defects per KLOC,
  - $ Per KLOC,
  - Pages of documentation per KLOC
- The Size-oriented metrics are not considered as the best solution for measuring the software process.

Syllabus Topic : Line of Code (LOC) Based Estimation

2.9.2 Line of Code (LOC)

GQ. Write note on LOC. **(5 Marks)**

- LOC (Line of Code) is considered as the most simplest among all metrics which are available for the purpose of estimating project size. Because of its simplicity, this metric is very popular.
- With the help of this metric, it is possible to estimate the project size by the way of counting the number of source instructions present in the respective developed program.
- It is most likely that in the process of counting the number of source instructions, lines which comment the code as well as the header lines will be ignored.
- At the end of a project it is very easy to determine the LOC count. However, in initial stage, exact estimation regarding the LOC count of a project is not easy.
- For the purpose of estimating the LOC count in the initial phase of a project, project managers generally split the problem into modules, and further those modules into submodules and so on, unless and until it is not possible to approximately predict the sizes of the different leaf-level modules.
- Experience of developing similar products is useful to be able to do this. By taking the reference of the estimation of the lowest level modules, total size estimation is done by the project managers.

Syllabus Topic : FP (Function Points) Based Estimation

2.9.3 FP (Function Points) Based Estimation

GQ. Explain Function Point Estimation Technique in detail. **(5 Marks)**

- As a normalization value, a measure of the functionality which is given by the application is taken by the Function-oriented software metrics.
- It is not possible to measure the functionality directly. Hence using other direct measures, it is necessary to derive it indirectly.
- The concept of Function-oriented metrics was proposed initially by Albrecht. A measure called the function point is suggested by him.
- Function points are basically derived with the help of an empirical relationship depending upon countable (direct) measures regarding the information domain of software as well as software complexity assessments.
- Five information domain characteristics are determined and counts are provided. These are as follows :
  - Number of user inputs : Each user input should be considered
  - Number of user outputs : Reports, screens, error messages, etc
  - Number of user inquiries : An on-line i/p which results on on-line o/p
  - Number of files : Each logical master file is counted
  - Number of external interfaces : Data files on storage media
- Once the collection of these data is completed, each count is assigned a complexity value. Enterprises which refer function point methods establish criteria for the purpose of determining whether a specific entry is simple, average, or complex.

Computing function points

Measurement parameter	Count		Weighting factor: Simle	Average	Complex		
Number of inputs		x	3	4	6	=	
Number of user outputs		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							

Fig. 2.9.1

- Following relationship is in general used for computing function points (FP) :

  FP = count total [0.65 + 0.01 Σ(Fi)]

  (where count total is the sum of all FP entries)
- The Fi (i = 1 to 14) are "complexity adjustment values" depending upon responses to the following given fourteen questions :
  1. Does there is need of reliable backup and recovery to the system ?
  2. Is there any requirement of communications ?
  3. Are there distributed processing functions ?
  4. Is performance critical ?
  5. Will it is possible to execute system in current, greatly utilized operational environment ?
  6. Does there is need of on-line data entry to system ?

7. Does there is need of the input transaction to on-line data entry so as to build over multiple screens or operations ?
8. Is the updation of master files is done on-line ?
9. Is there is complexity in the inputs, outputs, files, or inquiries ?
10. Is there is complexity in internal processing ?
11. Can the code designed be reusable ?
12. Is there is involvement of conversion and installation in the design ?
13. Is the system designed for more than one installation in various organizations ?
14. Is the design of application facilitates change and ease of use by the user ?

- Now all these questions are usually answered with the help of a scale in the ranges 0 – 5.
- The constant values which are present in the equation and the weighting factors which are applied to the information domain counts are decided based on the facts.
- As soon as the calculation of function points is completed, they are used so as to normalize measures for the different factors such as software productivity, quality, and other attributes like Errors per FP, Defects per FP, S per FP, Pages of documentation per FP, FP per person-month.

Syllabus Topic : Use Case Based Estimation

2.9.4 Use Case Based Estimation

GQ. Explain Use case based Estimation. (10 Marks)

- The UCP (Use Case Points) is a software estimation technique which is used to make forecast regarding the software size for the respective software development projects.
- UCP is helpful in the situations when the UML (Unified Modeling Language) and RUP (Rational Unified Process) techniques are being used for the the purpose of software design as well as development.
- The terminology of Use Case Points is depend upon the requirements for the particular system being written with the help of use cases, which is an integral element of the UML set of modeling techniques.
- The calculation of UCP (software size) is done as per the elements of the system related use cases with factoring to account for the considerations related to technical and environmental aspects.
- Then it is possible to use the UCP for a project for the purpose of calculating the estimated effort of the particular project.
- The **method** to calculate the size estimate for development of a system depends upon a calculation with the given elements :
  1. **UUCW (Unadjusted Use Case Weight) :** The point size of the software which is concerned with the number as well as complexity of use cases.
  2. **UAW (Unadjusted Actor Weight) :** The point size of the software which is concerned with the number as well as complexity of actors.
  3. **TCF (Technical Complexity Factor) :** Factor which helps to adjust the size depending upon technical considerations.
  4. **ECF (Environmental Complexity Factor) :** Factor which helps to adjust the size depending upon environmental considerations.
- After calculation of all the four elements, it is possible to calculate the final size estimate.
- This end result is called as the Use Case Points or UCP for a software development project.

1. Unadjusted Use Case Weight (UUCW)

- The Unadjusted Use Case Weight is an important factor which is responsible for the size of the software being developed.
- It's calculation is done depending upon the number as well as complexity of the use cases for the respective system.
- For calculation of the UUCW for a system, all of the use cases must be identified and classified in either of the three categories; Simple, Average or Complex depending upon the number of transactions the use case contains.
- Each and every classification has a predefined weight assigned. After classification of all the use cases, the total weight (UUCW) is calculated by the process of adding the corresponding weights for every use case.

- The following chart illustrates the types of classification of use cases depedning upon the number of transactions and the weight values which has been assigned for all of the use cases within the classification.

Use Case Classification	No. of Transactions	Weight
Simple	1 to 3 transactions	5
Average	4 to 7 transactions	10
Complex	8 or more transactions	15

UUCW = (Total No. of Simple Use Cases × 5) + (Total No. Average Use Case × 10)

+ (Total No. Complex Use Cases × 15)

2. Unadjusted Actor Weight (UAW)

- The UAW is the next important factor which is responsible for the size of the software being developed.
- It's calculation is done depending upon the number as well as complexity of the actors for the system.
- Same as of determining the UUCW, each of the actors must be identified and classified in either of the three categories; Simple, Average or Complex depending upon the type of actor.

Each and every classification has a predefined weight assigned.

The total weight (UAW) is calculated by the process of adding the total of the weights for each of the actors. The following chart illustrates the types of classification of actors and the weight value assigned.

Actor Classification	Type of Actor	Weight
Simple	External system which should interact with the system through a well-defined API	1
Average	External system which should interact with the system through standard communication protocols (e.g. TCP/IP, FTP, HTTP, database)	2
Complex	Human actor with the help of a GUI application interface	3

AW = (Total No. of Simple actors × 1) + (Total No. erage actors × 2) + (Total No. Complex actors × 3)

3. Technical Complexity Factor (TCF)

- The TCF factor is applied in the estimation of size of the software so as to account for technical considerations of the respective system.
- Its calculation is done by the way of assigning a score in the range zero (factor is irrelevant) and five (factor is essential) to all of the thirteen technical factors listed in the given table.
- This assigned score is further multiplied by the defined weighted values of all the factors. The sum of all calculated values is nothing but the technical factor (TF).
- The use of TF can be done for computing the TCF as follows :

TCF = 0.6 + (TF/100)

Factor	Description	Weight
T1	Distributed system	2.0
T2	Response time/performance objectives	1.0
T3	End-user efficiency	1.0
T4	Internal processing complexity	1.0
T5	Code reusability	1.0
T6	Easy to install	0.5
T7	Easy to use	0.5
T8	Portability to other platforms	2.0
T9	System maintenance	1.0
T10	Concurrent/parallel processing	1.0
T11	Security features	1.0
T12	Access for third parties	1.0
T13	End user training	1.0

4. Environmental Complexity Factor (ECF)

- The ECF is the last factor which has been applied to the estimated size of the software so as to account for the relevant environmental considerations of the system.
- Its calculation is done by the way of assigning a score in the range zero (no experience) and five (expert) to all of the eight environmental factors listed in the given table.
- This assigned score is further multiplied by the defined weighted values of all the factors.
- The sum of all calculated values is nothing but the environment factor (EF).

- The use of EF can be done for computing the ECF as follows :

$$ECF = 1.4 + (-0.03 \times EF)$$

Factor	Description	Weight
E1	Familiarity with development process used	1.5
E2	Application experience	0.5
E3	Object-oriented experience of team	1.0
E4	Lead analyst capability	0.5
E5	Motivation of the team	1.0
E6	Stability of requirements	2.0
E7	Part-time staff	– 1.0
E8	Difficult programming language	– 1.0

Use Case Points (UCP)

- Finally the determination of UCP is possible after determination of following :
  - Unadjusted project size (UUCW and UAW)
  - Technical factor (TCF)
  - Environmental factor (ECF)
- The UCP is calculated as follows :

$$UCP = (UUCW + UAW) \times TCF \times ECF$$

Empirical Estimation Models - COCOMO II Model

GQ. Elaborate in detail COCOMO II Model.

- An estimation model for computer software uses empirically derived formulas to predict effort as a function of LOC or FP.11 Values for LOC or FP are estimated using the approach.
- The empirical data that support most estimation models are derived from a limited sample of projects. For this reason, no estimation model is appropriate for all classes of software and in all development environments. Therefore, you should use the results obtained from such models judiciously.
- An estimation model should be calibrated to reflect local conditions. The model should be tested by applying data collected from completed projects, plugging the data into the model, and then comparing actual to predicted results. If agreement is poor, the model must be tuned and retested before it can be used.

The Structure of Estimation Models

A typical estimation model is derived using regress analysis on data collected from past software projects. overall structure of such models takes the form

$$E = A + B * (e_v)^c \quad ...(2.9$$

where A, B, and C are empirically derived constants is effort in person-months, and e_v is the estimation variat (either LOC or FP). In addition to the relationship noted Equation (2.9.1), the majority of estimation models ha some form of project adjustment component that enables to be adjusted by other project characteristics (e.g., proble complexity, staff experience, development environment Among the many LOC-oriented estimation models propose in the literature are

$E = 5.2 * (KLOC)^{0.91}$ Walston-Felix model

$E = 5.5 + 0.72 * (KLOC)^{1.16}$ Bailey-Basili model

$E = 3.2 * (KLOC)^{1.16}$ Boehm simple model

$E = 5.288 * (KLOC)^{1.047}$ Doty model for KLOC 9

FP-oriented models have also been proposed. These include

$E = -91.4 + 0.355\ FP$ Albrecht and Gaffney model

$E = -37 + 0.94\ FP$ Kemerer model

$E = -12.88 + 0.405\ FP$ Small project regression model

A quick examination of these models indicates that each will yield a different result for the same values of LOC or FP. The implication is clear. Estimation models must be calibrated for local needs!

The COCOMO II Model

The original COCOMO (COnstructive COst Model) model became one of the most widely used and discussed software cost estimation models in the industry. It has evolved into a more comprehensive estimation model, called COCOMO II . Like its predecessor, COCOMO II is actually a hierarchy of estimation models that address the following areas:

- Application composition model. Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity are paramount.
- Early design stage model. Used once requirements have been stabilized and basic software architecture has been established.

- Post-architecture-stage model. Used during the construction of the software. Like all estimation models for software, the COCOMO II models require sizing information. Three different sizing options are available as part of the model hierarchy: object points, function points, and lines of source code

Object type	Complexity weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3 GL component			10

Fig 2.9.2 : Complexity weighting for object types

- The COCOMO II application composition model uses object points and is illustrated in the following paragraphs. It should be noted that other, more sophisticated estimation models (using FP and KLOC) are also available as part of COCOMO II.
- Like function points, the object point is an indirect software measure that is computed using counts of the number of (1) screens (at the user interface), (2) reports, and (3) components likely to be required to build the application. Each object instance (e.g., a screen or report) is classified into one of three complexity levels (i.e.,simple, medium, or difficult) using criteria. In essence, complexity is a function of the number and source of the client and server data tables that are required to generate the screen or report and the number of views or sections presented as part of the screen or report.
- Once complexity is determined, the number of screens, reports, and components are weighted according to the table illustrated in Fig. 2.9.2. The object point count is then determined by multiplying the original number of object instances by the weighting factor in the figure and summing to obtain a total object point count.
- When component-based development or general software reuse is to be applied, the percent of reuse (%reuse) is estimated and the object point count is adjusted:

$$NOP = (\text{object points}) * \left[\frac{100 - \%\ \text{reuse}}{100}\right]$$

where NOP is defined as new object points. To derive an estimate of effort based on the computed NOP value, a "productivity rate" must be derived. Fig. 2.9.3 presents the productivity rate

$$PROD = \frac{NOP}{(\text{person} - \text{month})}$$

for different levels of developer experience and development environment maturity. Once the productivity rate has been determined, an estimate of project effort is computed using

$$\text{Estimated effort} = \frac{NOP}{PROD}$$

Developer's experience/ capability	Very low	Low	Nominal	High	Very high
Environment maturity / capability	Very low	Low	Nominal	High	Very high
PROD	4	7	13	25	50

Fig. 2.9.3 : Productivity rate for object points

The Software Equation

The software equation is a dynamic multivariable model that assumes a specific distribution of effort over the life of a software development project. The model has been derived from productivity data collected for over 4000 contemporary software projects. Based on these data, we derive an estimation model of the form

$$E = \frac{LOC * B^{0.333}}{P^3} * 1/t^4 \qquad \text{...(2.9.2)}$$

Where

E effort in person-months or person-years t project duration in months or years B "special skills factor" P "productivity parameter" that reflects: overall process maturity and management practices, the extent to which good software engineering practices are used, the level of programming languages used, the state of the soft-ware environment, the skills and experience of the software team, and the complexity of the application Typical values might be P = 2000 for development of real-time embedded software, P = 10,000 for telecommunication and systems software, and P = 28,000 for business systems applications. The productivity parameter can be derived for local

conditions using historical data collected from past development efforts. You should note that the software equation has two independent parameters: (1) an estimate of size (in LOC) and (2) an indication of project duration in calendar months or years.

To simplify the estimation process and use a more common form for their estimation model, a set of equations derived from the software equation. Minimum development time is defined as

$$t_{min} = 8.14\ LOC/P^{0.43} \text{ in month for } t_{min} > 6 \text{ months} \quad \ldots(2.9.2(a))$$

$$E = 180\ Bt^3 \text{ in person-month for } E \geq 20 \text{ person-months} \quad \ldots(2.9.2(b))$$

Using Equation (2.9.2(a) and (b)) with P 12,000

$$t_{min} = 8.14 * \frac{33,200}{12000^{0.43}} = 12.6 \text{ calendar months}$$

$$E = 180 * 0.28 * (1.05^3) = 58 \text{ person-months}$$

Chapter Ends...

□□□

MODULE III

CHAPTER 3

Design Engineering

Syllabus

Design Process & quality, Design Concepts, The design Model, Pattern-based Software Design. 4.2 Architectural Design :Design Decisions, Views, Patterns, Application Architectures, Modeling Component level Design: component, Designing class based components, conducting component-level design User Interface Design: The golden rules, Interface Design steps & Analysis, Design Evaluation.

.1 Analysis Modelling 3-2
.1.1 Analysis Rules of Thumb 3-2
.2 Introduction of Analysis Elements 3-3
.3 Software Design 3-3
Q. What is Design ? MU - Dec. 16, 3 Marks 3-3
.3.1 Qualities of Good Design 3-5
.4 Design Principles 3-5
Q. Explain design principles. MU - Dec. 16, 7 Marks 3-5
5 Design Concepts 3-5
Q. Explain software design concepts. MU - May 16, 10 Marks 3-5
Q. Explain design concepts. MU - May 17, Dec. 17, 10 Marks 3-5
Q. Explain design concepts. Abstraction. MU - May 16, May 18, 10 Marks 3-6
Q. Explain design concepts : Modularity. MU - May 16, May 18, 10 Marks 3-7
.1 Interface Design Elements 3-10
Architectural Design 3-14
.1 Functions of Architectural Design 3-15
.2 Architectural Design Document 3-15
.3 Architectural Style 3-15
Q. Explain different Architectural styles. MU - Dec. 16, 5 Marks 3-15
Q. Explain architectural design for e-commerce System. MU - May 15, Dec. 15, 10 Marks 3-18
3.7 Component Level Design 3-19
UQ. Write note on Component Based Development. MU - May 16, 10 Marks 3-19
UQ. Explain component level model. MU - Dec. 17, 5 Marks 3-19
3.8 System Level Design 3-21
3.8.1 Concurrency and Sub-system Allocation 3-21
3.8.2 Task Management Components 3-22
3.8.3 Data Management Components 3-22
3.8.4 Inter Sub-system Communication 3-22
3.8.5 Object Design Process 3-23
3.9 User Interface Design 3-23
3.9.1 Command Line Interface (CLI) 3-23
3.9.2 Graphical User Interface (GUI) 3-24
3.9.2(A) GUI Elements 3-24
3.9.2(B) Application Specific GUI Components 3-25
3.9.2(C) User Interface Design Activities 3-25
3.9.2(D) GUI Implementation Tools 3-26
3.9.2(E) User Interface Golden Rules 3-26
- **Chapter Ends** 3-29

3.1 ANALYSIS MODELLING

GQ. Write importance of analysis modeling. **(5 Marks)**

- After the Requirement Analysis, as output gets the specification regarding software's operational characteristics, designates software's interface with other system elements and set up constraints which the specific software must need.
- All the way through analysis modeling the primary focus of software engineer is on *what* and *how*.
- Requirement Analysis helps the software engineer to :
  - Get detailed information of basic requirements which are set in earlier requirement engineering tasks.
  - Establish models which describe user scenarios, functional activities, problem classes with their relationships, system and class behavior, and the data flow as it is transformed.
- There are three primary objectives of analysis model :
  1. Describe customer needs
  2. Establish a basis for the software design generation, and
  3. Define a set of requirements which can be validated after creation of software.
- The **analysis model** fills the **gap** between a **system-level description** which elaborates overall functionality as it is accomplished by the process of applying software, hardware, data, human, and other system elements and a **software design** which elaborates the software application architecture.
- That means Analysis model plays a role of link in between the 'system description' and the 'design model'.
- In this model, information, functions as well as behaviour of the system is defined and all of the elements of 'design modeling' are translated into the architecture, interface and component level design.

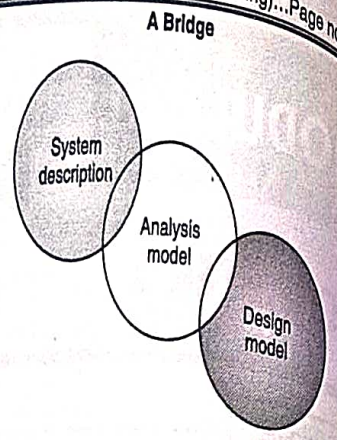


Fig. 3.1.1 : Analysis Model

3.1.1 Analysis Rules of Thumb

- The model must concentrate on requirements which are visible within the problem or business domain. The abstraction level must be comparatively high.
- Each and every element regarding the analysis model must be added to an overall understanding of software requirements and supply insight into the information domain, function and behavior of the system.
- Delay consideration of infrastructure and remaining non-functional models until design.
  - For example, there may be need of a database, but the classes required to implement it, the functions necessary to access it, and the behavior which will be exhibited after its use must be considered after the completion of problem domain analysis.
- Minimize coupling throughout the system.
  - The level of interconnectedness among classes and functions must be minimized.
- Ensure that the analysis model offers value to all the relevant stakeholders.
  - Each constituency has its own use for the model.
- Keep the possible simplest form of the model.
  - Do not introduce extra diagrams if they do not provide any new information.
  - Implement only those modeling elements which have values.

Syllabus Topic : Introduction of Analysis Elements

3.2 INTRODUCTION OF ANALYSIS ELEMENTS

GQ. Explain the various elements of analysis modeling in detail. **(5 Marks)**

1. Scenario based element

- Scenario based elements represent the system user point of view.
- **Examples :** Use case diagram, user stories.

2. Class based elements

- The object of class based elements is manipulated by the system.
- It defines the object, attributes and relationship.
- The collaboration is happening among the classes.
- **Examples :** Class diagram, collaboration diagram.

3. Behavioral elements

- These types of elements represent state of the system and how external events affect it.
- **Examples :** Sequence diagram, state diagram.

4. Flow oriented elements

- Information is transferred via a computer-based system.
- It shows way of transforming data objects when they flow between the different system functions.
- **Example :** Data flow diagram, control flow diagram.

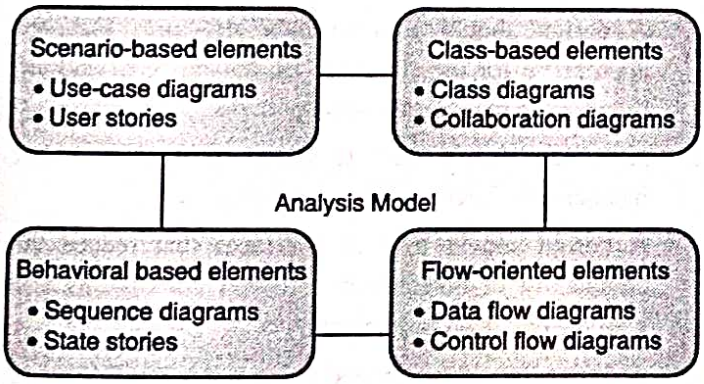


Fig. 3.2.1 : Elements of Analysis Model

3.3 SOFTWARE DESIGN

UQ. What is Design ? **MU - Dec. 16, 3 Marks**

- Once the requirements document regarding the software to be developed is presented, the phase of software design gets started.
- The requirement specification activity is considered purely related with the problem domain whereas design is considered as the initial phase of transforming the problem into a solution.
- In the design phase, all the relevant entities such as the customer, business requirements and technical considerations collaborate to formulate a product or a system.
- In design process there are several elements such as set of principles, concepts and practices, which help a software engineer to model the system or product which is to be built.
- The design model is assessed for quality and reviewed before generation of code and execution of tests.
- The design model gives detailed information regarding software data structures, architecture, interfaces and components which are necessary to employ the system.

Basic of Software Design

- Software design is considered as a phase in software engineering which develops a blueprint that will be a base for constructing the software system.
- **IEEE** defines software design as 'both a process of defining the architecture, components, interfaces, and other characteristics of a system or component and the result of that process.'
- In the design phase, important and strategic decisions are made to get the expected functionality and quality of the system.
- These decisions are considered to successfully develop the software and handle its maintenance in a way that the quality of the end product will be improved.
- Software design encompasses the set of principles, concepts, and practices that lead to the development of a high-quality system or product.
- Design concepts must be understood before the design practices are applied.

- Design practice itself leads to the creation of various representations of the software.
- Design Practices serve as a guide for the construction activity that follows.
- The design model provides details about software data structures, architecture, interfaces, and components that are necessary to implement the system.
- Mitch Kapor, the creator of Lotus 1-2-3, presented a "software design manifesto" in Dr. Dobbs Journal.
- He said :! Good software design should exhibit :

A. **Firmness :** A program should not have any bugs that inhibit its function.

B. **Commodity :** A program should be suitable for the purposes for which it was intended.

C. **Delight :** The experience of using the program should be pleasurable one.

Software design model consists of 4 designsy

1. Data/class design
2. Architectural design
3. Interface design
4. Component design

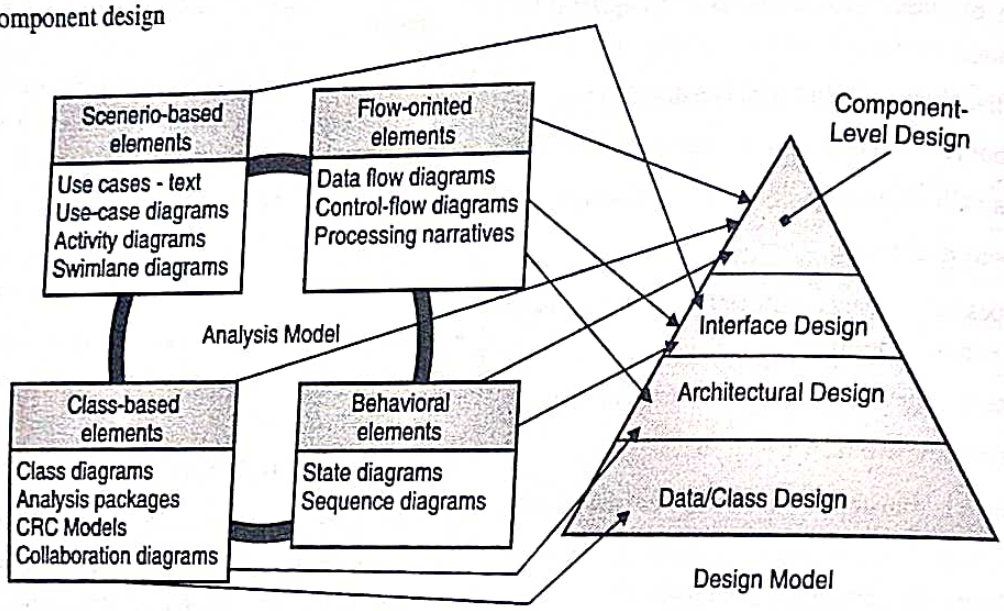


Fig. 3.3.1 : Translating Requirement Model into Design Model

Design Guidelines

1. A design should exhibit an architecture that :
   (a) Has been created using recognizable architectural styles or patterns,
   (b) Is composed of components that exhibit good design characteristics and
   (c) Can be implemented in an evolutionary fashion!
2. A design should be modular; that is, the software should be logically partitioned into elements or subsystems.
3. A design should contain distinct representations of data, architecture, interfaces, and components.
4. A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns.
5. A design should lead to components that exhibit independent functional characteristics.
6. A design should lead to interfaces that reduce the complexity of connections between components and with the external environment.
7. A design should be derived using a repeatable method that is driven from information obtained during software requirements analysis.
8. A design should be represented using a notation that effectively communicates its meaning.

3.3.1 Qualities of Good Design

GQ. Explain qualities of good software design. **(5 Marks)**

There are number of qualities of good design :

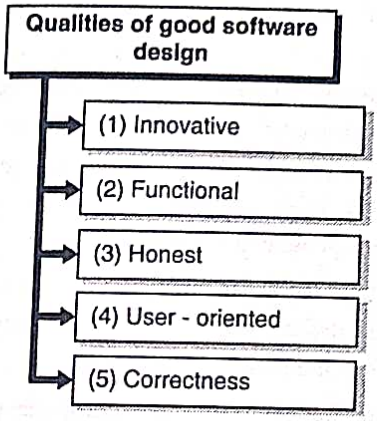


Fig. 3.3.2 : Qualities of good software design

(1) Innovative

- Innovative design can be either completely new design or redesign of existing product.
- New design gives unseen value to market where as redesign improves the quality of an existing product.

(2) Functional

Good design fulfils all its intended functions to solve user's problem. It focuses on usefulness of a product by optimizing its functionality.

(3) Honest

A good design is honest. An honest design expresses the functions and values it offers. It never attempts to modify Organiser's and User's view with promises it can't keep.

(4) User - oriented

Good design is developed based on its use and intended to improve solution to problem for its user.

User-oriented design gives intellectual as well as material value to system which in turn achieves user's satisfaction.

(5) Correctness

Correctness is an important quality of good design. A good design should correctly achieve all required functionalities as per SRS document.

3.4 DESIGN PRINCIPLES

UQ. Explain design principles. **MU - Dec. 16, 7 Marks**

Following are important design principles :

1. The design process should not suffer from 'tunnel vision'.
2. The design should be traceable to the analysis model.
3. The design should not reinvent the wheel.
4. The design should "minimize the intellectual distance" between the software and the problem as it exists in the real world.
5. The design should exhibit uniformity and integration.
6. The design should be structured to accommodate change.
7. The design should be structured to degrade gently, even when unusual data, events, or operating conditions are encountered.
8. Design is not coding, coding is not design.
9. The design should be assessed for quality as it is being created, not after the creation.
10. The design should be reviewed to minimize conceptual (semantic) errors.

3.5 DESIGN CONCEPTS

UQ. Explain software design concepts. **MU - May 16, 10 Marks**

UQ. Explain design concepts. **MU - May 17, Dec. 17, 10 Marks**

- All of the software processes are characterized by basic concepts in conjunction with some practices or methods.
- Methods represent the way by which the concepts are applied. An Old technology is replaced by new technology, several modifications occur in the methods which are used to apply the concepts for the process of developing the software.
- However, the fundamental concepts which underline the software design process always remain the same,

These concepts are :

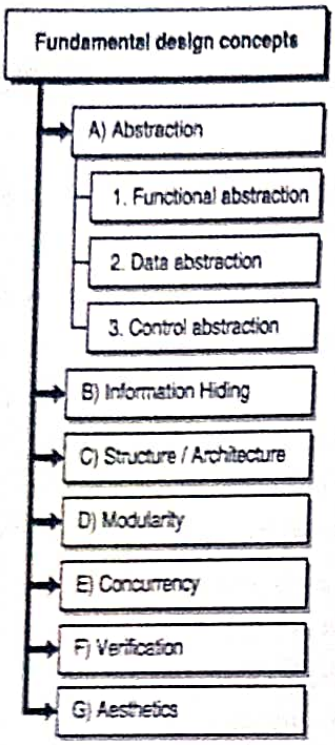


Fig. 3.5.1 : Fundamental design concepts

(A) Abstraction

UQ. Explain design concepts. Abstraction.

MU - May 16, May 18, 10 Marks

GQ. Explain with reference to design concepts in design modelling : Abstraction. **(5 Marks)**

- Abstraction refers to a process by which software designers consider components at an abstract level, at the same time as ignoring the implementation details of the components.
- IEEE defines abstraction as 'a view of a problem that extracts the essential information relevant to a particular purpose and ignores the remainder of the information.'
- There are two ways to use the concept of abstraction; as a process and as an entity.
- As a **process**, abstraction refers to a mechanism of hiding irrelevant extra details and representing just the important essential features of an element so that one can concentrate on essential things at a time.
- As an **entity**, abstraction refers to a model or view of an item.
- In the software process all the steps are accomplished via several levels of abstraction.
- At the highest level, one can get the outline of the solution to the respective problem while at the lower levels, he/she will get the solution to the problem in detail.
- For example, in the requirements analysis phase, language is used to provide a solution to the problem and as one proceeds during the software process, the abstraction level reduces and source code of the software is generated at the lowest level.
- In general three abstraction mechanisms used in software design are functional abstraction, data abstraction and control abstraction.
- These mechanisms help to control the complexity of the design process by starting from the abstract design model and ending at concrete design model in a systematic manner.

1. **Functional abstraction :** This type of abstraction includes the use of parameterized subprograms. Functional abstraction can be created as a set of subprograms known as 'groups'. In these sets, routines are exist which may be visible or invisible. The use of visible routines can be done within the containing groups and also within other groups, while invisible routines are hidden from other groups and are allowed to be used in the containing group only.
2. **Data abstraction :** This type of abstraction includes describing data which specifies a data object. For example, the data object *flower* encompasses a set of attributes (color, fragrance) which describe the flower object clearly. In this abstraction mechanism, representation as well as manipulation details are ignored.
3. **Control abstraction :** This type of abstraction states the desired effect, without conducting the exact mechanism of control. For example, in programming language like C, *if* and *while* statements are abstractions of machine code implementations containing conditional instructions.

(B) Information Hiding

It is important to specify and design modules in such a manner that the data structures as well as processing details of one module should not be accessible to any other modules.

Only required information is transferred between the modules. The way of hiding unnecessary details is referred to as **information hiding.**

IEEE defines information hiding as 'the technique of encapsulating software design decisions in modules in such a way that the module's interfaces reveal as little as possible about the module's inner workings; thus each module is a 'black box' to other modules in the system.

Information hiding is an important aspect when there is need of modifications during the testing and maintenance phase.

Advantages of information hiding

Results in low coupling.

Put emphasis on communication by controlled interfaces.

Reduces the possibility of adverse effects.

Controls the impacts of changes in one component to others.

Results in higher quality software.

(C)Structure / Architecture

Software architecture refers to the structure of the system, which consists of several components regarding a program/ system, the attributes (properties) of those components and the relationship among them.

The software architecture helps the software engineers in the process of analyzing the software design proficiently.

Additionally it also helps the software engineers in decision-making as well as handling risks. Following tasks are implemented by the software architecture :

- Give an insight to all the relevant stakeholders which helps them to communicate with each other.
- Spot the early design decisions having big impact on the software engineering activities such as coding and testing which are followed after the design phase.
- Generates intellectual models regarding the way by which the system is organized into components and the way by which these components communicate with each other.

(D) Modularity

UQ. Explain design concepts : Modularity.
MU - May 16, May 18, 10 Marks

- Modularity can be achieved by the process of dividing the software into components which are uniquely named and addressable. These components are also known as **modules.**
- A complex system (huge application) is partitioned into a set of discrete modules in such a manner that one should be able to develop those modules independent of each other.
- After development is completed, the modules are integrated to form an entire project.
- One has to remember that as the number of modules a system is divided into increases, it becomes easy to handle the software but it also increases the effort required to integrate the modules.
- The process of modularizing a design helps to plan the development in a more efficient way, accommodate changes without difficulty, conduct testing and debugging effectively as well as efficiently, and carry out the maintenance work without adversely affecting the functionality of the software.

Example

- The ubiquitous television set is an example of a system made up of a number of modules – speakers, projection tube, channel buttons, volume buttons, etc.
- Each module has its own defined functionality but when they are put together synergistically, the complete functionalities of a television are realized.

(E) Concurrency

- It is important to utilize the resources efficiently as much as possible. For this purpose multiple tasks must be executed concurrently.
- This aspect makes concurrency one of the important concepts of software design.

- Every system should be designed in such a manner that it should facilitate multiple processes to execute concurrently.
- For example, if the currently executing process is waiting for some resource, the system must be able to execute any other process in the mean time.

(F) Verification

- Verification is described as a mechanism of confirmation by examining and giving evidence that there is no mismatch in design output and the design input specifications.
- Design input can be any physical as well as performance requirement which is used as the basis for the purpose of designing.
- Design output is considered as the result of all the design phases' efforts. The final design output is a basis for device master record.

(G) Aesthetics

- Aesthetics is the philosophical study of beauty and taste.
- Software engineering is a largely creative design discipline, where new designs (both graphical design models as well as code) are created from scratch.
- Compared to art, there are many targets (e.g. desired functionality) and constraints (e.g. desired quality attributes) that the software engineer must adhere to, but these still leave almost infinitely many ways to create a model that fulfills all those requirements.
- In addition, for essentially the same model, there are many ways to represent it. Since models and code must also serve to communicate ideas to other people (or to the same people months or years later), the chosen structure and representation are key attributes for the success of the product.
- It's a well-accepted hypothesis that the **aesthetics** of the artifacts produced during software development are an indication of 'the quality': it is very common to comment on designs as being 'beautiful" or "elegant".
- Such a qualification is used to indicate the apparent quality of the design. Apparently, we make an intuitive connection between aesthetics and quality.

The Design Model

GQ. Discuss about Design model and importance in Design Engineering.

- The design model can be viewed in two different dimensions as illustrated in Fig. 3.5.1. The process dimension indicates the evolution of the design model as design tasks are executed as part of the software process.
- The abstraction dimension represents the level of detail as each element of the analysis model is transformed into a design equivalent and then refined iteratively. Referring to Fig. 3.5.2, the dashed line indicates the boundary between the analysis and design models.
- In some cases, a clear distinction between the analysis and design models is possible. In other cases, the analysis model slowly blends into the design and a clear distinction is less obvious.
- The elements of the design model use many of the same UML diagrams that were used in the analysis model. The difference is that these diagrams are refined and elaborated as part of design; more implementation-specific detail is provided, and architectural structure and style, components that reside within the architecture, and interfaces between the components and with the outside world are all emphasized.

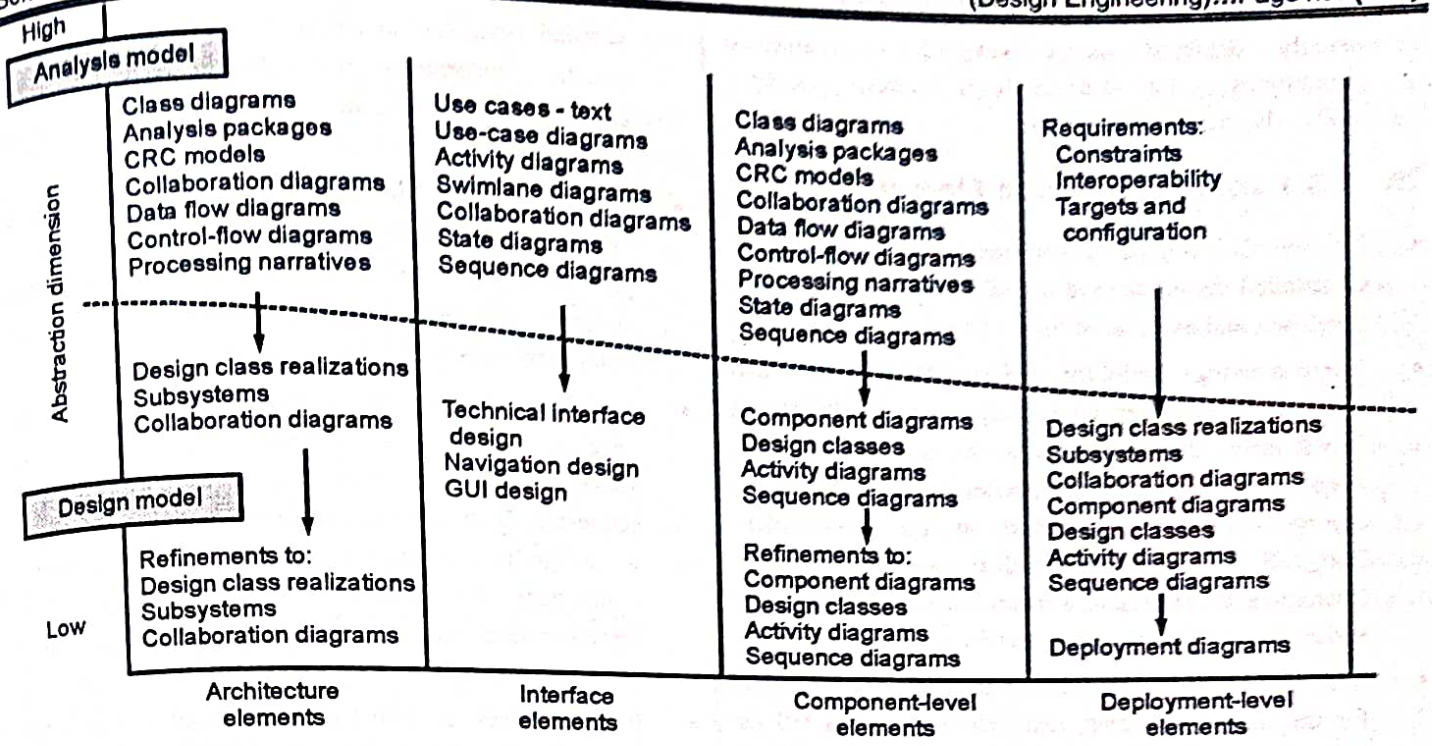


Fig. 3.5.2 : The process dimension

You should note, however, that model elements indicated along the horizontal axis are not always developed in a sequential fashion. In most cases preliminary architectural design sets the stage and is followed by interface design and component-level design, which often occur in parallel. The deployment model is usually delayed until the design has been fully developed.

Data Design Elements

- Like other software engineering activities, data design (sometimes referred to as data architecting) creates a model of data and/or information that is represented at a high level of abstraction (the customer/user's view of data). This data model is then refined into progressively more implementation-specific representations that can be processed by the computer-based system. In many software applications, the architecture of the data will have a profound influence on the architecture of the software that must process it.
- The structure of data has always been an important part of software design. At the program component level, the design of data structures and the associated algorithms required to manipulate them is essential to the creation of high-quality applications.
- At the application level, the translation of a data model (derived as part of requirements engineering) into a database is pivotal to achieving the business objectives of a system.
- At the business level, the collection of information stored in disparate databases and reorganized into a "data warehouse" enables data mining or knowledge discovery that can have an impact on the success of the business itself.

Architectural Design Elements

- The architectural design for software is the equivalent to the floor plan of a house. The floor plan depicts the overall layout of the rooms; their size, shape, and relationship to one another; and the doors and windows that allow movement into and out of the rooms. The floor plan gives us an overall view of the house. Architectural design elements give us an overall view of the software.
- The architectural model is derived from three sources: (1) information about the application domain for the software to be built; (2) specific requirements model elements such as data flow diagrams or analysis classes, their relationships and collaborations for the problem at hand; and (3) the availability of architectural styles . The architectural design element is

usually depicted as a set of interconnected subsystems, often derived from analysis packages within the requirements model.

3.5.1 Interface Design Elements

- The interface design for software is analogous to a set of detailed drawings (and specifications) for the doors, windows, and external utilities of a house.
- These drawings depict the size and shape of doors and windows, the manner in which they operate, the way in which utility connections (e.g., water, electrical, gas, telephone) come into the house and are distributed among the rooms depicted in the floor plan. They tell us where the doorbell is located, whether an intercom is to be used to announce a visitor's presence, and how a security system is to be installed.
- In essence, the detailed drawings (and specifications) for the doors, windows, and external utilities tell us how things and information flow into and out of the house and within the rooms that are part of the floor plan. The interface design elements for software depict information flows into and out of the system and how it is communicated among the components defined as part of the architecture.
- There are three important elements of interface design: (1) the user interface (UI); (2) external interfaces to other systems, devices, networks, or other producers or consumers of information; and (3) internal interfaces between various design components.
- These interface design elements allow the software to communicate externally and enable internal communication and collaboration among the components that populate the software architecture..
- Usability design incorporates aesthetic elements (e.g., layout, color, graphics, interaction mechanisms), ergonomic elements (e.g., information layout and placement, metaphors, UI navigation), and technical elements (e.g., UI patterns, reusable components). In general, the UI is a unique subsystem within the overall application architecture.
- The design of external interfaces requires definitive information about the entity to which information is sent or received. In every case, this information should be collected during requirements engineering and verified once the interface design commences. The design of external interfaces should incorporate error checking and (when necessary) appropriate security features.
- The design of internal interfaces is closely aligned with component-level design. Design realizations of analysis classes represent all operations and the messaging schemes required to enable communication and collaboration between operations in various classes.
- Each message must be designed to accommodate the requisite information transfer and the specific functional requirements of the operation that has been requested. If the classic input-process-output approach to design is chosen, the interface of each software component is designed based on data flow representations and the functionality described in a processing narrative.
- In some cases, an interface is modeled in much the same way as a class. In UML, an interface is defined in the following manner : "An interface is a specifier for the externally-visible [public] operations of a class, component, or other classifier (including subsystems) without specification of internal structure." Stated more simply, an interface is a set of operations that describes some part of the behavior of a class and provides access to these operations.
- For example, the SafeHome security function makes use of a control panel that allows a homeowner to control certain aspects of the security function. In an advanced version of the system, control panel functions may be implemented via a wireless PDA or mobile phone.
- The ControlPanel class (Fig. 3.5.3) provides the behavior associated with a key- pad, and therefore, it must implement the operations readKeyStroke () and decodeKey (). If these operations are to be provided to other classes (in this case, WirelessPDA and MobilePhone), it is useful to define an interface as shown in the Fig. 3.5.3. The interface, named KeyPad, is shown as an <<interface>> stereotype or as a small, labeled circle connected to the class with a line.
- The interface is defined with no attributes and the set of operations that are necessary to achieve the behavior of a keypad. The dashed line with an open triangle at its

end (Fig. 3.5.3) indicates that the Control Panel class provides KeyPad operations as part of its behavior. In UML, this is characterized as a realization. That is, part of the behavior of Control Panel will be implemented by realizing KeyPad operations. These operations will be provided to other classes that access the interface.

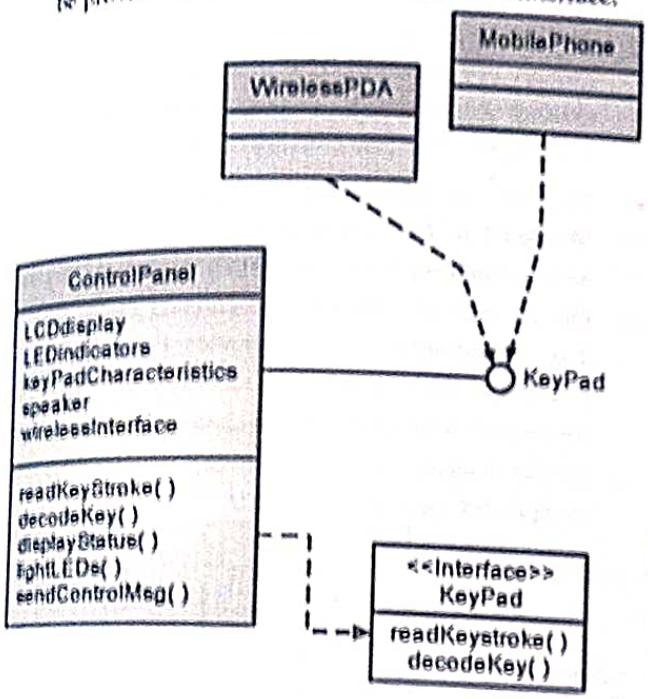


Fig. 3.5.3 : Interface representation for Control-Pane

Component-Level Design Elements

- The component-level design for software is the equivalent to a set of detailed drawings (and specifications) for each room in a house. These drawings depict wiring and plumbing within each room, the location of electrical receptacles and wall switches, faucets, sinks, showers, tubs, drains, cabinets, and closets.
- They also describe the flooring to be used, the moldings to be applied, and every other detail associated with a room. The component-level design for software fully describes the internal detail of each software component.
- To accomplish this, the component-level design defines data structures for all local data objects and algorithmic detail for all processing that occurs within a component and an interface that allows access to all component operations (behaviors). Within the context of object-oriented software engineering, a component is represented in UML diagrammatic form as shown in Fig. 3.5.4. In this Fig. 3.5.4 a component named Sensor Management (part of the SafeHome security function) is represented.
- A dashed arrow connects the component to a class named Sensor that is assigned to it. The Sensor Management component performs all functions associated with SafeHome sensors including monitoring and configuring them.
- The design details of a component can be modeled at many different levels of abstraction. A UML activity diagram can be used to represent processing logic. Detailed procedural flow for a component can be represented using either pseudocode (a programming language) or some other diagrammatic form (e.g., flowchart or box diagram).
- Algorithmic structure follows the rules established for structured programming (i.e., a set of con- strained procedural constructs). Data structures, selected based on the nature of the data objects to be processed, are usually modeled using pseudocode or the programming language to be used for implementation.
- Deployment-Level Design Elements Deployment-level design elements indicate how software functionality and subsystems will be allocated within the physical computing environment that will support.

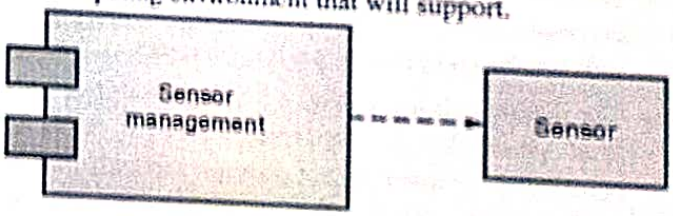


Fig. 3.5.4 : A UML component diagram

- The software. For example, the elements of the SafeHome product are configured to operate within three primary computing environments—a home-based PC, the SafeHome control panel, and a server housed at CPI Corp. (providing Internet-based access to the system).
- During design, a UML deployment diagram is developed and then refined as shown in Fig. 3.5.5. In the figure, three computing environments are shown (in actuality, there would be more including sensors, cam[illegible]nd others).

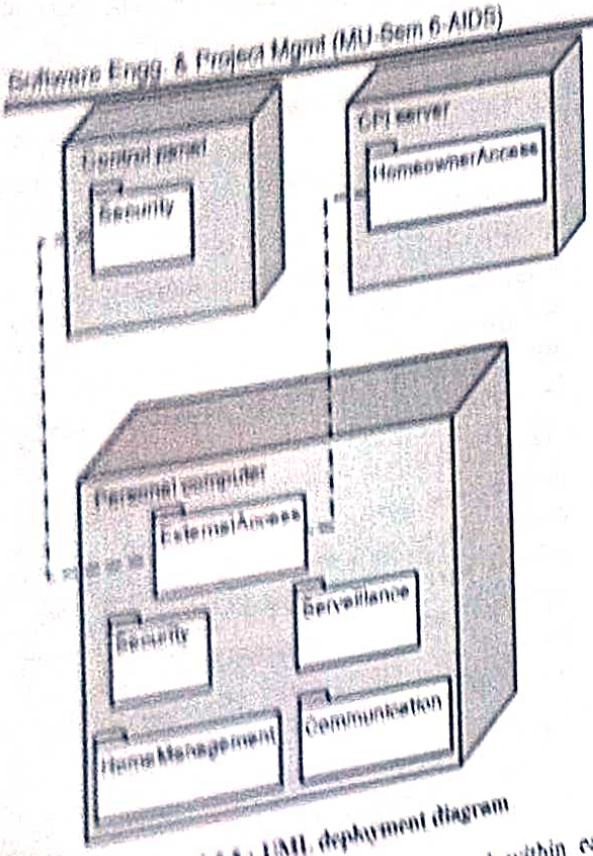


Fig. 3.5.5 : UML deployment diagram

- The sub-systems (functionality) housed within each computing element are indicated. For example, the personal computer houses subsystems that implement security, surveillance, home management, and communication features. In addition, an external access subsystem has been designed to manage all attempts to access the SafeHome system from an external source. Each subsystem would be elaborated to indicate the component that is implemented.
- The diagram shown in Fig. 3.5.5 is descriptive form. This means that the deployment diagram shows the computing environment but does not explicitly indicate configuration details. For example, the "personal computer" is not further identified. It could be a Mac or a Windows based PC, a Sun workstation, or a Linux box.
- These details are provided when the deployment diagram is revisited in instance form during the later stages of design or as construction begins. Each instance of the deployment (a specific network hardware configuration) is identified.

Pattern-based Software Design

- Pattern-based design is relatively new in the field of software development, indeed [illegible] technology has only [illegible] design. Even pattern-based design [illegible] standard centuries [illegible] years of [illegible]

configurations provide design elements that are used to engineer automobiles, aircraft, machine tools, and robots.

- Applying pattern based design to software development promises the same benefits to software as it does to industrial technology: predictability, risk mitigation, and increased productivity. Throughout the design process, you should look for every opportunity to apply existing design patterns (when they meet the needs of the design) rather than creating new ones.
- The role of pattern-based design in all of this is illustrated in Fig. 3.5.6. A software designer begins with a requirements model (either explicit or implied) that presents an abstract representation of the system. The requirements model describes the problem set, establishes the context, and identifies the system of forces that hold sway. It may imply the design in an abstract manner, but the requirements model does little to represent the design explicitly.

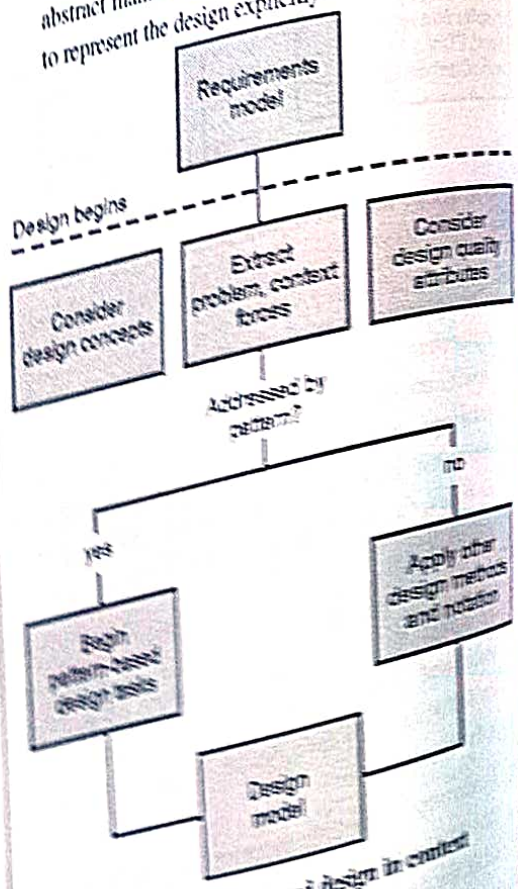


Fig. 3.5.6 : Pattern-based design in context

Thinking in Patterns

- Good design begins by considering context—the [illegible] picture. As context is evaluated, you extract a broad

of problems that must be solved. Some of these problems will be global in nature, while others will address specific features and functions of the software. All will be affected by a system of forces that will influence the nature of the solution that is proposed. following approach that enables a designer to think in patterns:

1. Be sure you understand the big picture—the context in which the software to be built resides. The requirements model should communicate this to you.
2. Examining the big picture, extract the patterns that are present at that level of abstraction.
3. Begin your design with "big picture" patterns that establish a context or skeleton for further design work.
4. "Work inward from the context" [Sha05] looking for patterns at lower levels of abstraction that contribute to the design solution.
5. Repeat steps 1 to 4 until the complete design is fleshed out.
6. Refine the design by adapting each pattern to the specifics of the software you're trying to build. It's important to note that patterns are not independent entities. Design patterns that are present at a high level of abstraction will invariably influence the manner in which other patterns are applied at lower levels of abstraction. In addition, patterns often collaborate with one another.

   The implication—when you select an architectural pattern, it may very well influence the component-level design patterns you choose. Likewise, when you select a specific interface design pattern, you are some- times forced to use other patterns that collaborate with it. To illustrate, consider the SafeHomeAssured.com WebApp. If you consider the big picture, the WebApp must address a number of fundamental problems such as:

- How to provide information about SafeHome products and services
- How to sell SafeHome products and services to customers
- How to establish Internet-based monitoring and control of an installed security system

Each of these fundamental problems can be further refined into a set of sub problems. For example How to sell via the Internet implies an E-commerce pattern that itself implies a large number of patterns at lower levels of abstraction. The E-commerce pattern (likely, an architectural pattern) implies mechanisms for setting up a customer account, displaying the products to be sold, selecting products for purchase, and so forth. Hence, if you think in patterns, it is important to determine whether a pattern for setting up an account exists. If Set Up Account is available as a viable pattern for the problem context, it may collaborate with other patterns such as Build Input Form, Manage Forms Input, and Validate- Forms Entry. Each of these patterns delineates problems to be solved and solutions that may be applied

Design Tasks

GQ. Elaborate design tasks in pattern based design.

The following design tasks are applied when a pattern-based design philosophy is used:

1. Examine the requirements model and develop a problem hierarchy. Describe each problem and subproblem by isolating the problem, the context, and the system of forces that apply. Work from broad problems (high level of abstraction) to smaller subproblems (at lower levels of abstraction).
2. Determine if a reliable pattern language has been developed for the problem domain. The SafeHome software team would look for a pattern language developed specifically for home security products. If that level of pattern language specificity could not be found, the team would partition the SafeHome software problem into a series of generic problem domains (e.g., digital device monitoring problems, user interface problems, digital video management problems) and search for appropriate pattern languages.
3. Beginning with a broad problem, determine whether one or more architectural patterns is available for it. If an architectural pattern is available, be certain to examine all collaborating patterns. If the pattern is appropriate, adapt the design solution proposed and

build a design model element that adequately represents it.

4. Using the collaborations provided for the architectural pattern, examine subsystem or component-level problems and search for appropriate patterns to address them. It may be necessary to search through other pattern repositories as well as the list of patterns that corresponds to the architectural solution. If an appropriate pattern is found, adapt the design solution proposed and build a design model element that adequately represents it. Be certain to apply step 7.
5. Repeat steps 2 through 5 until all broad problems have been addressed. The implication is to begin with the big picture and elaborate to solve problems at increasingly more detailed levels.
6. If user interface design problems have been isolated (this is almost always the case), search the many user interface design pattern repositories for appropriate patterns. Proceed in a manner similar to steps 3, 4, and 5.
7. Regardless of its level of abstraction, if a pattern language and/or patterns repository or individual pattern shows promise, compare the problem to be solved against the existing pattern(s) presented. Be certain to examine context and forces to ensure that the pattern does, in fact, provide a solution that is amenable to the problem.
8. Be certain to refine the design as it is derived from patterns using design quality criteria as a guide.

Common Design Mistakes

GQ. What is common Design mistakes ?

- Pattern-based design can make you a better software designer, but it is not apanacea. Like all design methods, you must begin with first principles, emphasizing software quality fundamentals and ensuring that the design does, in fact, address the needs expressed by the requirements model.
- A number of common mistakes occur when pattern-based design is used. In some cases, not enough time has been spent to understand the underlying problem and its context and forces, and as a consequence, you select a pattern that looks right but is inappropriate for the solution required.
- Once the wrong pattern is selected, you refuse to see your error and force-fit the pattern. In other cases, the problem has forces that are not considered by the pattern you've chosen, resulting in a poor or erroneous fit. Sometimes a pattern is applied too literally and the required adaptations for your problem space are not implemented.
- Can these mistakes be avoided ? In most cases the answer is "yes". Every good designer looks for a second opinion and welcomes review of her/him work.

Syllabus Topic : Architectural Design

3.6 ARCHITECTURAL DESIGN

GQ. Explain architectural design. (10 Marks)

- Requirements of the software should be transformed into an architecture that describes the software's top-level structure and recognizes its components.
- This can be achieved by using the architectural design which acts as an initial 'blueprint' from which software can be developed.
- IEEE described architectural design as a process of defining a set of hardware and software components and their interfaces to set up the framework for the development of a computer system. This framework is established by observing the software requirements document and designing a model for providing implementation details.
- These details are used to state the components of the system together with their inputs, outputs, functions, and the interaction between them.
- Architectural design can be represented with the help of following models :
  1. **Structural model :** Demonstrate architecture as an ordered collection of program components.
  2. **Dynamic model :** This model illustrates the behavioural aspect of the software architecture and shows how the structure or system configuration changes as the function changes because of the change in the external environment.

3. **Process model :** This model concentrates on the design of the business or technical process, which must be implemented in the system.
4. **Functional model :** This model signifies the functional hierarchy of a system.
5. **Framework model :** This model ties to recognize repeatable architectural design patterns which are found in same types of applications. This will results in increase in the level of abstraction.

3.6.1 Functions of Architectural Design

1. It describe an abstraction level at which the designers can state the functional and performance behavior of the system.
2. It estimates all top-level designs.
3. It acts as a guideline for improving the system by illustrating those features of the system that can be changed easily while maintaining the system integrity.
4. It develops and documents top-level design for the external and internal interfaces.
5. It develops initial versions of user documentation.

3.6.2 Architectural Design Document

- The result of architectural design process is Architectural Design Document (ADD).
- This document contains a number of graphical representations that encompasses software models along with related descriptive text.
- The software models consist of static model, interface model, relationship model, and dynamic process model which illustrates how the system is arranged into a process during run-time.

Architectural design document provides the developers a solution to the problem defined in the Software Requirements Specification (SRS).

In addition to ADD, other outputs of the architectural design are given below :

1. A range of reports including audit report, progress report, and configuration status accounts report.
2. Different plans for detailed design phase, which consist of following :
   (i) Software verification and validation plan,
   (ii) Software configuration management plan,
   (iii) Software quality assurance plan,
   (iv) Software project management plan.

3.6.3 Architectural Style

UQ. Explain different Architectural styles.
MU - Dec. 16, 5 Marks

- Architectural styles describe a set of systems which are internally linked with each other and share structural and semantic properties.
- In short, the purpose of using architectural styles is to set up a structure for all the components present in a system.
- If an existing architecture is to be re-engineered, then burden of an architectural style results in basic changes in the structure of the system.
- By applying certain restriction on the design space, we can make different style-specific analysis from an architectural style.
- A computer-based system (software is part of this system) demonstrates one of the several available architectural styles.
- Every architectural style defines a system type that includes the following :
  1. Computational components like clients, server, filter, and database to execute the required system function.
  2. A collection of connectors like procedure call, events broadcast, database protocols, and pipes to offers communication between the computational components.
  3. Constraints to describe integration of components to form a system.
  4. A semantic model, which allow the software designer so as to recognize the characteristics regarding the system as a whole by the way of studying the characteristics of its components.

- Following are the common architectural styles :

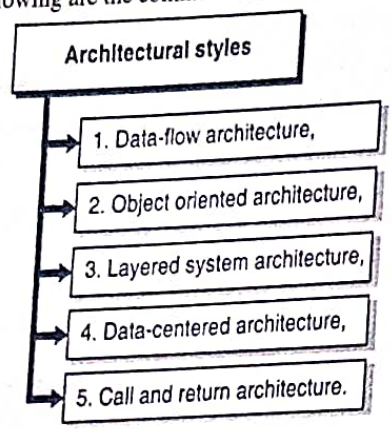
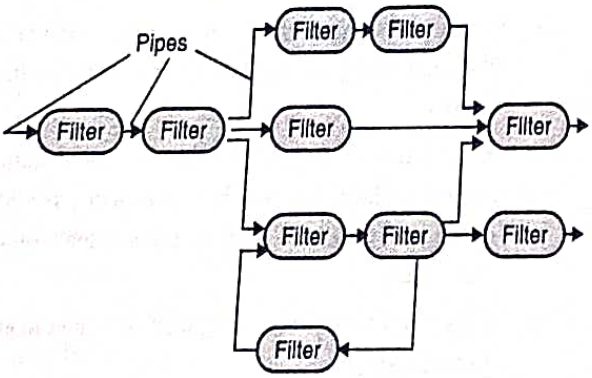


Fig. 3.6.1 : Architectural styles

1. Data-flow Architecture

- The purpose of data-flow architecture is to accept some inputs and transforms it into the required outputs by applying a sequence of transformations.
- Each component called as filter transforms the data and sends this transformed data to other filters for additional processing with the help of the connector called as pipe.
- Every filter operates as an independent entity, that is, it is not focused on the filter which is generating or consuming the data.
- A pipe is a unidirectional channel which transfers the data received on one end to the other end.
- It does not change the data in anyway; it just delivers the data to the filter on the receiver end.



(a) Pipes and filters

(b) Batch sequential

Fig. 3.6.2 : Data flow architecture

- Most of the times, the data-flow architecture reprodu a batch sequential system.
- In this system, a batch of data is accepted as input a then a sequence of sequential filters are applied transform this data.
- Example of this architecture is UNIX shell programs.
- In these programs, UNIX processes act as filters an the file system by which UNIX processe communicates act as pipes.

Advantages of data-flow architecture

1. It supports maintainability, reusability and modifiability.
2. Concurrent execution is supported by this style.

Disadvantages of data-flow architecture

1. It frequently degenerates to batch sequential system.
2. The application which needs more user interactions is not supported by data-flow architecture.
3. It is not easy to coordinate two different but related streams.

2. Object-oriented Architecture

- In this architectural style, the components of a system wrap data and operations into single unit, which are used to change the data.
- In this style, components are represented as *objects* and they communicate with each other through methods (connectors).

Characteristics of object-oriented architecture

1. Objects preserve the integrity of the system.
2. An object is not aware of other objects' representation.

Advantages of object-oriented architecture

1. It allows designers to divide a problem into a group of independent objects.
2. The implementation detail of objects is known to other object so that they can be changed without affecting other objects.

3. Layered System Architecture

- In layered architecture, every layer is responsible for performing a well-defined set of operations.
- These layers are organized in a hierarchical manner, where every layer is built upon one below another.

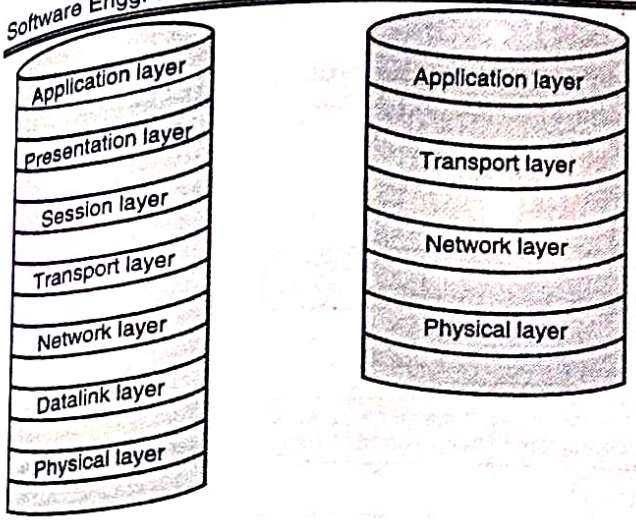


Fig. 3.6.3 : OSI and Internet Protocol Suite

- Each layer offers a set of services to the layer which is located above it and acts as a client to the layer which is located below it.
- The interaction among layers is provided through protocols that describe a set of rules to be followed throughout the interaction.
- One common example of this architectural style is OSI-ISO (Open Systems Interconnection-International Organization for Standardization) communication system.

4. Data-centered Architecture

- A data-centered architecture has two unique components :
  1. Central data structure
  2. Collection of client software
- A central data structure is also known as data store.
- The data store such as database or a file demonstrates the current state of the data where as the client software performs various operations such as add, delete, update, etc. on the data which is stored in the data store.
- Sometimes the data store permits the client software to access the data independent of any changes or the actions of other client software.
- In this architectural style, new components related to clients can be added and existing components can be changed easily which does not require change in other clients.
- This is because client components work without depending upon any other components.
- A variation of this architectural style is blackboard system in which the data store is transformed into a blackboard that informs the client software when the data is modified.
- Additionally, the information can be transferred between the clients with the help of blackboard components.

Advantages of the data-centered architecture

1. Data repository is not dependant of the clients.
2. Clients work independent of each other.
3. Adding new clients can be easy.
4. Modification can be very easy.
5. It accomplishes data integration in component-based development with the help of blackboard.

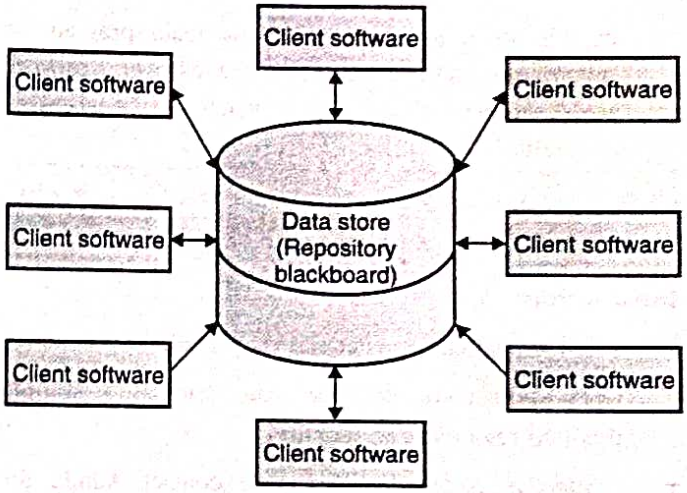


Fig. 3.6.4 : Data-centered architecture

5. Call and Return Architecture

- A call and return architecture allows software designers to get such a program structure where modification is easy.
- This architecture style contains following two sub-styles.

A. Main program/subprogram architecture

In this style, function is divided into a controlled hierarchy where the main program contains and executes a number program components, where as the contained components may in turn executes other components.

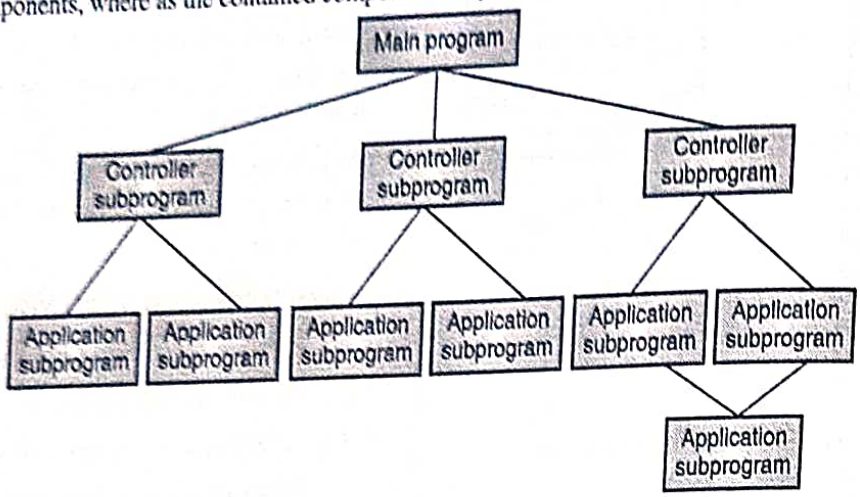


Fig. 3.6.5 : Main program/subprogram architecture

B. Remote procedure call architecture

In this style, components of the main program or subprogram architecture are distributed over a network across number of computers which can be accessed remotely.

UQ. Explain architectural design for e-commerce System. **MU - May 15, Dec. 15, 10 Marks**

Description

- Any member can register and view available products
- Only registered member can purchase multiple products regardless of quantity.
- ContactUs page is available to contact Admin for queries
- There are three roles available : Visitor, User and Admin.
  - Visitor can view available products.
  - User can view and purchase products.
  - An Admin has some extra privilege including all privileges of visitor and user.
  - Admin can add products, edit product information and remove product.
  - Admin can add user, edit user information and can remove user.
  - Admin can ship order to user based on order placed by sending confirmation mail.

Using the code

- Attach the database in your "SQL Server Management Studio Express".
- Run the application on Microsoft Visual Studio as web site.
- Locate the database.

MasterPage details

OnlineShopping Master Page (Similar MasterPage for Visitor, User and Admin)

Web Pages details

- Home Page
- AboutUs Page
- Clothing Page
- OrderUs Page
- ContactUs Page
- Admin Page
- Login Page
- Register Page
- Track

Project Detail

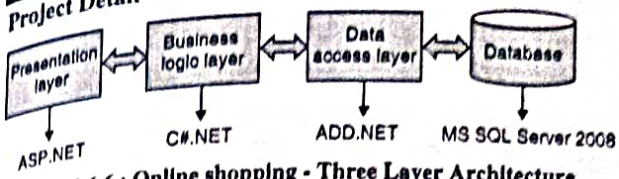


Fig. 3.6.6 : Online shopping - Three Layer Architecture

System requirements

Use-case diagram

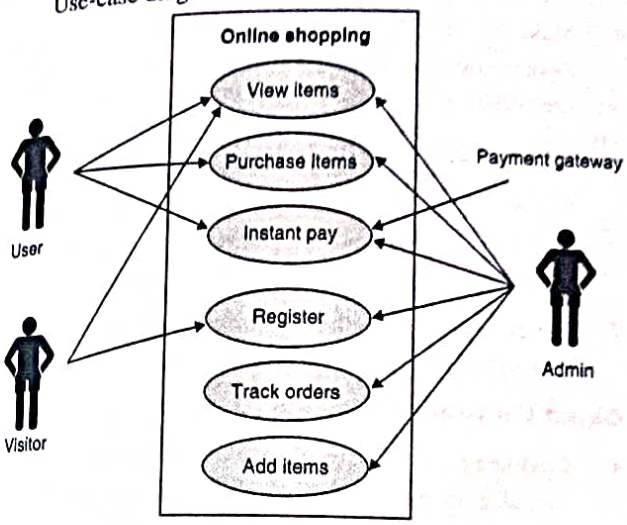


Fig. 3.6.7

Syllabus Topic : Component Level Design

3.7 COMPONENT LEVEL DESIGN

UQ. Write note on Component Based Development.

MU - May 16, 10 Marks

UQ. Explain component level model.

MU - Dec. 17, 5 Marks

- The main function of component level design is to define data structures, algorithms, interface characteristics as well as communication mechanisms for all the present software components which are identified in the phase of architectural design.
- The phase of component level design is expected when the data, architectural, and interface designs are completed.
- The software is represented by the component-level design in such a manner which lets the designer to review it for correctness as well as consistency prior to its built.
- The work product generated is considered as a design for all of the software components. It is represented with the help of graphical, tabular, or text-based notation.
- Design walkthroughs are carried out to observe accuracy of the data transformation or control transformation which has been allocated to all of the components in the earlier design steps.

Component Definitions

- A **component** is a modular, deployable, replaceable part of a system which encapsulates implementation and exposes a set of interfaces.
- As per object-oriented view a component contains a set of collaborating classes
- As per traditional view, a component (or module) resides in the software and serves one of three roles as follows :
  - *Control components* coordinate invocation of all other problem domain components
  - *Problem domain components* implement the functionality needed to the customer
  - *Infrastructure components* implement the functionality required to support the processing needed in a domain application.
- Process-Related view focus on building systems out of existing components which are selected from a catalog of reusable components as a way of populating the architecture.

Object-Oriented Component Design

- Put emphasis on describing the domain oriented analysis classes as well as the definition of infrastructure classes
- Comprehensive information of class attributes, operations, and interfaces is necessary before beginning construction activities.

Principles for Designing Class-based Components

- ***Open-Closed Principle (OCP)*** : Extensions should be allowed in the class but not modification.
- ***Liskov Substitution Principle (LSP)*** : It should be possible to substitute subclasses for their base classes.
- ***Dependency Inversion Principle (DIP)*** : Depend on abstractions, do not depend on concretions.

A. Main program/subprogram architecture

In this style, function is divided into a controlled hierarchy where the main program contains and executes a number program components, where as the contained components may in turn executes other components.

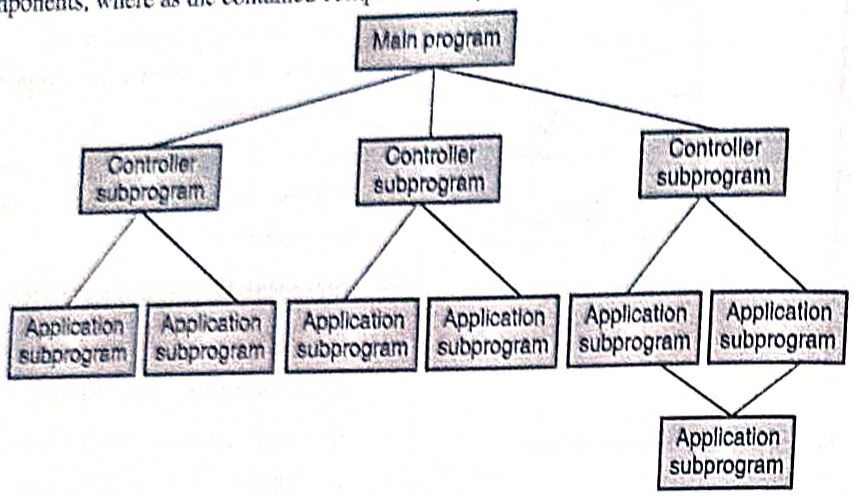


Fig. 3.6.5 : Main program/subprogram architecture

B. Remote procedure call architecture

In this style, components of the main program or subprogram architecture are distributed over a network across number of computers which can be accessed remotely.

UQ. Explain architectural design for e-commerce System. **MU - May 15, Dec. 15, 10 Marks**

Description

- Any member can register and view available products
- Only registered member can purchase multiple products regardless of quantity.
- ContactUs page is available to contact Admin for queries
- There are three roles available : Visitor, User and Admin.
  - Visitor can view available products.
  - User can view and purchase products.
  - An Admin has some extra privilege including all privileges of visitor and user.
  - Admin can add products, edit product information and remove product.
  - Admin can add user, edit user information and can remove user.
  - Admin can ship order to user based on order placed by sending confirmation mail.

Using the code

- Attach the database in your "SQL Server Management Studio Express".
- Run the application on Microsoft Visual Studio as web site.
- Locate the database.

MasterPage details

OnlineShopping Master Page (Similar MasterPage for Visitor, User and Admin)

Web Pages details

- Home Page
- AboutUs Page
- Clothing Page
- OrderUs Page
- ContactUs Page
- Admin Page
- Login Page
- Register Page
- Track

Project Detail

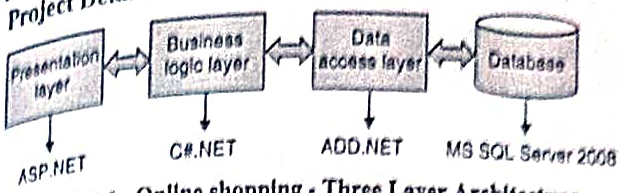


Fig. 3.6.6 : Online shopping - Three Layer Architecture

System requirements

Use-case diagram

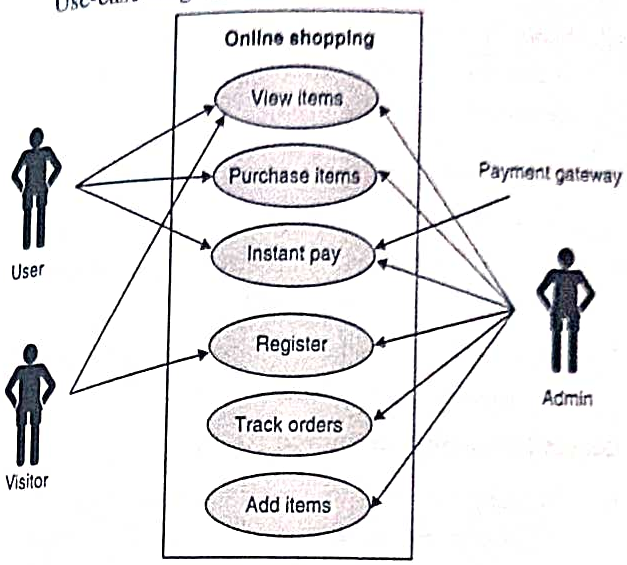


Fig. 3.6.7

Syllabus Topic : Component Level Design

3.7 COMPONENT LEVEL DESIGN

UQ. Write note on Component Based Development.

MU - May 16, 10 Marks

UQ. Explain component level model.

MU - Dec. 17, 5 Marks

- The main function of component level design is to define data structures, algorithms, interface characteristics as well as communication mechanisms for all the present software components which are identified in the phase of architectural design.
- The phase of component level design is expected when the data, architectural, and interface designs are completed.
- The software is represented by the component-level design in such a manner which lets the designer to review it for correctness as well as consistency prior to its built.
- The work product generated is considered as a design for all of the software components. It is represented with the help of graphical, tabular, or text-based notation.
- Design walkthroughs are carried out to observe accuracy of the data transformation or control transformation which has been allocated to all of the components in the earlier design steps.

Component Definitions

- A **component** is a modular, deployable, replaceable part of a system which encapsulates implementation and exposes a set of interfaces.
- As per object-oriented view a component contains a set of collaborating classes
- As per traditional view, a component (or module) resides in the software and serves one of three roles as follows :
  - *Control components* coordinate invocation of all other problem domain components
  - *Problem domain components* implement the functionality needed to the customer
  - *Infrastructure components* implement the functionality required to support the processing needed in a domain application.
- Process-Related view focus on building systems out of existing components which are selected from a catalog of reusable components as a way of populating the architecture.

Object-Oriented Component Design

- Put emphasis on describing the domain oriented analysis classes as well as the definition of infrastructure classes
- Comprehensive information of class attributes, operations, and interfaces is necessary before beginning construction activities.

Principles for Designing Class-based Components

- ***Open-Closed Principle (OCP)*** : Extensions should be allowed in the class but not modification.
- ***Liskov Substitution Principle (LSP)*** : It should be possible to substitute subclasses for their base classes.
- ***Dependency Inversion Principle (DIP)*** : Depend on abstractions, do not depend on concretions.

- ***Interface Segregation Principle (ISP)*** **:** Multiple client oriented interfaces are considered as better compared to one general purpose interface.
- ***Release Reuse Equivalency Principle (REP)*** **:** The granule of reuse is the granule of release.
- ***Common Closure Principle (CCP)*** **:** Classes that change together belong together.
- ***Common Reuse Principle (CRP)*** **:** Classes that can't be used together should not be grouped together.

Component-Level Design Guidelines

1. **Components**

- Establish naming conventions in the phase of architectural modeling.
- Names of Architectural components should be meaningful for the stakeholders.
- Names of Infrastructure components must have implementation specific meanings.
- The nature of components can be identified with the help of UML stereotypes.

2. **Interfaces**

- Use of lollipop representation is considered as better as compared to formal UML box and arrow notation.
- For the purpose of consistency, the flow of interfaces should be from the left-hand side of the component box.
- Display only those interfaces which are relevant to the component under construction.

3. **Dependencies**

- To improve readability it should be better to have the model dependencies from left to right and inheritance from bottom (child classes) to top (parent classes).
- Interfaces must be used to represent component interdependencies rather than component to component dependencies.

Conducting Component Level Design

1. Identify all of the respective design classes which are corresponding to the problem domain.
2. Identify all of the respective design classes which are corresponding to the infrastructure domain.
3. Elaborate all of the design classes which are not acquired as reusable components.
   (a) Specify details regarding the message when there is collaboration of classes or components.
   (b) Identify suitable interfaces for all the components.
   (c) Elaborate attributes and define data types and data structures which are necessary for their implementation.
   (d) Describe processing flow of all the operations thoroughly.
4. Make the Identification of persistent data sources such as databases and files and recognize the classes which are necessary to manage them.
5. Develop and elaborate behavioral representations for all the classes or components.
6. Elaborate deployment diagrams for the purpose of providing additional implementation details.
7. Factor all of the component-level diagram representations and consider alternatives.

Object Constraint Language (OCL)

- Complements UML by permitting a developer to use formal grammar as well as syntax for the purpose of constructing unambiguous statements regarding design model element.
- Parts of OCL language statements :
  1. A context which indicates the restricted situation in which the statement is considered as valid.
  2. A property which represents few characteristics of the context.
  3. An operation which manipulates or qualifies a property.
  4. Keywords which are used for the purpose of specifying conditional expressions.

Conventional or Structured Component Design

- Each block of code has a single entry at the top.
- Each block of code has a single exit at the bottom.
- Only three control structures are required : sequence, condition (if-then-else), and repetition (looping).
- Reduces program complexity by enhancing readability, testability, and maintainability.

Design Notation

- *Flowcharts* : Arrows for flow of control, diamonds for decisions, rectangles for processes.
- *Decision table* : Subsets of system conditions and actions are associated with each other to define the rules for processing inputs and events.
- *Program Design Language (PDL)* : Structured English or pseudocode used to describe processing details.

Program Design Language Characteristics

- Predetermined syntax with keywords provided for the purpose of representing all structured constructs, data declarations as well as module definitions.
- Free syntax of natural language for the purpose of describing processing features.
- Data declaration facilities for simple as well complex data structures.
- Subprogram definition as well as invocation facilities.

Design Notation Assessment Criteria

- *Modularity* : Notation provide support for development of modular software.
- *Overall simplicity* : Easy to learn, easy to use, easy to write.
- *Ease of editing* : Easy to make changes in design representation whenever required.
- *Machine readability* : Notation can be input directly into a computer-based development system.
- *Maintainability* : Maintenance of the configuration in general engage maintenance of the procedural design representation.
- *Structure enforcement* : Enforces the use of structured programming constructs.
- *Automatic processing* : Allows the designer to verify the correctness and quality of the design.
- *Data representation* : Ability to represent local as well as global data directly.
- *Logic verification* : Automatic logic verification improves testing competence.
- Easily converted to program source code (makes code generation quicker).

Syllabus Topic : System Level Design

3.8 SYSTEM LEVEL DESIGN

GQ. Write note on system level design. **(10 Marks)**

- System design is process of defining architecture, components, interfaces and data elements of a system to satisfy all system requirements.
- The main goal of system design is to provide essential information and data for system implementation.
- It is an iterative process which converts requirements of system into its blue print.
- At the end of system design process, design model and specification report is prepared.
- During system design process designer must go for following decisions :
  - (i) Make selection of appropriate method for data store and retrieval.
  - (ii) Write design specifications for the developers.
  - (iii) Assign priority to each activity in a process.
  - (iv) Determine system interfaces.
  - (v) Define and manage boundary conditions for system etc.
- During system development process, system is divided into various components These system component are called as **sub-systems.**

3.8.1 Concurrency and Sub-system Allocation

- Concurrency means integration of technology and processes which enables computer program or system to perform many activities at the same time.
- If two or more independent activities occurs in system simultaneously, (without interacting with each other) then they are inherently concurrent activities.

Sub-system allocation

- In concurrency, subsystem allocation can be done in two ways :
  - (i) Allocate subsystems to independent processors

(ii) Provide support to concurrency through operating system features by allocating subsystems to same processor.

- If two subsystems are inherently concurrent then their allocation can be done using second way i.e. by using operating system features.
- Main goal of system design is to identify concurrent and inherently concurrent activities of system.
- While making subsystem allocation, the system designer must perform following tasks :

(i) Identify resource need of subsystems.

(ii) Make selection of appropriate hardware and software implementation for subsystems.

(iii) Allocate subsystems to processors in such a way that satisfies performance need of system and minimizes inter processor communication.

(iv) Identify connectivity between physical units that implement the subsystem

3.8.2 Task Management Components

- Each task should be assigned to a hardware unit for its execution.
- Before assigning task, following basic things are identified :

(a) Characteristics of task are defined depending on how task is initiated.

(b) A coordinator task and its associated objects are identified.

(c) The coordinator task is integrated with other tasks.

(d) Priority and criticality of each task is identified.

- In general task template includes :

Task name : Name of object

Description : Purpose of object

Priority : Priority of task

Services : Activities performed by task

Coordinates by : The way in which object is invoked

Communicates via :

Input and output values used by task

- Tasks are assigned to multiple processors to improve the productivity of system.
- While assigning task, system designer should select appropriate arrangement and form of physical connections.
- System designer must decide which subsystem to be implemented in hardware and which to be implemented in software.

3.8.3 Data Management Components

- The components which must consider while formulating data plan are called data management components.
- The data management components concerns with two distinct areas :

(a) Managing data of specific application and

(b) Creating infrastructure for storage and retrieval of system objects.

- In general DBMS is used to create infrastructure for storage and retrieval of data since it supports reliability, concurrency control and data independence.
- There are various data management components as follows :

(a) Where and how to store and retrieve data.

(b) Why to store data.

(c) Data privacy and copyrights

(d) How to document data

(e) File formats and data types

(f) Storage, security, backups etc.

- These components must be considered while designing data plan of system.
- Store and access of data is main function of transaction manager in database system.

3.8.4 Inter Sub-system Communication

- Each system is internally composed of many subsystems.
- All these subsystems must be integrated to one another in an efficient and effective way for successful execution of whole system.
- Communication between various subsystems occurs by sending and receiving messages to each other.
- An interface defines how one subsystem can interact with another subsystem.

- To specify interface for subsystem following steps are used :
  - (i) Identify all requests made by subsystem and define these requests using appropriate interfaces.
  - (ii) Identify responsibilities of each system interface.
  - (iii) Design collaboration diagram for representing interaction between subsystems.
- Problem with subsystem communication can be : Performance and exception handling, output format, flow of control, user interface etc.

3.8.5 Object Design Process

- Each activity in a system involves various objects.
- Object design means identifying all required objects of system, defining them, determining their attributes and functions and defining interaction between various objects of system.
- Object diagram gives representation of objects in terms of attributes and functions and show how they interact with each other.
- Objects of system can be either active object, passive object, environment object, class object or virtual mode object.
- Objects which are executing simultaneously are called as active objects.
- Objects which execute only sequentially are called as passive objects.
- Class objects describe general packages of system and virtual mode objects shows nodes of distributed system.
- The object design process describes all objects of system in detail. It defines interfaces, algorithms and operations for each object.
- Objects acts as skeleton for activity of system.
- At the end of object design process, object optimization is also done.

Syllabus Topic : User Interface Design

3.9 USER INTERFACE DESIGN

GQ. What is user interface design process ? **(10 Marks)**

GQ. Write short note on User Interface Design. **(5 Marks)**

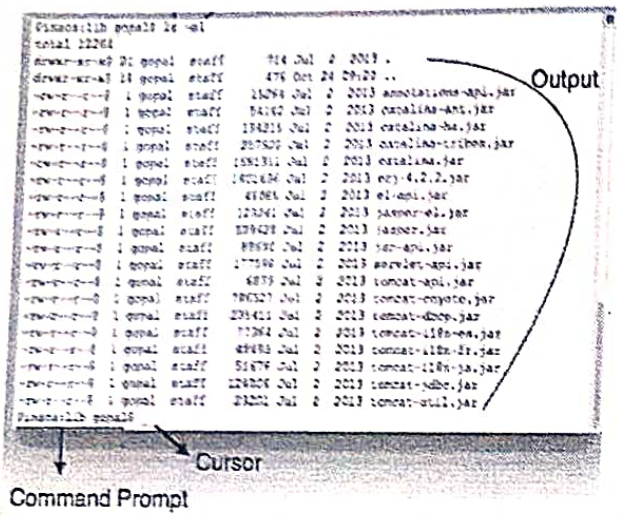
- User interface is considered as a front-end application view which is interacted by the user so as to use the software.
- User is able to manipulate as well as control the software and hardware with the help of user interface.
- Now-a-day, user interface appears in each and every place where there is existence of digital technology, right from desktops, smart phones, cars, music players, airplanes, ships etc.
- User interface is an integral component of software and is designed in such a manner that it should be able to provide the user insight of the software.
- UI offers a fundamental platform for the communication of user and computer.
- The form of UI can be graphical, text-based, audio-video based. It is usually depends upon the underlying platform (hardware and software combination).
- The popularity of software depends upon following aspects :
  - Attractive
  - Simple to use
  - Responsive in short time
  - Clear to understand
  - Consistent on all interfacing screens
- UI is broadly divided into two categories :
  1. Command Line Interface
  2. Graphical User Interface

3.9.1 Command Line Interface (CLI)

- CLI was considered as a great tool of communication with computers in previous days.
- Now-a-days also CLI is the first choice of number of technical users as well as programmers. CLI is considered as minimum interface which software can provide to its users.
- A command prompt is provided by the CLI which is a place where the user writes instructions and feeds to the system.
- Here there is need for the user to remember the syntax of command and its use. In previous days CLI were not programmed to handle the user errors effectively.

- A command is considered as a text-based reference to set of instructions, which should be executed by the respective system.
- There are several methods such as macros, scripts which make it simple for the user to operate.
- Very less amount of computer resource is used by the CLI in comparison with GUI.

CLI Elements



There are following elements in a text-based command line interface :

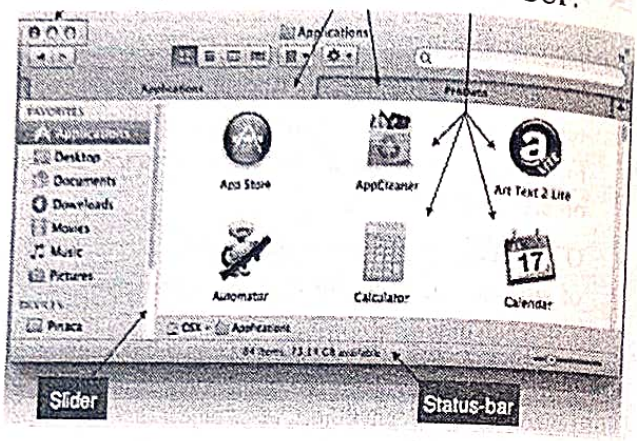
- **Command Prompt :** It is text-based notifier which usually displays the context where the user is working. Software system generates it.
- **Cursor :** It is a small horizontal line or a vertical bar with the height of line. It is used to represent position of character at the time of typing. Cursor is usefully found in blinking state. It moves when user writes or deletes text.
- **Command :** A command is nothing but an executable instruction. There may be one or more parameters to it. Output of command is displayed inline on the screen. After displaying output, command prompt is set to the next line.

3.9.2 Graphical User Interface (GUI)

- Graphical User Interface offers graphical means to the use for interacting with the system. GUI can be combination of both hardware and software. Using GUI, user is able to interpret the software.
- In general GUI is considered as more resource consuming as compared to CLI. With the help of latest advanced technology, the programmers and designers are able to create complex GUI designs which perform with better efficiency, accuracy and speed.

3.9.2(A) GUI Elements

- GUI provides several components for the purpose of interacting with software or hardware.
- All of these graphical components give ways to interact with the system.
- Following are some important elements of GUI :



- **Window :** It is the area in which contents of application are displayed.
- To represent file structure, the Window contents can be displayed in the form of icons or lists.
- In an exploring window, the navigation is very easy for user.
- Windows provide facility to minimize, resize or maximize. It is possible to move them anywhere on the screen. A window may contain another window (child window) of the same application inside it.
  - **Tabs -** When an application executes more than one instances of itself, then those instances appear on the screen as separate windows.
  - **Tabbed Document Interface -** Provides the facility to open more than one document in the same window. This interface is used to view preference panel in application. Now-a-days this feature is used by almost all web-browsers.
  - **Menu -** Menu is considered as an array of standard commands, combined together and

located at a prominent place (mostly top) inside the application window. It is also possible to program the menu to appear or hide on mouse clicks.

- **Icon** - An icon is small picture which is used to represent an application. These icons are clicked or double clicked to open the application window.
- **Cursor** - In GUI cursors are used to represent interacting devices like mouse, touch pad, digital pen, etc.

- In almost real-time, the instructions given by hardware are followed by the cursor. Cursors which are also called as pointers are used to select menus, windows and other application features.

3.9.2(B) Application Specific GUI Components

A GUI of an application contains following elements :

- **Application Window :** Most of the application windows use the constructs provided by underlying OS but some have their own customized windows to contain the contents of application.
- **Dialogue Box :** Dialog box is a child window which has message for the user and request for some action to be taken. For Example: A dialog box is displayed by an application to get confirmation from user to delete a file.
- **Text-Box :** Used to accept input from user in text format.
- **Buttons :** Used to submit inputs to the application.
- **Radio-button :** Displays list of options for selection. Only single radio button can be selected among all offered.
- **Check-box :** Its functionality is same as of the radio button but it allows multi selection.
- **List-box :** Provides list of available items using single control. Multiple items can be selected.

3.9.2(C) User Interface Design Activities

- Several activities are carried out for designing user interface. The process of GUI design and implementation is same as of the SDLC (Software Development Life Cycle).
- For GUI implementation there are number of models available such as Waterfall, Iterative or Spiral Model.
- Following GUI specific steps should be fulfilled by the model used for GUI design and development.

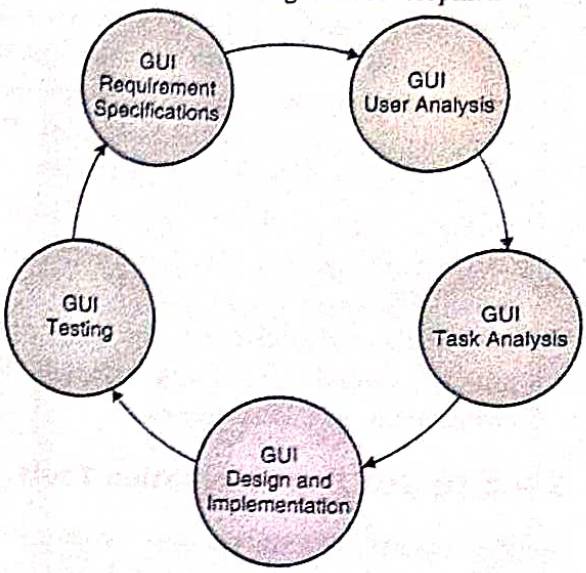


Fig. 3.9.1 : GUI Steps

- **GUI Requirement Gathering :** The designers need the list of list of all functional as well as non-functional requirements regarding GUI. These requirements are accepted from users and their existing software solution.
- **User Analysis :** The designer do the study regarding who will actually use the software GUI. The target audience is an important aspect since the design details change according to the knowledge as well as competency level of the end user.

- If user looks to be techno savvy then advanced and complex GUI can be developed.
- For a new user, more informative data is included regarding how-to of software.
  - **Task Analysis :** Designers have an important responsibility to analyze the task which is to be done by the respective software solution.
- Here in case of GUI, it is not important how it will be done. It is possible to represent the task in hierarchical manner considering one major task at the top and dividing it further into smaller sub-tasks.
- For GUI presentation, goals are provided by the tasks.

- In the software the flow of GUI contents is decided by the flow of information among sub-tasks.
  - **GUI Design and Implementation :** Designers after getting complete information regarding requirements, tasks and user environment, design the respective GUI and implement the design into code and integrate the GUI with currently working or any dummy software in the background. Then self-testing is implemented by the developers.
  - **Testing :** There are several ways to implement the GUI testing. Organization has options such as in-house inspection, straight involvement of end users and prior release of beta version. Important aspects verified in testing are usability, compatibility, user acceptance etc.

3.9.2(D) GUI Implementation Tools

- There are number of tools available for designers to create entire GUI on a single mouse click. Out of them some tools can be easily integrated into the software environment.
- A huge set of GUI controls is provided by GUI implementation tools.
- The code is modified by the designers for software customization.
- There are diverse segments of GUI tools in accordance with their use and platform.

Example

- Mobile GUI, Computer GUI, Touch-Screen GUI etc.
- Following are some tools which are used to build GUI :
  - FLUID
  - AppInventor (Android)
  - LucidChart
  - Wavemaker
  - Visual Studio

3.9.2(E) User Interface Golden Rules

- Designers need to solve problems every day, and finding the right solution involves in-depth research and carefully planned testing.
- The following rules are know[illegible] golden rules for GUI design, described by Sh[illegible] and Plaisant in their book (Designing the User Interface).

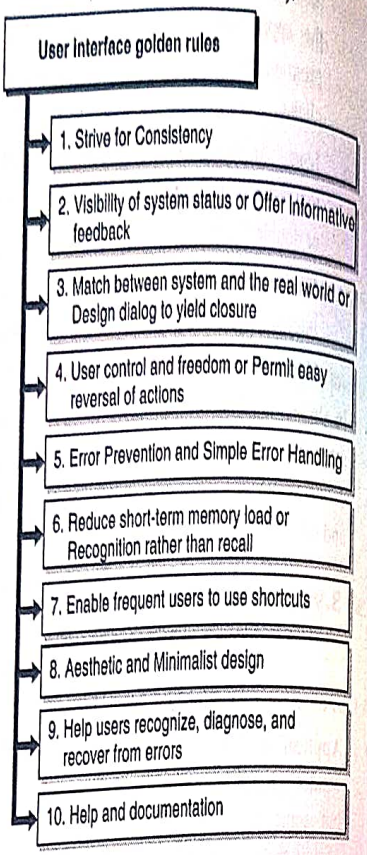


Fig. 3.9.2 : User interface golden rules

1. Strive for Consistency

- Users should not have to wonder whether different words, situations, or actions mean the same thing. Do not confuse your user keep *words and actions* consistent.
- Use "The Principle of Least Surprise".
- **Example :** Car Climate Control

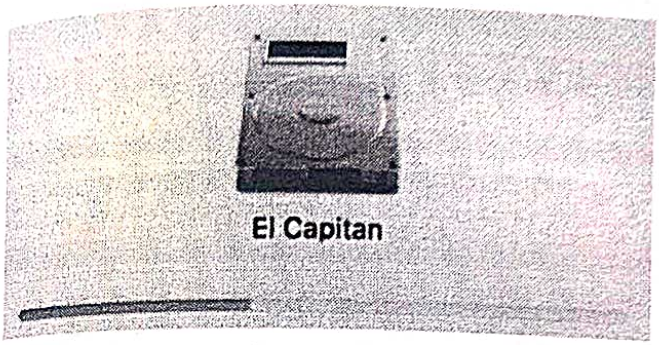
- In other words, use all elements across your application consistently. For example a certain style of button should always do the same thing, or navigation should function logically, going deeper in hierarchy.
- Consistency of :
  - Workflow / Processes

- Functionality
- Appearance
- Terminology

2. Visibility of system status or Offer informative feedback

- The system should always keep users informed about what is going on. Through appropriate feedback in a *reasonable* time. Don't keep the users guessing tell the user what's happening.
- **Example :** OS Installation status

  Preparing to install. Your computer will restart automatically.



About 25 seconds remaining.

- The user wants to be in control, and trust that the system behaves as expected. It could be even said that users don't like surprises.
- For frequent and minor actions, the response can be modest, while for infrequent and major actions, the response should be more substantial.

Feedback

- Relevant
- Fits importance and urgency
- Comprehensible and meaningful
- Within appropriate context (time and place)

3. Match between system and the real world or Design dialog to yield closure

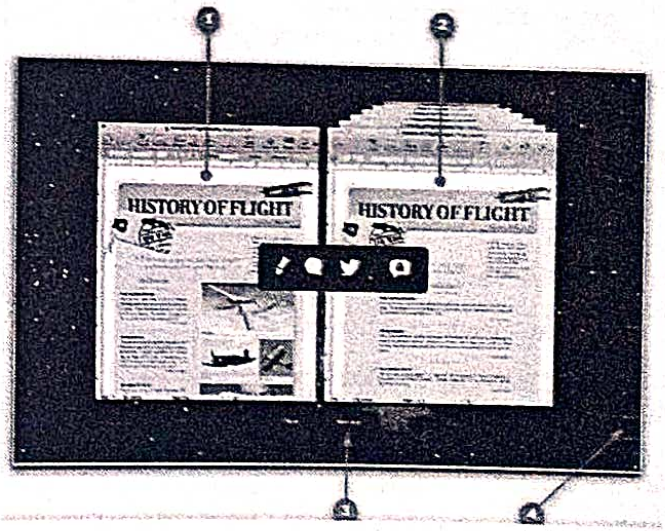
- Again, the less the users have to guess the better. The system should speak the users' language (use words, phrases and concepts familiar to the user), rather than special system terms.
- **Example :** Payment proceeding (by Ramakrishna).
- Sequence of actions should be organized into groups with a beginning, middle and end. When a process is finished, remember to display a notification message.
- Let the user know that he has done all that's needed.

Design

- Grouping of actions.
- Explicit completion of an action.
- Well-defined options for the next step.

4. User control and freedom or Permit easy reversal of actions

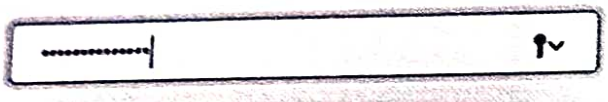
- Shneiderman puts it nicely "This feature relieves anxiety, since the user knows that errors can be undone; it thus encourages exploration of unfamiliar options."
- **Example :** Document History



- In applications, this refers to the undo and redo functionality. Clearly mark an "*emergency exit*" to leave the unwanted state without having to go through an extended dialogue.
- Reversal of actions
  - No interference with workflow
  - More freedom for the user
  - Single-action undo and action history

5. Error Prevention and Simple Error Handling

- Users hate errors, and even more so hate the feeling that they themselves have done something wrong. Either *eliminate* error-prone conditions or *check for them* and notify users about that before they commit to the action.
- **Example :** Password Enter.



Your password must have:

- 8 or more characters
- Upper & lowercase letters
- At least one number

Strength: strong

Avoid passwords that are easy to guess or used with other websites.

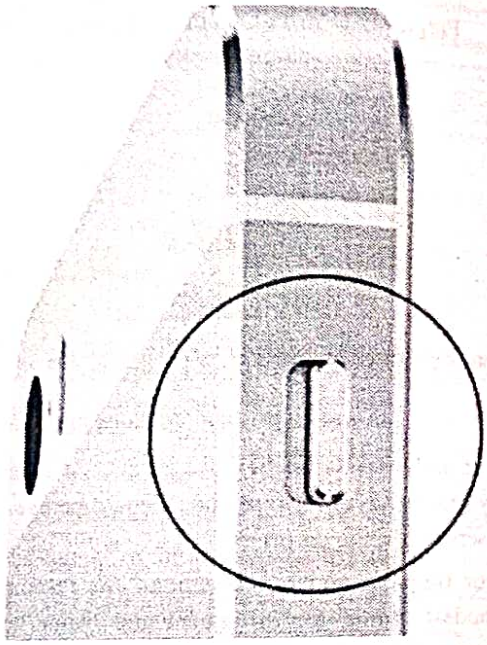
- As much as possible, design the system so the user cannot make a serious error.
- If an error is made, the system should be able to detect the error and offer a simple, comprehensive mechanism for handling the error.

Error Prevention

- Error prevention over error correction
- Automatic detection of errors
- Clear error notifications
- Hints for solving the problem

6. Reduce short-term memory load or Recogni rather than recall

- As Nielsen says, recognizing something is easier remembering it. Minimize the user's memory loa making objects, actions, and options *available*.
- The user should not have to remember informa from one part of the dialogue to another. Instruct should be *visible*.
- **Example :** Ring/Silent switch



- Use iconography and other visual aids such as then coloring and consistent placement of items to help returning users find the functionalities.

Reduce memory load

- App has a clear structure
- "Recognition over recall"
- Implicit help
- Visual aids

7. Enable frequent users to use shortcuts

- Allow users to *tailor* (manipulate and personali frequent actions.
- Abbreviations, function keys, hidden commands, a macro facilities are very helpful to an expert user.

- **Example :** Click Commands and Shortcuts

New Window	⌘N
New Private Window	⇧⌘N
New Tab	⌘T
Open File...	⌘O
Open Location...	⌘L

Shortcuts

- Keyboard shortcuts
- "Power User" features
- Action automation

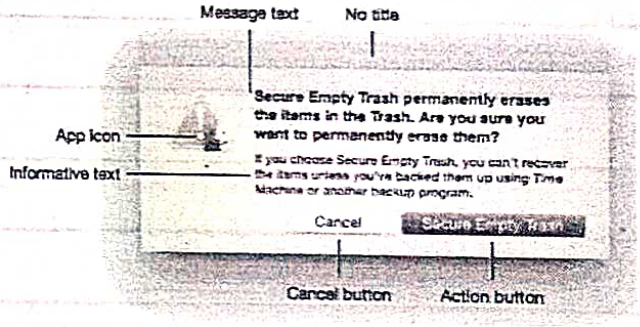
8. **Aesthetic and Minimalist design**

- Minimalist doesn't mean limited. All information should be *valuable* and *relevant*.
- **Example :** Simple Interface

- Simplify interfaces by removing unnecessary elements or content that does not support user tasks.

9. **Help users recognize, diagnose, and recover from errors**

- Error messages should be expressed in *plain language* (don't use system language to describe what the system is doing), precisely *indicate* the problem, and constructively *suggest* a solution.
- **Example :** Plain Message



- Tell the user clearly and plainly what's happening both on the background and when they perform an action.

10. **Help and documentation**

- Even though it is better if the interface can be used without documentation, it is necessary to provide help and documentation.
- Any help information should be *easy to search, focused* on the user's task, list of *concrete steps*, and *not too large*.

Chapter Ends...

Note

MODULE IV

CHAPTER 4

Software Risk, Configuration Management

Syllabus

Risk Identification, Risk Assessment, Risk Projection, RMMM Software Configuration management, SCM repositories, SCM process Software Quality Assurance Task and Plan, Metrics, Software Reliability, Formal Technical Review (FTR), Walkthrough.

4.1 Risk Identification 4-2
4.2 Risk Assessment 4-3
4.2.1 Qualitative Risk Analysis 4-3
4.2.2 Quantitative Risk Analysis 4-3
4.3 Risk Planning and Projection 4-3
4.4 RMMM 4-4
4.5 Evaluating Risks to the Schedule 4-5
4.6 Software Configuration Management 4-6
4.6.1 Benefits of Software Configuration Management 4-6
4.6.2 SCM Disciplines 4-6
4.6.3 SCM Repositories 4-8
4.6.4 SCM Process 4-9
4.6.4(A) Configuration Identification 4-10
4.6.4(B) Change Control (Configuration Control) 4-11
UQ. Explain change control activities in SCM. MU - May 15, Dec. 15, May 18, 5 Marks 4-11
UQ. Explain change management process. MU - Dec. 16, 5 Marks 4-11
4.6.4(C) Version Control 4-12
UQ. Explain version control activities in SCM. MU - May 15, Dec. 15, May 18, 5 Marks 4-12
4.6.4(D) Configuration Audit 4-13
4.6.4(E) Status Reporting 4-13
• **Chapter Ends** 4-18

4.1 RISK IDENTIFICATION

- Risk is an integral part of an IT projects and therefore risk identification at the earliest becomes all the more critical. Risk identification is an iterative process, wherein the project manager and his team are always on the lookout for risk that may sneak into the project or may have already sneaked in.
- Risk identification needs to be done throughout the project lifecycle, because as the project develops new risk is likely to emanate and pose a threat to the project. An idealistic situation would be one in which all the likely risks could be identified before the execution of the project but the reality is that new risk emanate as the project progresses and hence the project manager and his team have to always be alert and identify risk and nub them in their bud.

i. To identify the risks are always the biggest and the toughest task in project management.

ii. The risk can be categorized on the basis of the size of the project, and the software development, the risk identified is then modified.

iii. The risks which create a business impact due to the constraints imposed by management or the marketplace during the development phase till the deployment phase.

iv. As the customer played the critical role in the project development being the end user, the various characteristics related to the customer for communication are very important.

v. The exact requirement gathering and the analysis to be done for the development of the project at the Software Development Life Cycle has to be determined to avoid the risks.

vi. Good quality tools should be used and made available for the developers.

vii. New advances in the area of technology and the project development has to be studied for a perfect time lined software development.

viii. The major risks are associated if the staff and the team are not as per the required project.

Risk Components and Drivers

The risk components are defined and identified on the criteria of the performance, support, cost and the schedule of the project :

1. **Performance risk :** If the specifications don't meet the final requirements of the project.
2. **Support risk :** The support the application developer provides after deployment of the project
3. **Cost risk :** To maintain the time and the cost of the project.
4. **Schedule risk :** The schedule and the parameters of deliverable should be maintained

There are several tools available that can help the project manager in identifying risks.

- **Project Documents :** Project documents is the first place to start looking for project risks. The project scope statement, WBS, project plan, resource requirement, etc will help identify many risks associated with the project.
- **SWOT Analysis :** Similar to an organizational SWOT, the project is examined in terms of its strengths, weaknesses, opportunities and threats and areas are identified where the project could fail and where the project could improve. SWOT is a great technique when it comes to resources and requirements not used before.
- **Brainstorming :** A brainstorming session is one in which the project team sits together and thinks up with as many risks as possible. A brainstorming session is one in which the participants can let their imagination go wild, there is no restriction on the number of risks to be identified on the other hand the team is encouraged to anything that remotely resembles a risk.
- **Delphi Technique :** Although brainstorming sessions do manage to identify the likely risks, a few participants may shy from identifying risks on account of personal reasons. Therefore, to overcome these limitations the Delphi Technique is adopted. The Delphi technique comprises of anonymous surveys conducted amongst team members to build consensus on risk events.
- **Assumption analysis :** Every project is based on certain assumptions. However, these assumptions need

to be tested to ensure their validity. Assumptions not tested have the potential to become risk in the future. Therefore, these assumptions need to be tested, researched and the results confirmed as to how they would be affecting the project. An assumption log should form a part of the project documents.

- **Root cause analysis :** This analysis examines an effect that the project is experiencing and then goes to the root of the effect to identify its cause.

All the identified risks need to be entered into a risk register. The risk register will contain a log of all the risks and its impact on the project.

4.2 RISK ASSESSMENT

- The next step in the risk management plan is to analyze the identified risks. The identified risks need to be analysed to determine the probability of their occurrence and the quantum of impact they would be having on the project.
- The procedure for analysis is to conduct the qualitative analysis and once the risk qualifies then a quantitative analysis is carried out.

4.2.1 Qualitative Risk Analysis

- Qualitative risk analysis does not involve a detailed analysis of the risk. It is fast and subjective approach to risk analysis. Usually the probability and quantum of impact of the risk is based on past experience or merely gut feeling.
- The purpose of qualitative risk analysis approach is to decide whether the risk qualifies for further analysis.

Impact →	1	2	3	4	5
Probability ↓	Negligible	Minor	Moderate	Significant	Severe
(81-100)%	Low Risk	Moderate risk	High Risk	Extreme Risk	Extreme Risk
(61-80)%	Minimum Risk	Low Risk	Moderate Risk	High Risk	Extreme Risk
(41-60)%	Minimum Risk	Low Risk	Moderate Risk	High Risk	High Risk
(21-40)%	Minimum Risk	Low Risk	Low Risk	Moderate Risk	High Risk
(1-20)%	Minimum Risk	Minimum Risk	Low Risk	Moderate Risk	High Risk

- Qualitative risk analysis is carried out using a risk matrix also called as a probability-impact matrix.
- A risk is "rated" for its Probability and Impact on a scale to understand where on the Risk Matrix it lies. Which Risks in the process move forward into the Risk Management process will depend on the project and people. Some are by nature more risk tolerant than others.
- For example, we can have a project where the team agrees that any risk that is in the moderate, high or extreme cell can move forward in the Risk Management process.
- The rest remain in the watch list and are "accepted". Another project could have different criteria.

4.2.2 Quantitative Risk Analysis

- Once the risk has passed the qualitative test it is time to take it more seriously and research the risk events to gauge their true potential and probability of impact.
- The risk in quantitative risk analysis is subject to detailed analysis that takes time and has a budget associated with it.
- The probability of the risk happening and quantum of its impact is gauged in a controlled environment.
- A ranking of risks based on their probability and quantum impact is created.

Syllabus Topic : Risk Projection

4.3 RISK PLANNING AND PROJECTION

- Once the risk has been identified and analyzed the project manager can decide on the manner in which the risk can be dealt with. The response to risk depends on the type of risk. However, all risk should be documented along with the chosen response and its outcome in the risk register for future reference.
- The response to risk depends on its type i.e. negative risk or positive risk. The three most common responses to negative risk are:
  - **Avoidance :** The first natural response to any risk to find ways to avoid it. This would mean creating workarounds, adjusting project objectives, or

changing the project schedule, in short whatever required to avoid the risk.

- **Transference :** The risk is transferred to a third party that is in a better position to manage the risk. Thus, instead of trying to manage the risk in-house the risk is transferred to a third party.
- **Mitigation :** Here the project manager is left with no alternative but to tackle the risk in-house. This would mean spending extra time and money to reduce the probability and impact of the risk.

- The three most common responses to positive risks are:
  - **Exploitation :** The project manager tries to gain maximum benefits from the positive risk. The idea here to utilize the risk to maximum.
  - **Enhancement :** The project manager creates an environment that is ideal for the positive risk to happen.
  - **Sharing :** The positive risk is shared by the project team with another entity to gain maximum benefits from the risk. The project team may not be capable of leveraging maximum benefits from the risk and hence feels that by bringing in a third party it may be able to derive maximum.
- In addition to the above given responses there is another response that is suitable for both positive as well negative risk and this is the response of Acceptance. Positive as well as negative risks with very low impact should be accepted as it is.
- The steps to determine the overall consequences of a risk :
  - The project component which can cause the risk needs to be identified.
  - Using Fig. 4.3.1, the impact for each component has to be identified.
  - The overall risk exposure, RE, is determined using the following relationship RE = P x C
  - Where P is the probability of occurrence for a risk, and C is the cost to the project should the risk occur.

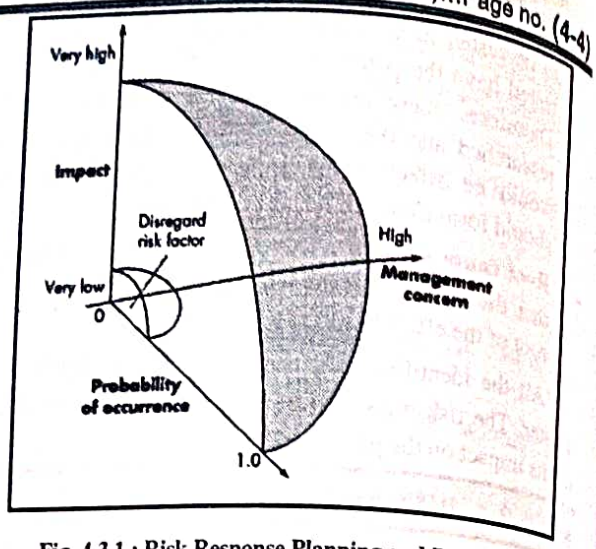


Fig. 4.3.1 : Risk Response Planning and Projection

- For example, assume that the software team defines a project risk in the following manner:
  - Risk Identification. Only 60 integrated into the application. The remaining functionality will have to be custom developed.
  - Risk Probability. 70% (likely).
  - Risk Impact. 65 reusable software components were planned. If only 60 percent can be used, 22 components would have to be developed from scratch (in addition to other custom software that has been scheduled for development). Since the average component is100 LOC and local data indicate that the software engineering cost for each LOC is Rs160,the overall cost (impact) to develop the components would be 22 x 100 x 160 = Rs. 3,52,000.
  - Risk Exposure. RE = 0.70 x Rs. 352000 ~ Rs. 246400.

Syllabus Topic : RMMM

4.4 RMMM

An effective strategy for dealing with risk must consider three issues

1. Risk mitigation (i.e., avoidance)
2. Risk monitoring
3. Risk management and contingency planning

Need for RMMM

- Mitigate all the requirements and the causes that are under our control before the start of the project.
- After the start of the project, the deliverables have to be done on the deadlines by the PERT.
- The development of various groups has to be done by proper and timely communications between the teams.
- The various documentations have to be prepared at the time of the deployment of the project.
- The peer review, formal reviews and walkthroughs to be conducted for the project.
- During risk monitoring, the project manager monitors factors that may provide an indication of whether a risk is becoming more or less likely to identify its intensity.
- Risk management and its removal plan to be prepared and has to come to the reality of development.
- RMMM steps add to an extra project cost during development and deployment.
- Large projects may have identified 40 – 50 risks
- Risk is identified with the various factors related to software projects.
- Risk can be developed after the deployment of the project also.

4.5 EVALUATING RISKS TO THE SCHEDULE

Schedule risk is the risk that the project takes longer than scheduled. It can lead to cost risks, as longer projects always cost more, and to performance risk, if the project is completed too late to perform its intended tasks fully. Apart from the cost estimation and resource allocation used in CPM, most of the techniques used in quantitative cost risk analysis are different from those used in schedule risk analysis.

The earliest technique used for schedule risk analysis was the Gantt chart, developed by Henry Gantt in 1917. We have studied the Gantt Chart in the previous chapter and hence there is no need for any further explanation other than that the Gantt chart gives a graphical summary of the progress of a number of project activities by listing each activity vertically on a sheet of paper, representing the start and duration of each task by a horizontal line and then representing the current time by a vertical line. This makes it easy to see where each activity should be and to show its current status.

- Many tasks require that prior tasks are completed before they can be initiated, but unfortunately, Gantt charts are not a good method of showing the interrelationship between tasks, so computers must be used to set up and maintain the network of tasks. One commonly-used technique is Program Evaluation Review Technique (PERT) which uses a detailed diagram of all anticipated tasks in a project, organised into a network to represent the dependence of each task on those that must precede it.
- PERT can be used to analyse the tasks involved in completing a project, especially the duration of each task, and identify the minimum time needed to complete the total project. PERT makes it possible to schedule a project without knowing the precise details and durations of all the activities.
- The Critical Path Method (CPM) is a similar project planning and management technique which also uses a network representation. Earlier versions did not try to estimate probability distributions for task durations, making it easier to derive the critical path, ie the set of tasks that determined the final project length. Various enhancements were made to CPM to allow alternative resource allocations to be explored, within specified cost constraints.
- The increasing availability of computing power allowed led to the inclusion of probability distributions for task durations in CPM. This allowed Monte Carlo Simulation to be substituted for the PERT assumptions. Monte Carlo simulations are a type of algorithm used to simulate the behaviour of systems. They are described as stochastic or nondeterministic because they are based on the use of random numbers. A stochastic model is a method of estimating probability distributions of potential outcomes by allowing for random variation in one or more inputs.

4.6 SOFTWARE CONFIGURATION MANAGEMENT

GQ. Write note on SCM with benefits and disciplines. **(10 Marks)**

- Configuration Management (CM) is the process of identifying and defining the configuration items in a system, controlling the release and change of these items throughout the system lifecycle, recording and reporting the status of configuration items and change requests, and verifying the completeness and correctness of configuration items.
- Configuration Management is practiced in one form or another as part of any software engineering project where several individuals or organizations have to coordinate their activities.
- While the basic disciplines of Configuration Management are common to both hardware and software engineering projects, there are some differences in emphasis due to the nature of software products.
- Software Configuration Management (SCM) is a system for managing the evolution of software products, both during the initial stages of development and during all stages of maintenance.
- A software product encompasses the complete set of computer programs, procedures, and associated documentation and data designated for delivery to end user.
- All supporting software used in development, even though not part of the software product, should also be controlled by SCM.

4.6.1 Benefits of Software Configuration Management

- SCM provides significant benefits to all projects regardless of size, scope, and complexity.
- Some of the most common benefits experienced by project teams applying the SCM disciplines described here are possible because the SCM system :
- Provides a snapshot of dynamically changing software to help :
  - Make decisions based on an instantaneous view
  - Determine status at module and system levels
  - Make better decisions about the future of a software project.
- Tracks concurrent development of modules or components of the overall system to :
  - Prevent different developers from making changes to the same module at the same time,
  - Allow for the overall system progress to be faster,
  - Provide visibility of the entire software project to all developers.
- Organizes all concurrently developing code and associated documentation to :
  - Save overall project time
  - Focus each phase of the software product development to be organized and executed in a documented, prescribed manner.

4.6.2 SCM Disciplines

- SCM disciplines are used to simultaneously coordinate configurations of many different representations of the software product (e.g., source code, relocatable code, executable code, and documentation).
- In practice, the ways in which SCM disciplines are performed will be different for the various kinds of software being developed (e.g., commercial, embedded, original equipment manufacturer).
- The degree of formal documentation required within and across different lifecycle management phases (e.g., research, product development, operations, and maintenance) will also vary.
- The use of interactive software development environments extends the concepts of SCM to managing evolutionary changes that occur during the routine generation of specifications, designs, and implementation of code, as well as to the more rigidly documented and controlled baselines defined during development and system maintenance.
- The effectiveness of the SCM system increases in proportion to the degree that its disciplines are an explicit part of the normal day-to-day activities of those involved in the software development and maintenance efforts.

- Fig. 4.6.1 presents a diagram showing how the SCM disciplines interact at the highest level.
- The Software Configuration Management Plan (SCM Plan) provides information on the requirements and procedures necessary for the configuration management activities of the software engineering project.

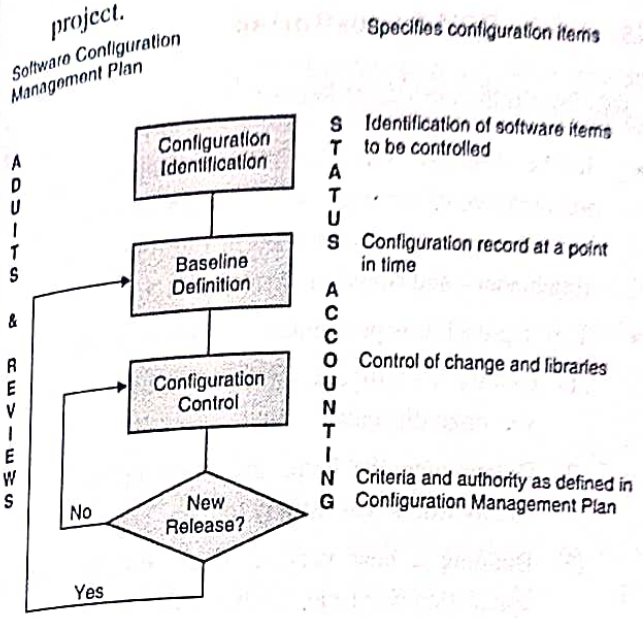


Fig. 4.6.1 : High-Level View of Software Configuration Management

- It specifies who will be responsible for accomplishing the planned activities, what activities will be performed, when the activities will be performed in coordination with the project schedule, and what tools and human resources will be required to perform the activities.
- Consideration should also be given to the overall system configuration.
- The first step in the SCM system is to identify the Configuration Items to be controlled and, as the project progresses, to identify the structure of the intermediate or complete software products.
- Elements to be controlled will normally include the software code and associated documentation.
- Documents could include requirements specifications, designs, user guides, test suites, test results, and user lists.
- Baselines are defined and created in accordance with the SCM Plan and the phase of the project's lifecycle.
- A baseline is a specification or product that has been formally reviewed and agreed upon, that serves as the basis for further development, and can only be changed through formal change control procedures.
- A baseline is like a snapshot of the aggregate of software components as they exist at a given point in time.
- During each phase of the software lifecycle, configuration items that are completed and accepted by the approval authority become the baseline for that phase.
- Any addition, alteration, or deletion to the approved baseline is believed a change, and is subject to change control.
- Baselines, plus the approved changes from those baselines, constitute the current configuration identification. Each of the baselines established during a software lifecycle controls subsequent software development.
- The following are typical configuration baselines that are established during the life of a software project.
  - **Functional Baseline :** Established by the acceptance or approval of the software or system specification. This baseline typically corresponds to the completion of the software requirement review.
  - **Allocated Baseline :** Established by approval of the software requirements specification. This baseline typically corresponds to the completion of the software specification review.
  - **Developmental Baseline :** Established by the approval of the technical documentation that defines the functional and detailed design (including documentation of interfaces and databases for the computer software). Normally, this baseline corresponds to the time frame spanning the preliminary design review and the critical design review.
  - **Product Baseline :** Established by approval of the product specification following completion of the last Formal Functional Configuration Audit (FCA).

- Once configuration items have been identified and baselined, configuration control provides a system to initiate, review, approve, and implement proposed changes.
- Each engineering change or problem report that is initiated against a formally identified configuration item is evaluated to determine its necessity and impact.
- Approved changes are implemented, testing is performed to verify that the change was correctly implemented and that no unexpected side effects have occurred as a result of the change, and the affected documentation is updated and reviewed.
- The discipline of Status Accounting collects information about all configuration management activities.
- The status account should be capable of providing data on the status and history of any configuration item.
- The information maintained in Status Accounting should enable the rebuild of any previous baseline.
- Audits and reviews are conducted to verify that the software product matches the configuration item descriptions in the specifications and documents, and that the package being reviewed is complete.
- Any anomalies found during audits should be corrected and the root cause of the problem identified must be corrected to ensure that the problem does not resurface.
- Generally, there should be a Physical Configuration Audit (PCA) and the Functional Configuration Audit (FCA) of configuration items prior to the release of a software product baseline or an updated version of a product baseline.
- The PCA is performed to determine if all items identified as being part of the configuration are present in the product baseline.
- The audit must also establish that the correct version and revision of each part are included in the product baseline and that they correspond to information contained in the baseline's configuration status report.
- The FCA is performed on each software configuration item to determine that it satisfies the functions defined in the specifications or contracts for which it was developed.
- Appendix C provides a checklist of some of the activities commonly associated with the basic SCM disciplines.
- This list can be modified to include additional activities that are appropriate to the size and complexity of a specific software engineering project.

4.6.3 SCM Repositories

GQ. Explain the term "SCM Repositories". (10 Marks)

- In the first few years of software engineering, the maintenance of software configuration items was done through paper documents sited in file folders or three-ring binders, and stored in metal cabinets.
- This approach was problematic for many reasons :
  (1) Getting a configuration item whenever required was often difficult,
  (2) Determining the items that were updated, when and by whom was usually tough,
  (3) Building a new version of an already existing application was time wasting and error prone, and
  (4) Narrating thorough or complex relationships among the respective configuration items was virtually not possible.
- Today, SCIs (Software Configuration Items) are maintained in a project database or repository.
- Webster's Dictionary explains the word repository as "anything or person thought of as a center of accumulation or storage."
- In early days of software engineering, the repository was nothing but indeed a person (the developer) who had to memorize the location of entire data relevant to the software project, who had to recall the data that was not noted and reconstruct information that had been lost.
- Although this approach of using an individual as "the center for accumulation and storage" does not act properly.
- Now-a-days, the repository is considered as a "thing" (a database) which works as the center for both accumulation as well as storage regarding software engineering information.

- The role of the individual (the software developer) is to communicate with the repository through tools which are integrated with it.

The Role of the Repository

- The SCM repository is the group of mechanisms and data structures which enable a software team to deal with modifications in an efficient manner.
- It offers the apparent functionality of a modern DBMC (database management system) by the way of taking care of data integrity, sharing, and integration.
- Additionally, the SCM repository provides a hub for the purpose of integration of different available software tools. It helps to manage the flow of the software process, and can implement uniform structure as well as format for software engineering related work products.
- To get these abilities, the repository is described in terms of a meta-model.
- The meta-model is used to identify the way by which information is kept in the repository, the way by which data can be accessed by tools and seen by software developers, how well data security as well as integrity can be maintained, and how simply the present model can be extended to fulfill new requirements.

General Features and Content

- To understand the features and contents of any repository, we have to see it from two perspectives: what are the content which are to be stored in the repository and what are the precise services which are provided by the repository.
- A comprehensive breakdown of sort of representations, documents, and supplementary work products which are kept in the repository is shown in Fig. 4.6.2.
- There are two different classes of services regarding any robust repository :
  (1) The similar types of services which may be required by any sophisticated DBMS.
  (2) The services which are specific to the environment of software engineering.
- A repository that serves a software engineering team should also :
  (1) Incorporate with or directly support process management functions.
  (2) Support particular rules which govern the SCM function and the information which is maintained inside the repository.
  (3) Provide an interface to other tools which are related to software engineering.
  (4) Maintain storage of several types of sophisticated data objects such as text, graphics, video, audio, etc.

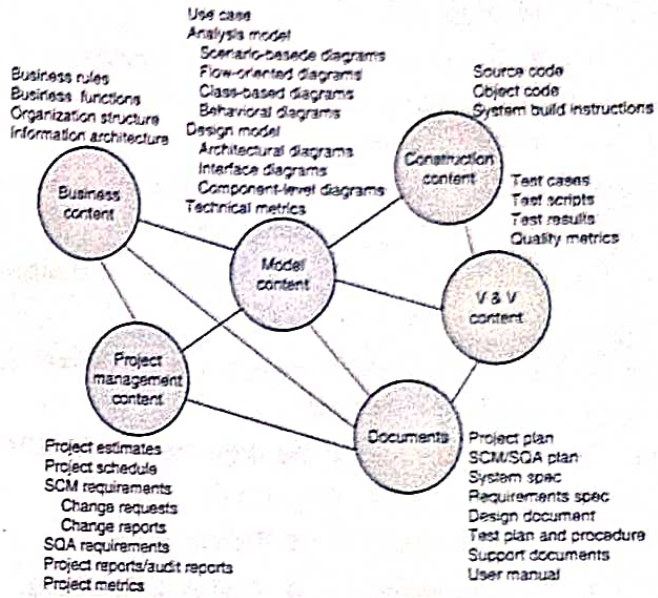


Fig. 4.6.2 : Content of the repository

Syllabus Topic : SCM Process

4.6.4 SCM Process

GQ. Write note on SCM Process. **(10 Marks)**

- The SCM process defines a sequence of tasks which have 4 most important objectives :
  (1) To identify all of the respective items which jointly define the software configuration,
  (2) To manage changes applied to these items,
  (3) To ease the building of different versions of an application, and
  (4) To make it sure that there is consistency in software quality as the configuration evolves over time.

- A process which is able to achieve these objectives need not be technical or heavy, but it must be characterized in such a way that it should enable a software team to solve all of the complex issues :
  - What will be the way for the software team to recognize the distinct elements of a software configuration?
  - What will be the way for the organization to manage the several existing versions of an application that will help to accommodate change efficiently?
  - What will be the way for the organization to control changes prior and later the software is released to a customer?
  - Who are individuals responsible for approving as well as ranking requested changes?
  - How can we conform that the respective changes have been made correctly?
  - What will be the mechanism to apprise others of changes that are made?
- These questions lead to the definition of five SCM tasks :

1. Identification,
2. Version control,
3. Change control,
4. Configuration auditing,
5. Reporting

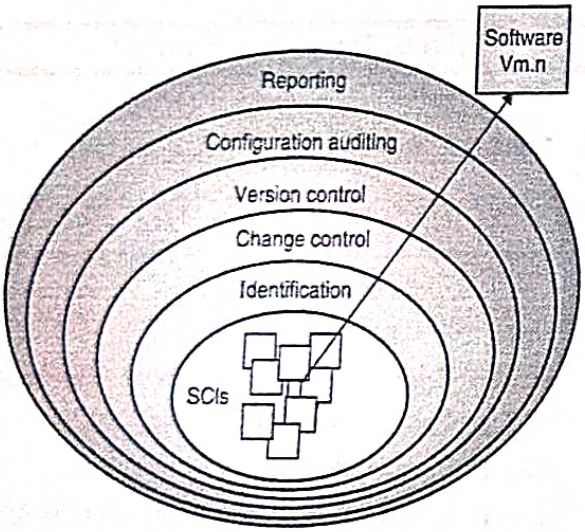


Fig. 4.6.3 : SCM Tasks

4.6.4(A) Configuration Identification

GQ. What is Configuration Identification in SCM ? **(5 Marks)**

- The first step in the SCM system is the identification of the items to be managed and the specification of the management authority responsible for each item.
- This is a critical step in the system since the identification scheme is employed throughout the lifecycle of the software product.
- The levels of authority assigned responsibility for each configuration item will vary depending on the complexity of the software product, the size of the development team, and the management style of the responsible organization.
- Configuration identification consists of selecting the configuration items and recording their functional and physical characteristics as set forth in technical documentation such as specifications, drawings, and associated lists.
- The following are examples of items managed in the SCM system :
  - Management plans
  - Specifications (requirements, design)
  - User documentation
  - Test plans, test design, case, and procedure specifications
  - Test data and test generation procedures
  - Support software used in development, even though not part of the delivered software product
  - Data dictionaries and various cross-references
  - Source code (on machine-readable media)
  - Executable code (the run-time system)
  - Libraries
  - Databases (data that are processed and data that are part of a program)
  - Maintenance documentation (e.g., listings, detail design descriptions)
- Effective management of the development of a software product requires careful definition of the configuration items.

- Changes to these items also need to be defined since these changes, along with the project baselines specify the evolution of the software product.
- SCM includes all of the entities of the software product as well as their various representations in documentation.
- The entities are not all subject to the same SCM disciplines at the same time.
- Support software is the class of software that may or may not be delivered with the software product, but is necessary for designing, enhancing, or testing the changes made during the life of the product.
- Support software may be user-furnished, developed in-house, leased from a vendor, or purchased off-the-shelf.
- In SCM, the focus is on describing the necessary controls used to manage support software.
- Developers and maintainers need to ensure that the support software is available for use for as long as the software product is in use.
- Whenever a production baseline is established, it is very important to archive all environmental and support tools along with the production code.
- Software tools used in development, especially compilers and middleware, should be placed under configuration control so their identities and versions are included in the baseline data about each release.
- Consideration also needs to be given to the hierarchy of entities managed during the SCM system. There are several different ways of structuring this hierarchy.
- One way is a three-level hierarchy consisting of configuration items, components, and units.
- Another way of structuring the entities is in terms of the interrelationships between the software being developed and the other software entities used during development and testing such as support software, test software, and the operating system.
- A third method for structuring entities is in terms of the intermediate products generated in the process of building the software product, such as: modifiable entities (e.g., source code, document, test data); compilation entities (e.g., compilers, support software); and configuration items (e.g., different representations created in the process of producing deliverables).
- An important aspect of configuration identification is the use of a formal naming convention for each entity.
- This naming convention, which typically uses a combination of mnemonic labels and version numbers, should be applied to all components of a configuration item.
- In establishing the naming convention, consideration should be given to system constraints such as name length or composition.
- Consistency in the application of this identification scheme is critical to accurate tracking of the software engineering process.

4.6.4(B) Change Control (Configuration Control)

UQ. Explain change control activities in SCM.

MU - May 15, Dec. 15, May 18, 5 Marks

UQ. Explain change management process.

MU - Dec. 16, 5 Marks

- Changes occur at all phases in the software lifecycle. Design or implementation changes may be necessary if requirements changes, or when deficiencies are identified in the design or implementation approach to a stable requirement.
- Testing may uncover defects that require changes in the code or the design and requirements.
- Changes must be made to the right version of the code, testing must verify performance of the change and the integrity of the remaining software, and all associated documentation must be updated to be consistent with the final code.
- A mechanism is needed to process change requests (e.g., discrepancies, failure reports, requests for new features) from a variety of sources throughout the development process and during operation and support of the software product.
- This mechanism should extend to a definition of a process to track, evaluate, and implement change requests into a new version and new release of the software.

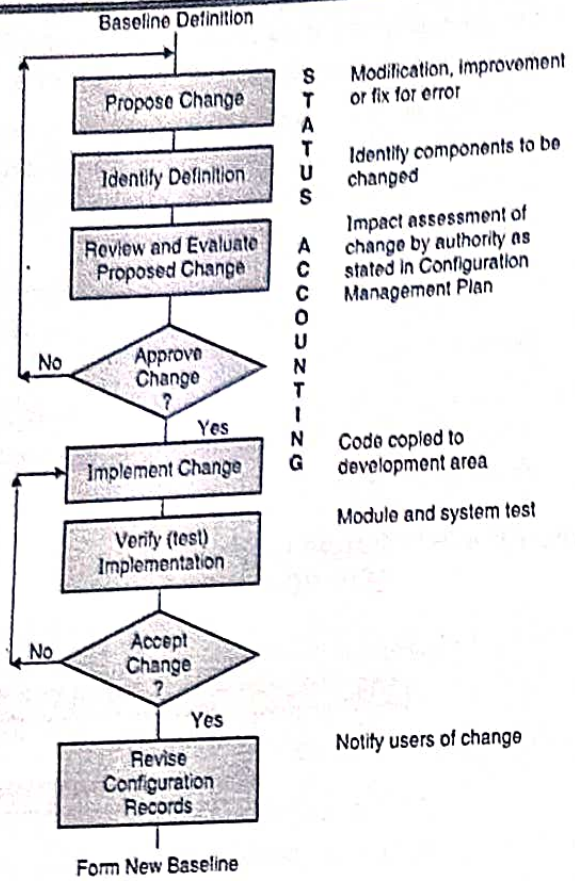


Fig. 4.6.4 : Configuration Control Process

- Generally, no single procedure can meet the needs of all levels of change management and approval levels.
- The minimum activities needed for processing changes include :
  - Defining the information needed for approving the requested change.
  - Identifying the review process and routing of information.
  - Describing the control of the libraries used to process the change.
  - Developing a procedure for implementing each change in the code, the documentation, and the released software product.

4.6.4(C) Version Control

UQ. Explain version control activities in SCM.

MU - May 15, Dec. 15, May 18, 5 Marks

- Version control integrates the procedures with tools for the purpose of managing different versions regarding the configuration objects which are generate throughout the software process.
- Following are the important capabilities which version control system implements or straightforwardly integrated with :
  - A project database (repository) which holds all th appropriate configuration objects,
  - A version management capability which holds a the relevant versions of a configuration object,
  - A make facility which helps to get all respectiv configuration objects and build a particula version of the software.
- Additionally, version control as well as change contro systems usually implement an issues/bugs tracking capability which helps the team to record and track th status regarding all of the important issues related wit each and every configuration object.
- Most of the version control systems create a change se of all the respective changes which are necessary t build a particular version of the software.
- It is possible to identify various named change sets fo an application or system. This helps in building a version of the software by mentioning the change sets (by name) which must be applied to the baseline configuration.
- A system modeling approach is used to accomplish it. The system model contains:
  (1) A template which contains a component hierarchy and a "build order" for those components which gives information regarding how the system must be constructed,
  (2) Construction rules, and
  (3) Verification rules.
- Now-a-days there are several automated approaches have been proposed to version control.
- The basic distinction in those approaches is the sophistication of the attributes which helps in building specific versions.

4.6.4(D) Configuration Audit

GQ. What is Configuration Audit in SCM ? (5 Marks)

- Identification, version control, and change control are the processes which help user to maintain sequence in what would otherwise be a confused situation.
- However change is tracked by even most successful control mechanisms only until generation of ECO (Engineering Change Order).
- What will be the way by which a software team make it sure that the change has been appropriately implemented?
- The answer has two aspects :
  (1) Technical reviews and
  (2) The software configuration audit.
- The focus of technical review is on the technical accuracy of the configuration object in which modifications are done.
- The reviewers assess the SCI (Software Configuration Items) for the purpose of determining consistency with other SCIs, omissions, or potential side effects.
- There is need to conduct a technical review for all but the most trivial changes.
- A software configuration audit in general complements the technical review by the way of assessing a configuration object for characteristics which are usually ignored in the process of review.
- The audit asks and answers the following questions :
  1. Has the modifications mentioned in the ECO been made? Have any additional modifications been incorporated?
  2. Is there any process of conducting technical review for assessing technical correctness?
  3. Has the software process been followed and is the application of software engineering standards done properly?
  4. Has the change been focused in the SCI? Have there is specification of change date and change author? Do the change is reflected by attributes of the configuration object?
  5. Has there is implementation of SCM procedures for noting the change, recording it, and reporting?
  6. Has there is proper updations in all related SCIs?
- Sometimes, the audit questions are asked as an integral part in the process of technical review.
- However, when SCM is a formal activity, the quality assurance group conducts the configuration audit separately.
- This formal configuration audits also make it sure that the accurate SCIs have been integrated into a precise build and that the whole documentation is up-to-date and consistent with the currently built version.

4.6.4(E) Status Reporting

- Configuration status reporting (CSR) which is also known as status accounting is an SCM task that gives answers of following questions:
  1. What happened?
  2. Who did it?
  3. When did it happen?
  4. What else will be affected?
- A CSR entry is made whenever a new or updated identification is assigned to SCI
- A CSR entry is made whenever the CCA (Change Control Authority) approves a change.
- Whenever a configuration audit is carried out, the results are reported as part of the CSR task.
- It is possible to keep the output from CSR in an online database or website, because of which it will be easy for software developers or support staff to access the change information by keyword category.

Additionally, generation of a CSR report is done regularly and is intended to give latest information regarding important changes to management as well as the practitioners.

Software Reliability

GQ. Discuss about Software Reliability in details .

- There is no doubt that the reliability of a computer program is an important element of its overall quality. If a program repeatedly and frequently fails to perform, it matters little whether other software quality factors are acceptable.

- Software reliability, unlike many other quality factors, can be measured directly and estimated using historical and developmental data. *Software reliability* is defined in statistical terms as "the probability of failure-free operation of a computer program in a specified environment for a specified time".
- To illustrate, program X is estimated to have a reliability of 0.999 over eight elapsed processing hours. In other words, if program X were to be executed 1000 times and require a total of eight hours of elapsed processing time (execution time), it is likely to operate correctly (without failure) 999 times. Whenever software reliability is discussed, a pivotal question arises: What is meant by the term *failure*? In the context of any discussion of software quality and reliability, failure is nonconformance to software requirements. Yet, even within this definition, there are gradations.
- Failures can be only annoying or catastrophic. One failure can be corrected within seconds, while another requires weeks or even months to correct.
- Complicating the issue even further, the correction of one failure may in fact result in the introduction of other errors that ultimately result in other failures.

Measures of Reliability and Availability

- Early work in software reliability attempted to extrapolate the mathematics of hardware reliability theory to the prediction of software reliability.
- Most hardware-related reliability models are predicated on failure due to wear rather than failure due to design defects.
- In hardware, failures due to physical wear (e.g., the effects of temperature, corrosion, shock) are more likely than a design-related failure. Unfortunately, the opposite is true for software. In fact, all software failures can be traced to design or implementation problems.
- There has been an ongoing debate over the relationship between key concepts in hardware reliability and their applicability to software.
- Although an irrefutable link has yet to be established, it is worthwhile to consider a few simple concepts that apply to both system elements. If we consider a computer-based system, a simple measure of reliability is *meantime-between-failure* (MTBF):

$$MTBF = MTTF + MTTR$$

where the acronyms MTTF and MTTR are *mean-time-to-failure* and *mean-time-to repair2* , respectively.

- Many researchers argue that MTBF is a far more useful measure than other quality-related software metrics discussed in Chapter 23. Stated simply, an end user is concerned with failures, not with the total defect count. Because each defect contained within a program does not have the same failure rate, the total defect count provides little indication of the reliability of a system. For example, consider a program that has been in operation for 3000 processor hours without failure.
- Many defects in this program may remain undetected for tens of thousand of hours before they are discovered. The MTBF of such obscure errors might be 30,000 or even 60,000 processor hours.
- Other defects, as yet undiscovered, might have a failure rate of 4000 or 5000 hours. Even if every one of the first category of errors (those with long MTBF) is removed, the impact on software reliability is negligible. However, MTBF can be problematic for two reasons: (1) it projects a time span between failures, but does not provide us with a projected failure rate, and (2) MTBF can be misinterpreted to mean average life span even though this is *not* what it implies.
- An alternative measure of reliability is *failures-in-time* (FIT)—a statistical measure of how many failures a component will have over one billion hours of operation. Therefore, 1 FIT is equivalent to one failure in every billion hours of operation. In addition to a reliability measure, you should also develop a measure of availability.
- *Software availability* is the probability that a program is operating according to requirements at a given point in time and is defined as

$$\text{Availability} = \frac{MTTF}{MTTF + MTTR} * 100\%$$

- The MTBF reliability measure is equally sensitive to MTTF and MTTR. The availability measure is somewhat more sensitive to MTTR, an indirect measure of the maintainability of software.

Software Safety

GQ. What is Software Safety ?

- *Software safety* is a software quality assurance activity that focuses on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail.
- If hazards can be identified early in the software process, software design features can be specified that will either eliminate or control potential hazards.
- A modeling and analysis process is conducted as part of software safety. Initially, hazards are identified and categorized by criticality and risk. For example, some of the hazards associated with a computer-based cruise control for an automobile might be: (1) causes uncontrolled acceleration that cannot be stopped, (2) does not respond to depression of brake pedal (by turning off), (3) does not engage when switch is activated, and (4) slowly loses or gains speed. Once these system-level hazards are identified, analysis techniques are used to assign severity and probability of occurrence.
- To be effective, software must be analyzed in the context of the entire system. For example, a subtle user input error (people are system components) may be magnified by a software fault to produce control data that improperly positions a mechanical device. If and only if a set of external environmental conditions is met, the improper position of the mechanical device will cause a disastrous failure.
- Analysis techniques such as fault tree analysis, real-time logic, and Petri net models can be used to predict the chain of events that can cause hazards and the probability that each of the events will occur to create the chain.
- Once hazards are identified and analyzed, safety-related requirements can be specified for the software. That is, the specification can contain a list of undesirable events and the desired system responses to these events. The role of software in managing undesirable events is then indicated.
- Although software reliability and software safety are closely related to one another, it is important to understand the subtle difference between them. Software reliability uses statistical analysis to determine the likelihood that a software failure will occur.
- However, the occurrence of a failure does not necessarily result in a hazard or mishap. Software safety examines the ways in which failures result in conditions that can lead to a mishap. That is, failures are not considered in a vacuum, but are evaluated in the context of an entire computer-based system and its environment.

Formal Technical Review (FTR)

GQ. What is the important of FTR in software development process ?

GQ. What is different guidelines for formal technical reviews.

- A *formal technical review* (FTR) is a software quality control activity performed by software engineers (and others). The objectives of an FTR are: (1) to uncover errors in function, logic, or implementation for any representation of the software; (2) to verify that the software under review meets its requirements; (3) to ensure that the software has been represented according to predefined standards; (4) to achieve software that is developed in a uniform manner; and (5) to make projects more manageable.
- In addition, the FTR serves as a training ground, enabling junior engineers to observe different approaches to software analysis, design, and implementation.
- The FTR also serves to promote backup and continuity because a number of people become familiar with parts of the software that they may not have otherwise seen. The FTR is actually a class of reviews that includes *walkthroughs* and *inspections*.
- Each FTR is conducted as a meeting and will be successful only if it is properly planned, controlled, and attended. In the sections that follow, guidelines similar to those for a walkthrough are presented as a representative formal technical review

The Review Meeting

Regardless of the FTR format that is chosen, every review meeting should abide by the following constraints:

- Between three and five people (typically) should be involved in the review.
- Advance preparation should occur but should require no more than two hours of work for each person.
- The duration of the review meeting should be less than two hours.
- Given these constraints, it should be obvious that an FTR focuses on a specific (and small) part of the overall software. For example, rather than attempting to review an entire design, walkthroughs are conducted for each component or small group of components. By narrowing the focus, the FTR has a higher likelihood of uncovering errors.

The focus of the FTR is on a work product (e.g., a portion of a requirements model, a detailed component design, source code for a component). The individual who has developed the work product—the *producer*—informs the project leader that the work product is complete and that a review is required. The project leader contacts *review leader*, who evaluates the product for readiness, generates copies of product materials, and distributes them to two or three *reviewers* for advance preparation. Each reviewer is expected to spend between one and two hours reviewing the product, making notes, and otherwise becoming familiar with the work. Concurrently, the review leader also reviews the product and establishes an agenda for the review meeting, which is typically scheduled for the next day.

The review meeting is attended by the review leader, all reviewers, and the producer.

One of the reviewers takes on the role of a *recorder*, that is, the individual who records (in writing) all important issues raised during the review. The FTR begins with an introduction of the agenda and a brief introduction by the producer. The producer then proceeds to "walk through" the work product, explaining the material, while reviewers raise issues based on their advance preparation. When valid problems or errors are discovered, the recorder notes each.

At the end of the review, all attendees of the FTR must decide whether to: (1) accept the product without further modification, (2) reject the product due to severe errors (once corrected, another review must be performed), or (3) accept the product provisionally (minor errors have been encountered and must be corrected, but no additional review will be required). After the decision is made, all FTR attendees complete a sign-off, indicating their participation in the review and their concurrence with the review team's findings.

Review Reporting and Record Keeping

- During the FTR, a reviewer (the recorder) actively records all issues that have been raised. These are summarized at the end of the review meeting, and a *review issues list* is produced. In addition, a *formal technical review summary report* is completed. A review summary report answers three questions:
  1. What was reviewed?
  2. Who reviewed it?
  3. What were the findings and conclusions?
- The review summary report is a single page form (with possible attachments). It becomes part of the project historical record and may be distributed to the project leader and other interested parties.
- The review issues list serves two purposes: (1) to identify problem areas within the product and (2) to serve as an action item checklist that guides the producer as corrections are made.
- An issues list is normally attached to the summary report. You should establish a follow-up procedure to ensure that items on the issues list have been properly corrected. Unless this is done, it is possible that issues raised can "fall between the cracks." One approach is to assign the responsibility for follow-up to the review leader.

Review Guidelines

- Guidelines for conducting formal technical reviews must be established in advance, distributed to all reviewers, agreed upon, and then followed.
- A review that is uncontrolled can often be worse than no review at all. The following represents a minimum set of guidelines for formal technical reviews:
  1. ***Review the product, not the producer.*** An FTR involves people and egos. Conducted properly, the FTR should leave all participants with a warm feeling of accomplishment. Conducted improperly, the FTR can take on the aura of an

inquisition. Errors should be pointed out gently; the tone of the meeting should be loose and constructive; the intent should not be to embarrass or belittle. The review leader should conduct the review meeting to ensure that the proper tone and attitude are maintained and should immediately halt a review that has gotten out of control.

2. ***Set an agenda and maintain it.*** One of the key maladies of meetings of all types is drift. An FTR must be kept on track and on schedule. The review leader is chartered with the responsibility for maintaining the meeting schedule and should not be afraid to nudge people when drift sets in.

3. *Limit debate and rebuttal.* When an issue is raised by a reviewer, there may not be universal agreement on its impact. Rather than spending time debating the question, the issue should be recorded for further discussion off-line.

4. ***Enunciate problem areas, but don't attempt to solve every problem noted.*** A review is not a problem-solving session. The solution of a problem can often be accomplished by the producer alone or with the help of only one other individual. Problem solving should be postponed until after the review meeting.

5. ***Take written notes.*** It is sometimes a good idea for the recorder to make notes on a wall board, so that wording and priorities can be assessed by other reviewers as information is recorded. Alternatively, notes may be entered directly into a notebook computer.

6. ***Limit the number of participants and insist upon advance preparation.*** Two heads are better than one, but 14 are not necessarily better than 4. Keep the number of people involved to the necessary minimum. However, all review team members must prepare in advance. Written comments should be solicited by the review leader (providing an indication that the reviewer has reviewed the material).

7. ***Develop a checklist for each product that is likely to be reviewed.*** A checklist helps the review leader to structure the FTR meeting and helps each reviewer to focus on important issues. Checklists should be developed for analysis, design, code, and even testing work products.

8. ***Allocate resources and schedule time for FTRs.*** For reviews to be effective, they should be scheduled as tasks during the software process. In addition, time should be scheduled for the inevitable modifications that will occur as the result of an FTR.

9. ***Conduct meaningful training for all reviewers.*** To be effective all review participants should receive some formal training. The training should stress both process-related issues and the human psychological side of reviews

10. ***Review your early reviews.*** Debriefing can be beneficial in uncovering problems with the review process itself. The very first product to be reviewed should be the review guidelines themselves.

Because many variables (e.g., number of participants, type of work products, timing and length, specific review approach) have an impact on a successful review, a software organization should experiment to determine what approach works best in a local context.

Sample-Driven Reviews

- In an ideal setting, every software engineering work product would undergo a formal technical review. In the real word of software projects, resources are limited and time is short. As a consequence, reviews are often skipped, even though their value as a quality control mechanism is recognized.
- A sample-driven review process in which samples of all software engineering work products are inspected to determine which work products are most error prone. Full FTR resources are then focused only on those work products that are likely (based on data collected during sampling) to be error prone.
- To be effective, the sample-driven review process must attempt to quantify those work products that are primary targets for full FTRs. To accomplish this, the following steps are suggested :
  1. Inspect a fraction a_i of each software work product *i*. Record the number of faults f_i found within a_i.
  2. Develop a gross estimate of the number of faults within work product *i* by multiplying f_i by $1/a_i$.
  3. Sort the work products in descending order according to the gross estimate of the number of faults in each.

4. Focus available review resources on those work products that have the highest estimated number of faults.

- The fraction of the work product that is sampled must be representative of the work product as a whole and large enough to be meaningful to the reviewers who do the sampling. As a_i increases, the likelihood that the sample is a valid representation of the work product also increases. However, the resources required to do sampling also increase. A software engineering team must establish the best value for a_i for the types of work products produced.

Walkthrough

Walk-through is an informal and manual review technique to find the defects, bugs and different kind of ambiguities and problems in the documents.

GQ. What are main goals and objectives of walk-through?

There are many objectives of the walkthrough. Some of them are as follows;

- To improve the product overall
- To detect the hidden and unseen bugs
- To evaluate the test cases and documents.
- Understanding of the review object
- To discuss the alternative of design
- To discuss the alternative of implementations
- Mutual learning

Chapter Ends...

MODULE V

CHAPTER 5

Software Testing and Maintenance

Syllabus

Testing: Software Quality, Testing: Strategic Approach, Strategic Issues,

Testing: Strategies for Conventional Software, Object oriented software, Web Apps Validating Testing- System Testing- Art of Debugging.

Maintenance : Software Maintenance-Software Supportability- Reengineering- Business Process Reengineering- Software Reengineering- Reverse Engineering- Restructuring- Forward Engineering.

5.1 Need of Software Testing 5-2
- 5.1.1 Advantages of Software Testing 5-2
- 5.1.2 Testing Tactics 5-2
- 5.1.3 Difference between Manual and Automation Testing 5-3
- 5.1.4 Principles of Software Testing 5-4
- 5.1.5 Objectives of Software Testing 5-4
- 5.1.6 Software Testing Process 5-5

5.2 Strategies Approach of Software Testing 5-6
- 5.2.1 Verification and Validation 5-6
- 5.2.2 Difference between Verification and Validation 5-7
- **UQ.** Compare Verification and Validation Testing. **MU - May 17, 3 Marks** 5-7
- 5.2.3 Testing Strategies 5-7
- **UQ.** Explain "Testing Strategy". **MU - Dec. 16, 10 Marks** 5-7

5.3 Unit Testing 5-8
- 5.3.1 Stubs and Drivers in Unit Testing 5-10
- 5.3.2 List of Errors Detected by Unit Testing 5-10

5.4 Integration Testing 5-10
- 5.4.1 Types of Integration Testing 5-11

5.5 Validation Testing 5-12
- 5.5.1 Validation-Test Criteria 5-13
- 5.5.2 Configuration Review 5-13
- 5.5.3 Alpha and Beta Testing 5-13
- 5.5.3(A) Difference between Alpha and Beta Testing 5-14
- **UQ.** Differentiate between Alpha and Beta Testing. **MU - May 17, 3 Marks** 5-14

5.6 System Testing 5-15
- **UQ.** Explain the need for system testing. **MU - May 15, 5 Marks** 5-15

5.7 Software Testing Fundamentals 5-16
- 5.7.1 Characteristics of a Good Test 5-17
- **Chapter Ends** 5-40

5.1 NEED OF SOFTWARE TESTING

- Software testing is a procedure to verify whether the actual results are same as of expected results.
- Software testing is performed to provide assurance that the software system does not contain any defects.
- Defects means errors which are detected by tester when actual result of our software module is not identical as expected result.
- Software testing helps us to find out whether all user requirements are fulfilled by our software or not.
- Software quality depends on; at what extend software fulfills user requirements and number of defects occurred in software.
- Software testing is used to give assurance that we deliver quality product to customer.
- To perform software testing we create test cases and test data. Test Case is a collection of actions which applied on our software product to check specific feature or functionality of it.
- Collection of test cases is called as test suit.
- Test data is the input provided to modules which are present in our software product. Test data represents the data which effects execution of the particular module. Sometime test data is used for positive testing. That means it is used to check that a provided set of input for given function generate expected result or not. Sometime test data is used for negative testing. That means test data is used to test the capability of our software modules to handle unexpected input.

5.1.1 Advantages of Software Testing

GQ. What are the advantages of Software Testing ? **(5 Marks)**

There are several advantages of software testing :

1. Testing reduces the possibility of software failure by removing errors which leads to software failure.
2. Testing process removes maximum possible errors from software and helps to deliver qualitative software to customer.
3. Testing ensures correctness and completeness of software along with quality.
4. Testing allows us to verify and validate the softw and ensures that software will satisfy all of the us requirements.
5. Testing enables the software to produce expec outcome by removing errors which prevent syst from producing expected outcome.

Syllabus Topic : Testing Tactics

5.1.2 Testing Tactics

GQ. Write note on Manual and Automation Testing. **(10 Marks**

Software Testing is divided in to two categories :

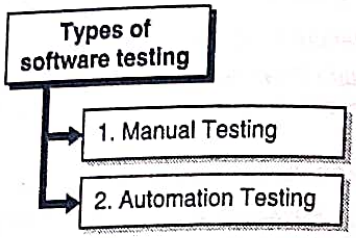


Fig. 5.1.1 : Types of software testing

1. Manual Testing

- Manual Testing is one of the type of Software Testing in which Tester treat himself as end user and use every module of application from login module to log out module to check application behavior according to requirement provided by user.
- In manual testing, tester manually runs all the test cases without help of any automation testing tools.
- Manual Testing is the very basic type of all testing types and supports to search the bug (error) in the software system.
- First we do manual testing for any new application before going for automation testing. We perform feasibility study of automation testing in manual testing i.e. do the analysis of benefits of performing automation testing for our project to take the decision that whether to do automation testing or not.
- For performing manual testing, we do not need knowledge of any testing tool.
- Manual Testing requires more effort, but is necessary to check automation feasibility.

- One important thing to know about Software Testing is that **"100% Automation is not possible"** so we need to perform manual testing.

Goals of Manual Testing

- The main purpose of performing manual testing is to give assurance that our software product does not contain any defect and it fulfills all functional requirements of end user.
- Test Suites (i.e. collection of test cases) or cases, are designed in the testing phase and they must test all functionalities present in our software product.
- Manual testing gives assurance that reported defects are fixed by developer and tester performs retesting on it after fixing the defects.
- Mainly manual testing verifies the quality of the software product and deploys bug-free software to the customer.
- List of manual testing is as follow :
  (i) Unit testing (ii) Integration testing
  (iii) System testing (iv) Acceptance testing

2. Automation Testing

- Automation Testing is a process in which we use automation tools to run our test cases to find out bugs from our software product.
- The automation software is able to enter test data into the software on which we perform testing and it also matches the expected and actual results to create test reports.
- Automation testing requires considerable amount of money and resources such as employees, testing tools etc.
- After change in requirement or occurrence of error, we need to make changes in code and we require executing same test suite again and again.
- We can record test suite by using test automation tools and re-play it when needed.
- Automation testing does not require human interaction to perform testing like manual testing.
- Aim of Automation testing is to decrease number of test cases to be executed manually but not replace Manual Testing completely.
- To perform automation testing we require generation of test cases, test data and test script.
- Automation testing is performed for project if requirements of projects are stable at some extent i.e. requirements are not frequently changing.
- List of some automation tools :
  (i) Selenium
  (ii) Quality Test Professional (QTP)
  (iii) HP ALM (Application Lifecycle Management)
  (iv) Buxila
  (v) Mentis

5.1.3 Difference between Manual and Automation Testing

GQ. Differentiate between manual and automation testing. **(5 Marks)**

Parameter	Manual Testing	Automation Testing
Human interaction	To perform manual testing human interaction is required for test execution.	To perform Automation Testing, human interaction is not required. In automation testing we use tools to run the test cases.
Requirements	To perform manual testing, we need skilled employees, more time and high costs.	To perform automation testing, we require automation tools. Automation testing saves time, cost and manpower. Once test suite is recorded in automation tool then it can easily run test suit.
Application to Test	Any application can be first tested manually. Informal testing like ad-hoc and monkey testing are performed manually.	Automated testing is used to test an application whose requirements are stable at some extent and automation testing mainly used to perform Regression Testing.

Parameter	Manual Testing	Automation Testing
Execution of Same test cases	We cannot use manual testing while executing same test suit. Repetitive execution of test cases become boring and error prone.	The most boring part which is of running same test cases repetitively can be performed with the help of automation testing.

5.1.4 Principles of Software Testing

GQ. Explain the principles of software testing. (10 Marks)

There are 7 Testing principles which are as follows :

(a) Complete testing of software is not possible

- Complete testing of software is not possible. Rather, we require performing the best possible amount of testing with the help of risk assessment of the application.
- Risk assessment is the process of guessing module where defects may occur and test that module to find out those defects. Because we have not that much amount of time to test each and every module from our software, exhaustive or complete testing of software is not possible.

(b) Defect clustering

- This principle states that about eighty percent of defects are found in twenty percent of modules.
- Using experience, we can find out such risky modules.
- But this approach has its own demerits such as if we run similar test cases repetitively, such test cases does not find any new defect.

(c) Pesticide Paradox

- If we use same test cases repetitively for testing then after particular point we cannot find new defect. To solve this problem, the test cases required to be reviewed and revised on regular basis and include new test cases to help in finding new defects.
- Testers cannot use only existing testing mechanism. He should try to improve the existing methods to find out more defects. But after performing testing, you can never claim your product is bug free.

(d) Testing shows presence of defects

- Testing process can show that the defects are present in the system under test but it cannot prove that there are no defects.
- Even after testing we cannot guarantee that system is 100 % error free.
- Objectives of testing process are as follows :
  - (i) Finding defects which are created by developer while developing software.
  - (ii) Providing information about quality of software under test.
  - (iii) To ensure that system meets the business and the user requirements.
  - (iv) To gain customer's confidence by providing them a qualitative product.

(e) Absence of error

The software which is 99% bug free can be still unusable if it is tested for wrong requirements i.e. absence of error does not help if system does not satisfy user's need and requirement.

(f) Early Testing

- Testing must be started as early as possible in the Software Development Life Cycle (SDLC).
- Therefore any bug in the requirements or design phase are find out in early phases of software testing.
- It is less costly to correct that bug in early stages of testing.
- It is suggested that we can start detecting the bug at time when requirements are gathered.

(g) Testing is context dependent

'Testing is context dependent' specifies that the way tester test an online shopping site will be different from the way he / she test stand alone application.

5.1.5 Objectives of Software Testing

GQ. What are the objective of testing ? (5 Marks)

- Finding defects which can be generated by the programmer while doing coding of software.
- Report the detected defects to developer for correction and after correction retest the software product to give

assurance that software does not contain any defect.

- Software testing is done to give assurance that our software product fulfills all requirements of end user.
- To give assurance that we deliver quality software to our end user.
- To make sure that it satisfies requirements present in the BRS (Business Requirement Specification) and SRS (System Requirement Specifications).
- To get the confidence of the customers by giving them a quality software product which must be user friendly and fulfills all the requirements and expectations of customer.

5.1.6 Software Testing Process

GQ. Explain the software testing process in detail. (10 Marks)

- Software testing process is a sequence of various activities which helps to certify software system.
- Software testing process activities are : Requirement analysis, Test planning, Test case development, environment set up, Test execution, Test cycle closures.
- Each of the activity of software testing process has defined entry and exit criteria.

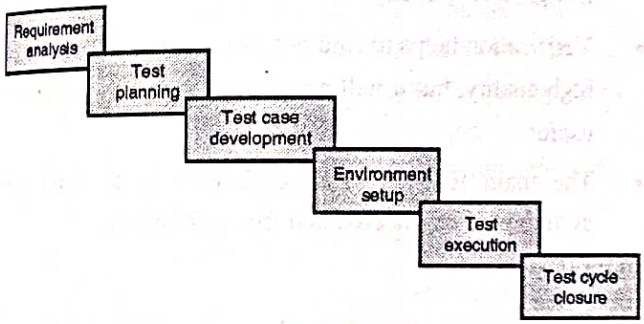


Fig. 5.1.2

- **Entry criteria :** Entry criterions of testing are prerequisite conditions that must be satisfied before testing begins.
- **Exit criteria :** An Exit criterion of testing describes the conditions that must be satisfied before testing is concluded.

Phases of Testing process

(i) **Requirement Analysis**

- In this phase all gathered requirements of system (i.e. functional as well as non-functional) are studies to determine whether these requirements are testable or not.
- Depending on requirements, it is decided to perform type of testing, what should be focus and priority of testing.
- At the end of this phase Requirement Traceability Matrix (RTM) is prepared.

(ii) **Test planning**

- It is also known as test strategy phase. It is responsibility of test manger. In this phase manager finds out required efforts and estimated cost for the system under development.
- At the end test plan is prepared as an outcome of test planning.

(iii) **Test Case Development**

- In this phase all test data is determined, reviewed and networked if necessary.
- It includes generation, verification and rework (as per requirement) of test cases and test scripts.
- Test cases and test scripts are outcome of this phase.

(iv) **Test environment set up**

- After completion of set up, smoke test is performed on it.
- Thus ready test environment and result of smoke test are outputs of this phase.

(v) **Test Execution**

- During this phase testing of software is carried out by test team using previously prepared test plan and test cases.
- Intension of test execution is to find bugs in software.
- These bugs are reported to development team for correcting them.
- After correction retest is performed in order to assure that reported bugs are removed.

(vi) **Test cycle closure**

- This is end of software testing process.
- In this phase cycle completion criteria is determined depending on time, test coverage, quality of system, cost of software, business objectives etc.
- By using all these attributes test metric is prepared.

- Before this phase testing has been completed and defect logs are available. Using or analyzing defect logs, test closure report is prepared.

5.2 STRATEGIES APPROACH OF SOFTWARE TESTING

GQ. Write note on Strategic Approach to software testing. **(10 Marks)**

- As we have seen, Testing is considered as a set of activities which can be planned in advance and implemented systematically.
- Hence there is need to define template for software testing which will be a series of steps in which specific test case design techniques as well as testing methods can be placed.
- There are number of software testing strategies available from which following characteristics are expected :
  - For effective testing, there is need of effective technical reviews. This helps to eliminate errors before testing commences.
  - Testing begins with component level and proceeds "outward" toward the incorporation of the whole computer-based system.
  - Different testing techniques are considered as suitable for different software engineering approaches and at different points in time.
  - In case of small projects software developer conducts the testing but for large projects, there is need of an independent test group.
  - Testing and debugging are considered as two distinct activities, but it is necessary to accommodate debugging in any testing strategy.
- For the strategy of software testing it is important to accommodate low-level tests which are required to verify that implementation of a small source code segment as well as high-level tests which are used to validate most important system functions against customer requirements is correct.
- A strategy must offer guidance for the practitioner (newcomer) and a set of milestones for the well experienced manager.
- Since the execution of steps regarding the test strategy is done at a time when there is increase in deadline pressure, progress must be measurable and problems should be identified as early as possible.

5.2.1 Verification and Validation

GQ. Explain verification and validation in brief. **(5 Marks)**

Verification

- Verification can be defined as a process of estimating the intermediary work products of a software development lifecycle to verify that we are in the correct track of creating the end user product.
- Now the question is : What is mean by the intermediary products? The answers is the **intermediary products** consists of documents which are generated during the development phases such as requirements specification, design documents, database table design, ER diagrams, test cases, traceability matrix etc.
- Sometimes we can avoid focusing on reviewing these documents but reviewing these documents can help us to find out most of the glitches. If these glitches are found at later phases of the development lifecycle then it will be very costly.
- Verification helps to find out whether the software is of high quality, but it will not guarantee that the system is useful.
- The main focus of the verification is to make the system well-engineered and free from errors.

Validation

- Validation can be defined as it is the process of evaluating the final product to ensure that the software meets all the specified requirements.
- It can also be defined as a process which shows that the product fulfills its intended use when deployed on suitable environment.
- Validation is performed in the software development lifecycle after the verification phase.

5.2.2 Difference between Verification and Validation

UQ. Compare Verification and Validation Testing. **MU - May 17, 3 Marks**

Parameter	Verification	Validation
Focus	Concentrates on right way to build a system.	Concentrates on output of build system.
Definition	Verification can be defined as it is a process of estimating the intermediary work products of a software development lifecycle to verify that we are in the correct track of creating the end user product.	Validation can be defined as it is the process of evaluating the final product to ensure that the software meets all the specified requirements.
Aim	The aim of Verification is to ensure that the product being develop is as per the requirements and design specifications.	The aim of Validation is to ensure that the product really meets up the user's requirements, and verify whether the specifications be correct in the first place.
Inclusion	Verification contains Reviews, Meetings and Inspections.	Validation contains testing such as black box testing, white box testing, gray box testing etc.
Performer	Verification is conducted by quality assurance team.	Validation is conducted by testing team.
Execution of code	Execution of code is not a part of Verification.	Execution of code is a part of Validation.
Explanation	Verification process gives explanation about whether the outputs are as per the inputs or not.	Validation process gives explanation about acceptance of software by the user or not.
Processes	Plans, Requirement Specifications, Design Specifications, Code, and Test Cases etc. are evaluated in verification.	Testing of actual software is done in validation.

Syllabus Topic : Testing Strategies

5.2.3 Testing Strategies

UQ. Explain "Testing Strategy". **MU - Dec. 16, 10 Marks**

- We have already seen phases of software testing process. Now we will see software process with different aspect.
- The software process can be considered as the spiral as shown in Fig. 5.2.1.
- In the beginning, the role of software is defined by the system engineering and proceed to software requirements analysis, in which there is establishment of the information domain, function, behavior, performance, constraints, and validation criteria for software.
- Moving towards inward in the spiral, we come to design and at the end to coding.
- It is possible to view a strategy for software testing in the context of the spiral.
- Unit testing gets started at the vortex of the spiral and focus on all of the software units such as component, class, or WebApp content object which are implemented in source code.

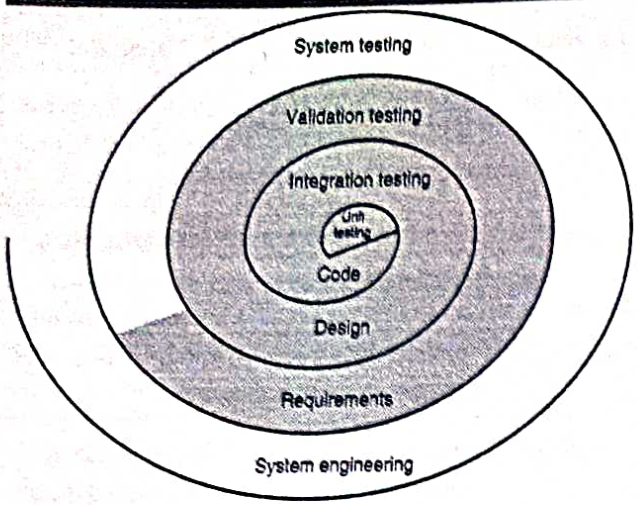


Fig. 5.2.1 : Software Process

- Testing continues by traversing outward direction along the spiral to integration testing. Here the main concentration is on designing and building the software architecture.
- Taking next turn outward on the spiral, we can observe validation testing, in which requirements which are set as part of requirements modeling are validated against the respective software which has been developed.
- At last we reach at system testing, where testing of the software as well as other system elements is done as a whole.
- For testing any computer software, we have to spiral out in a clockwise direction along streamlines which enhances the scope of testing with each and every turn.
- Taking into account the process from a procedural point of view, in the context of software engineering, practically testing is an implementation of four steps in a sequence.
- The steps are shown in Fig. 5.2.2.
- In the beginning, tests concentrate on every component individually, making sure they function properly as a unit.
- Therefore, the name is unit testing. Unit testing use testing techniques effectively which exercise particular paths in control structure of a component to confirm entire coverage as well maximum possible error detection.
- Further there is need to assemble or integrate components to form the whole software package.
- Integration testing highlights the concerns whic related to the dual problems of verification program construction.
- Test case design techniques which in general us concentrate on inputs and outputs are more comm the process of integration.

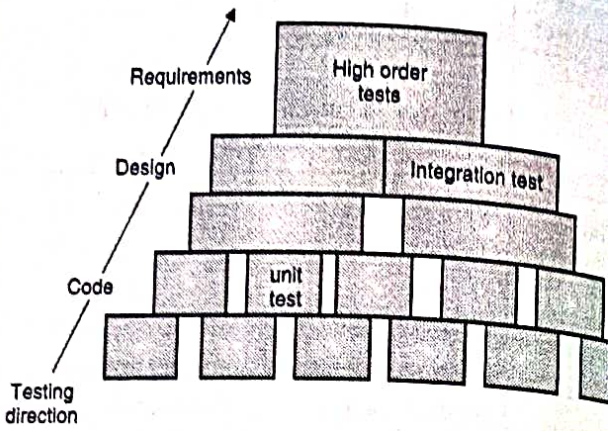


Fig. 5.2.2 : Four Steps of Testing

- After integration of software, a series of high-o tests is conducted.
- There is need to evaluate validation criteria which been set during requirements analysis.
- Validation testing presents absolute assurance that informational, functional, behavioral, and performa requirements are fulfilled by the software.
- The final high-order testing step is considered outs the limit of software engineering and in the wi context of computer system engineering.
- It is necessary to combine the software which validated with other system elements such as hardwa people, database, etc.
- System testing confirms that all the elements me correctly and the overall system function/performar has been achieved.

5.3 UNIT TESTING

GQ. Explain unit testing. **(5 Mark**

- **Software testing levels** are the different stages of t software development lifecycle where testing conducted.
- There are four levels of software testing : Unit > Integration >> System >> Acceptance.

- We are going to see the Unit testing in detail.

Unit Testing

- Unit Testing is a level of software testing where individual units / components of a software are tested.
- The purpose is to validate that each unit of the software performs as designed.
- A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output.
- In procedural programming, a unit may be an individual program, function, procedure, etc.
- In object-oriented programming, the smallest unit is a method, which may belong to a base / super class, abstract class or derived/ child class. (Some treat a module of an application as a unit.

Unit Testing Method

It is performed by using the White Box Testing method.

When is it performed ?

Unit Testing is the first level of software testing and is performed prior to Integration Testing.

Who performs it ?

- It is normally performed by software developers themselves or their peers. In rare cases, it may also be performed by independent software testers.

Unit Testing Tasks / Aspects of the software tested in unit testing

- Unit Test Plan
- Prepare
- Review
- Rework
- Baseline
- Unit Test Cases/Scripts
- Prepare
- Review
- Rework
- Baseline
- Unit Test
- Perform

Unit Testing Benefits

- Unit testing increases confidence in changing / maintaining code. If good unit tests are written and if they are run every time any code is changed, we will be able to promptly catch any defects introduced due to the change.
- Also, if codes are already made less interdependent to make unit testing possible, the unintended impact of changes to any code is less.
- Codes are more reusable. In order to make unit testing possible, codes need to be modular. This means that codes are easier to reuse.
- Development is faster. If there is no unit testing in place, developer writes the code and performs that fuzzy 'developer test' (Developer set some breakpoints, fire up the GUI, provides a few inputs that hopefully hit the code and hope that are all is set.)
- But, if there is unit testing in place, developer/tester writes the test, write the code and run the test. Writing tests takes time but the time is compensated by the less amount of time it takes to run the tests; there is no need to fire up the GUI and provide all those inputs. And, also, unit tests are more reliable than 'developer tests'.
- Development is faster in the long run too. The effort required to find and fix defects found during unit testing is very less in comparison to the effort required to fix defects found during system testing or acceptance testing.
- The cost of fixing a defect detected during unit testing is lesser in comparison to that of defects detected at higher levels.
- Debugging is easy. When a test fails, only the latest changes need to be debugged. With testing at higher levels, changes made over the span of several days/weeks/months need to be scanned.

5.3.1 Stubs and Drivers in Unit Testing

GQ. Draw and explain stub and driver mechanism of unit testing. **(5 Marks)**

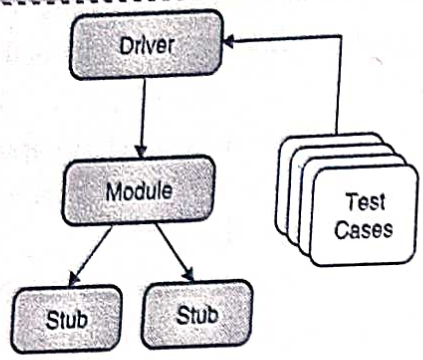


Fig. 5.3.1 : Stubs and Drivers

1) STUBS

Assume we have 3 modules, Module A, Module B and Module C. Module A is ready and we need to test it, but module A calls functions from Module B and C which are not ready, so developer will write a dummy module which simulates B and C and returns values to module A. This dummy module code is known as **stub**.

2) DRIVERS

Now suppose we have Modules B and C ready but Module A which calls functions from Module B and C is not ready so developer will write a dummy piece of code for Module A which will return values to Module B and C. This dummy piece of code is known as **driver**.

5.3.2 List of Errors Detected by Unit Testing

GQ. Enlist various types of errors detected by unit testing. **(10 Marks)**

Following is the list of errors which are usually Unit Testing can detect :

- Local data structure
- Boundary conditions
- Independent path
- Error handling path
- Local and global variable
- Incorrect initialization
- Incorrect symbolic representation of an expression
- Incorrect arithmetic precedence
- Global variable naming convention
- Mapping error
- Log and exception handling
- Referred different data type in the code and database
- Necessary input parameters not trim during the saving in the database.

5.4 INTEGRATION TESTING

GQ. Explain integration testing. **(5 Marks)**

- Integration testing is a method of software testing in which all units of software under test are integrated and tested as a single group.
- It is always done after unit testing is applied.
- It mainly focuses on testing data communication among all units of system.
- It makes use of mythologies as Big bang approach and incremental approach.
- Incremental approach can be either top-down, bottom-up or combination of both.
- The main objective of integration testing is to determine faults in communication between, integrated units.

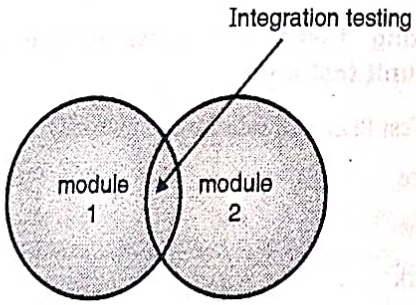


Fig. 5.4.1

Following are reasons for performing Integration testing :

- A Module is developed by software developer whose level of understanding and programming logic may not be same as of other software developers from project team.
- So, while performing Integration Testing, we required to check whether software module works efficiently with other modules or not.

- While developing single module, there is risk of changes in requirements by the clients.
- We may not perform unit testing for this new requirements and so performing integration testing of software is needed.
- There is possibility that interfaces (through which operations on database are done) present between software modules and database may contain error.
- There is possibility of error occurrence due to external hardware interfaces.
- There is possibility of raising issues if exceptions are not handled properly.
- For all these reasons we need to perform integration testing.

5.4.1 Types of Integration Testing

1. Big Bang Approach
2. Incremental Approach: which is categorized into given types :
   - Top Down Approach
   - Bottom Up Approach
   - Sandwich Approach - Combination of Top Down and Bottom Up

(1) Big bang approach

- In this strategy, all modules of software product are created first and then they are combined together and whole software is tested at once.
- Example of big bang approach As per the Fig. 5.4.2 all the modules from 'Module 1' to 'Module 7' are combined one by one and then the integration testing is performed.

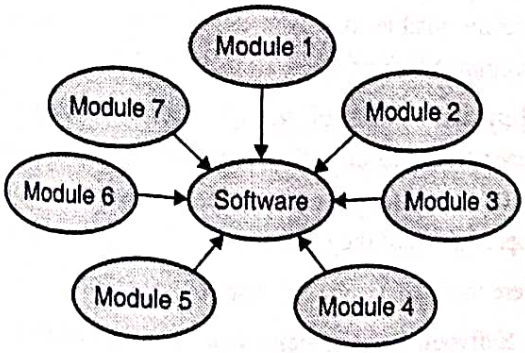


Fig. 5.4.2 : Big bang approach

Advantage of big bang approach

Suitable for small software projects.

Disadvantages of big bang approach

- Finding defect is difficult task because we test whole software at once.
- There are lots of interfaces which needed to be test.
- In big bang approach, there is possibility that some interfaces links may remain untested.
- Testing team get small amount of time for performing testing because in this approach integration testing starts after all the modules present in software are developed.
- High risk present in this approach is that critical modules are not separated and tested on priority basis because all modules are tested at the same time.
- Peripheral modules that are related with user interfaces are not separated and tested on priority basis.

(2) Incremental Approach

- In this approach, to perform testing two or more modules are merged with each other which are **logically related.**
- Then other associated modules are included in this group of modules and perform testing for checking whether they functioning correctly or not.
- This procedure continues until all of the modules are grouped together and tested successfully.
- This procedure is done with the help of dummy programs which are knows as **Stubs** and **Drivers.**
- Stubs and Drivers do not contain the complete programming logic of the software module but they contain code which is needed to perform communication with other modules.
- A Stub is dummy program which is called by the Module on which testing is performed. We replace called module by stub.
- Driver is dummy program which calls another module. We replace Module to be tested by driver i.e. Calling module is replaced by driver.

- Incremental Approach is performed using two different Methods such as :

(a) Top Down
(b) Bottom Up

(a) Top down Integration

- In Top to down approach, testing is performed from the modules present at top to modules which are present at the bottom.
- In top down approach, stubs are used for testing.

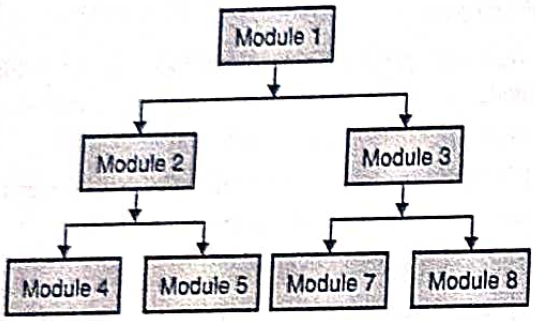


Fig. 5.4.3 : Top down approach

Advantages of Top down approach:

- Identification of bug is easy.
- Critical Modules are tested on priority basis so critical designing defects could be found and fixed early.

Disadvantages of Top down approach

- Top down testing requires many stubs because for replacing lower level modules there is need of stubs.
- Modules present at lower level are tested insufficiently because of lack of time.

(b) Bottom up Integration

- In the bottom up approach, every module present at lower level is tested with modules present at higher levels until all modules are tested.
- Drivers are used while performing bottom up incremental testing.

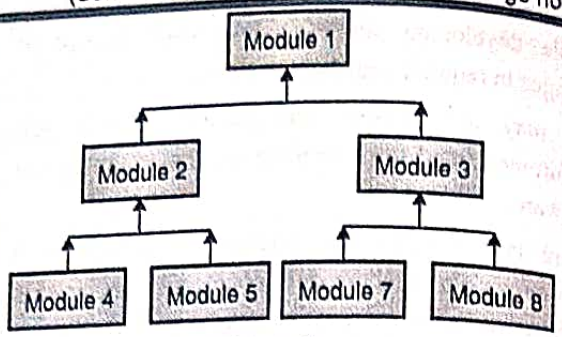


Fig. 5.4.4 : Bottom up approach

Advantages Bottom up approach

- Finding defect is easy task.
- After developing one module, testing is started. It does not waste time in waiting for all modules to be developed unlike Big-bang approach.

Disadvantage Bottom up approach

Critical modules which are present at the top level of software architecture that control the flow of software are tested last and may be defects occurs in critical modules.

5.5 VALIDATION TESTING

GQ. Explain Validation Testing in detail. **(10 Marks)**

- When integration testing ends, Validation testing begins. After completion of independent components, the software is completely integrated as a package, and interfacing errors have been found out and corrected.
- At the validation or system level, no difference remains in conventional software, object-oriented software, and WebApps.
- Testing is concentrated on the actions which are directly visible by user and output which is user-recognizable from the system.
- There are number of ways to define validation, but a general definition is that validation will be successful when the functionality of software is as per expectations of the customer.
- Here the main concern is user requirements.
- A Software Requirements Specification explains all of the user-visible attributes regarding the software and includes a Validation Criteria section which sets the basis for the approach regarding validation-testing.

5.5.1 Validation-Test Criteria

- To achieve software validation a series of tests is carried out which demonstrates conformity with user requirements.
- A test plan describes the classes regarding tests to be conducted.
- A test procedure defines particular test cases which are designed to ensure following :
  - All of the expected functional requirements are fulfilled.
  - All of the respective behavioral characteristics are achieved.
  - There is accuracy in all the content and are properly presented.
  - There is consideration of all performance requirements.
  - Documentation is accurate.
  - Usability as well as other essential requirements are met such as transportability, compatibility, error recovery, maintainability.
- After completion of each validation test case, there is one of two possible conditions :
  1. There is acceptance of the function or performance characteristic since it conforms to specification or
  2. A variation from specification is not covered and a list regarding deficiency is created.
- Before scheduled delivery, it is difficult to correct the deviations or errors found at this stage in the project.
- There is need to negotiate with the client (customer) to set a method for resolving deficiencies.

5.5.2 Configuration Review

- Configuration review is considered as a vital element of the validation process. The main aim of the review is making sure that all associated elements of the software configuration accurately developed are properly cataloged.
- The configuration review is also known as audit.

5.5.3 Alpha and Beta Testing

GQ. Write note on Alpha and Beta Testing. (10 Marks)

(1) Alpha testing

- Alpha Testing is software testing which is done to find out bugs before deploying the software application to end user.
- Alpha testing is a type of acceptance testing.
- This testing is called as an alpha testing since it is performed when development phase of software application is near to Beta Testing.
- The objective of alpha testing is to refine the software product by finding (and fixing) the bugs that were not discovered through previous tests.
- Alpha testing is typically performed by in-house software engineers or QA staff.
- It is the final testing stage before the software is released into the real world.
- Alpha testing is performed in two phases :
  (i) In first phase software is tested by development team members. They perform debugging of software to catch bugs quickly.
  (ii) In second phase software is tested by software quality analyst team for additional testing in actual user's environment setup.

Advantages of alpha testing

- Better insight about the software's reliability at its early stages.
- Free up team for other projects.
- Reduce delivery time to market.
- Early feedback helps to improve software quality.

(2) Beta testing

- Beta testing is testing which is performed at the location of customer. In this testing actual as well as intended users will test the software to determine whether the software is satisfying their needs and expectations.
- Beta testing allows users to test software before it is released to public.
- Beta testing minimizes the product failure risks and delivers qualitative product through customer validation. It gives direct feedback from the user.

- It ensures reliability, security, robustness etc. from user's perspective.
- Types of Beta testing are : Traditional, public, technical, focused and past release.
- Beta testing is also called as User Acceptance Testing, Customer acceptance Testing, Customer Validation testing and Pre-Release testing.
- Beta testing gives assurance that we deliver quality software to our customers by testing the product under various situations in real world environment that can't be created in a lab setting.

Advantages Beta Testing

- Beta testing decreases product failure risk by performing customer validations.
- Beta Testing permits tester to test the software application in post-launch infrastructure.
- Beta testing improves the quality of software product by using feedback of customer.
- Beta testing is cost effective as compared to other data collection techniques.
- Beta testing helps to increases customer satisfaction.

Types of beta testing

There are various types of Beta testing :

1. **Traditional Beta testing :** Software Product is provided to targeted end user and associated data is collected in all aspects. This data is useful for the improvement of Product.
2. **Public Beta Testing :** In this type of beta testing, product is released publicly in real world using online channels and data can be collected from anyone. Using feedback from end users, improvements in product are done.
3. **Technical Beta Testing :** In this type of beta testing, software Product is released in internal group of an organization and collect data from the employees of the organization.
4. **Focused Beta :** In this type of beta testing, software Product is released in the market for collecting feedback about specific features of the program.
5. **Post release Beta :** In this type of beta testing, software Product is released in the market and data is collected to improve the software product for the next release.

5.5.3(A) Difference between Alpha and Beta Testing

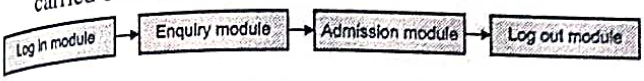
UQ. Differentiate between Alpha and Beta Testing.
MU - May 17, 3 Marks

Parameter	Alpha Testing	Beta Testing
Performed by	Alpha testing is done by Testers who are generally employees of organization.	Beta testing is done by End Users who are not employees of organization.
Testing environment	Alpha Testing done is in lab environment.	Beta testing is done in real world environment.
Factors tested	Security Testing and Reliability are not tested in depth during Alpha Testing.	Reliability, Security, Robustness is verified during Beta Testing.
Included testing type	Alpha testing includes white box and black box testing.	Beta Testing includes Black Box Testing.
Time required	Alpha testing requires lot of time to perform because in it white box as well as black box testing are performed.	Few weeks of time is required to perform the beta testing.
Fixing of issues	Important issues can be corrected by developers without delay in Alpha testing.	Issues detected in Beta testing will be corrected in future versions of the software product.
Focus	Alpha testing focuses on giving assurance about quality of the software product before going to Beta testing.	Beta testing also focus on quality of the product, but it collects feedback from end users about software product and gives assurance that the software product is ready for use in real world environment.

5.6 SYSTEM TESTING

UQ. Explain the need for system testing.
MU - May 15, 5 Marks

- System Testing is testing of entire and completely integrated software product.
- System Testing is sequence of various tests whose work is to test the complete computer based system.
- System testing is end-to-end testing. i.e. we test system from log in module to log out module.
- For e.g. In college admission project, system testing carried out as :



- System testing contains both functional (check whether requirements of user are fulfilled or not) as well as Non-Functional testing (check whether expectations of user are fulfilled or not).
- There are Two basic categories of Software Testing such as

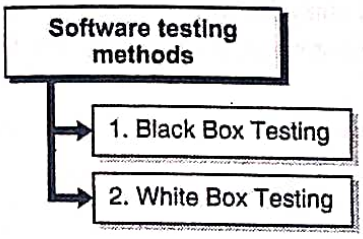


Fig. 5.6.1 : Software testing methods

1. Black Box Testing

In this type of testing, we check working of our software project by running it. We do not need to check code of our software in this testing type. **System testing** is included in black box testing

2. White Box Testing

In white box testing we check internal working of our code. So in this type of testing tester need deep knowledge of programming language which is used in our software product.

Types of System Testing

System Testing have more than 50 types. Below we have discussed some of them :

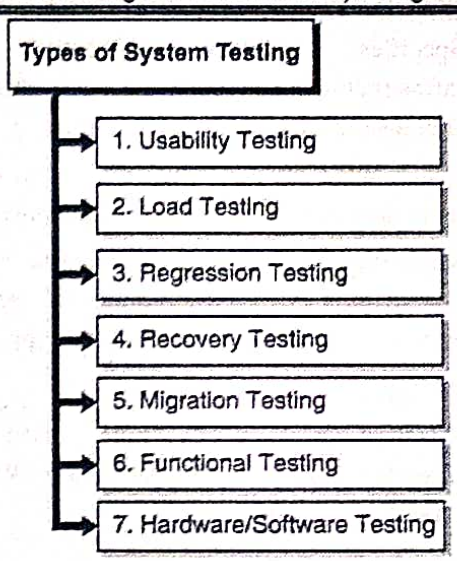


Fig. 5.6.2 : Types of System Testing

1. Usability Testing

- Usability Testing is a type of software testing in which a group of end-users of a software system use software to check user friendliness.
- It is non-functional testing.
- This testing basically concentrates on how easily user handles the application.
- It is verified that after few hours training, end user can use system comfortably.
- Icons used in user interface are not confusing. e.g. for cut option we use scissor picture, not CD picture.
- This type of testing is also known as User Experience Testing.
- This testing is suggested to perform at the initial design phase of SDLC because it find out bugs (errors) in user interface and give chance to improve them.

2. Load Testing

- Load testing is a type of Performance Testing which specifies performance of system under real-life load conditions.
- Load testing helps to specify how the application behaves when multiple users access it at same time.
- This testing normally used to find out :
  1. The maximum number of users who can access the software at same time.

2. Specifies whether currently available infrastructure i.e. software and hardware is adequate to execute the application.
3. Check what happen when maximum number of users accesses the system simultaneously.
4. Scalability (action taken to increase capacity of system such as increase storage etc.) to permit to more number of users to access our application.

- It is non-functional type testing.
- Load testing is normally used when we test Client/Server based applications and Web based applications.

3. Regression Testing

- Regression Testing is a type of software testing which is used to check whether changes have been made in code due to some error or change in requirement affects existing working functionality.
- In Regression Testing, we execute already executed test cases to give assurance that old functionalities work well after performing changes in code.
- This testing is performed to give guarantee that new code added in our software does not disturb working of existing functionalities.

4. Recovery Testing

- Recovery Testing is done to specify whether system recovers itself after the system crash due to disaster such as power failure or network not available etc.
- In recovery testing, system perform rollback i.e. identify the point where the behavior of the system was correct and then from that point again perform the operations up to the point where system get crashed.

5. Migration Testing

- Migration testing is performed to give assurance that the software can be moved from older system infrastructures to current system infrastructure without any problem.
- In Migration testing, we check data from old system for the compatibility with new system.

6. Functional Testing

- Functional testing is a type of software testing which checks that every **function** present in the software application works as per requirements of user.
- This testing includes black box testing and it does focus on the source code of the software application.
- Every functionality of the system is verified by tes using appropriate test data and actual result compared with expected result.
- If there is difference between actual result and expec result then bug is considered.
- Functional testing is done using Requirem Specification Document.

7. Hardware/Software Testing

- In Hardware/Software Testing, we perform testing communication between the hardware and softwa used in our system.
- IBM called the Hardware/Software Testing "HW/SW Testing".

5.7 SOFTWARE TESTING FUNDAMENTALS

- The main aim of testing is to search errors, and a goo test is the test which has a high probability of getting a error.
- Hence there is need to design and implement customized software or a product with keepin "testability" in mind.
- Simultaneously, it is also necessary that the tes themselves should exhibit a set of characteristics whic leads to encounter most of the errors with minimu efforts.
- **Testability :** James Bach provides the followin definition for testability: "Software testability is simpl how easily [a computer program] can be tested."

Following are characteristics of a good testable software

- **Operability :** "The better it works, the more efficientl it can be tested." If quality is focused while designin and implementing a system, comparatively sma number of bugs will block the execution of test allowing smooth testing.
- **Observability :** "What you see is what you test. Inputs which are given as part of testing return distinc outputs. System states as well as variables can be see

or queriable through execution. Wrong output is simply identified. Detection and reporting of internal errors is done automatically. Source code is accessible.

- **Controllability :** "The better we can control the software, the more the testing can be automated and optimized." Generations of all possible outputs is done using some combination of input, and I/O formats. The entire code can be executed by some combination of input. It is possible to control software and hardware states and variables straightly by the test engineer. It is also possible to specify tests conveniently, automate them, and reproduced.
- **Decomposability :** "By controlling the scope of testing, we can more quickly isolate problems and perform smarter retesting." The development of software system is implemented from individual modules which can be tested independently.
- **Simplicity :** "The less there is to test, the more quickly we can test it." Functional simplicity is expected from the program (e.g., the feature set is the minimum necessary to meet requirements); structural simplicity (e.g., architecture is modularized to limit the propagation of faults), and code simplicity (e.g., a coding standard is adopted for ease of inspection and maintenance).
- **Stability :** "The fewer the changes, the fewer the disruptions to testing." If Modifications to the software are less then it is easy to control them and the recovery of software is fast from failures.
- **Understandability :** "The more information we have, the smarter we will test." If there is availability of thorough information regarding architectural design and the dependencies among internal, external, as well as shared elements, test will be smarter.

5.7.1 Characteristics of a Good Test

GQ. Write Characteristics of a Good Test. **(5 Marks)**

There are some characteristics of a good test as follows :

- *There is high probability of finding out errors in a good test.* To achieve this goal, there is need for the tester to understand the software thoroughly and try to develop a view regarding how the software might fail.
- Preferably, the classes regarding failure are searched. For example, one class of possible failure in a GUI is the failure to identify proper mouse position.
- A series of tests possibly developed to exercise the mouse in an effort to exhibit an error in mouse position recognition.
- *A good test is not redundant.* There is limitation in testing time as well as resources.
- There is no meaning in performing a test which has same intention as another test.
- The purposes of different tests must be different.
- *A good test should be "best of breed".* In a set of tests which have same aim, time as well as resource limitations may mitigate toward the implementation of just a subset of these tests.
- In such situation, one should prefer the test having highest possibility of uncovering a whole class of errors.
- *A good test should be neither too simple nor too complex.* Even if sometimes tester can combine a series of tests into one test case, he should remember that the possible side effects related with this approach may mask errors.

  Usually it is better to execute each test separately.

Testing Strategies for Conventional Software:

GQ. What are different Strategies for Conventional Software.

- There are many strategies that can be used to test software. At one extreme, you can wait until the system is fully constructed and then conduct tests on the overall system in hopes of finding errors. This approach, although appealing, simply does not work. It will result in buggy software that disappoints all stakeholders.
- At the other extreme, you could conduct tests on a daily basis, whenever any part of the system is constructed. This approach, although less appealing to many, can be very effective. Unfortunately, some software developers hesitate to use it. What to do? A testing strategy that is chosen by most software teams falls between the two extremes.

- It takes an incremental view of testing, beginning with the testing of individual program units, moving to tests designed to facilitate the integration of the units, and culminating with tests that exercise the constructed system. Each of these classes of tests is described in the sections that follow.

Unit Testing

- *Unit testing* focuses verification effort on the smallest unit of software design—the software component or module. Using the component-level design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and the errors those tests uncover is limited by the constrained scope established for unit testing.
- The unit test focuses on the internal processing logic and data structures within the boundaries of a component. This type of testing can be conducted in parallel for multiple components.

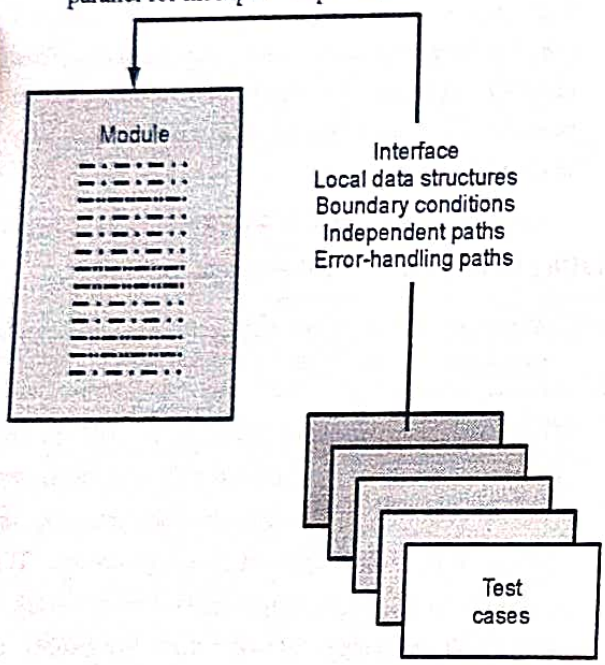


Fig. 5.7.1 : Unit Test

- **Unit-test considerations.** Unit tests are illustrated schematically in Fig. 5.7.1. The module interface is tested to ensure that information properly flows into and out of the program unit under test. Local data structures are examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution.
- All independent paths through the control structure exercised to ensure that all statements in a module h been executed at least once. Boundary conditions tested to ensure that the module operates properly boundaries established to limit or restrict processi And finally, all error-handling paths are tested. D flow across a component interface is tested before other testing is initiated.
- If data do not enter and exit properly, all other tests moot. In addition, local data structures should exercised and the local impact on global data should ascertained (if possible) during unit testing.
- Selective testing of execution paths is an essential ta during the unit test. Test cases should be designed uncover errors due to erroneous computatior incorrect comparisons, or improper control flow.
- Boundary testing is one of the most important ur testing tasks. Software often fails at its boundarie That is, errors often occur when the *n*th element of a *n*-dimensional array is processed, when the *i* repetition of a loop with *i* passes is invoked, when th maximum or minimum allowable value is encountered
- Test cases that exercise data structure, control flow, an data values just below, at, and just above maxima an minima are very likely to uncover errors. A goo design anticipates error conditions and establishe error-handling paths to reroute or cleanly terminate processing when an error does occur. Unfortunately there is a tendency to incorporate error handling into software and then never test it. A true story may serve to illustrate:
  - A computer-aided design system was developed under contract. In one transaction processing module, a practical joker placed the following error handling message after a series of conditional tests that invoked various control flow branches: ERROR! THERE IS NO WAY YOU CAN GET HERE. This "error message" was uncovered by a customer during user training!
  - Among the potential errors that should be tested when error handling is evaluated are: (1) error description is unintelligible, (2) error noted does not correspond to error encountered, (3) error condition causes system intervention prior to error

handling, (4) exception-condition processing is incorrect, or (5) error description does not provide enough information to assist in the location of the cause of the error.

- **Unit-test procedures.** Unit testing is normally considered as an adjunct to the coding step. The design of unit tests can occur before coding begins or after source code has been generated. A review of design information provides guidance for establishing test cases that are likely to uncover errors in each of the categories discussed earlier. Each test case should be coupled with a set of expected results. Because a component is not a stand-alone program, driver and/or stub software must often be developed for each unit test. The unit test environment is illustrated in Fig. 5.7.2.
- In most applications a driver is nothing more than a "main program" that maccepts test case data, passes such data to the component (to be tested), and prints relevant results. Stubs serve to replace modules that are subordinate (invoked by) the component to be tested.
- A stub or "dummy subprogram" uses the subordinate module's interface, may do minimal data manipulation, prints verification of entry, and returns control to the module undergoing testing.
- Drivers and stubs represent testing "overhead." That is, both are software that must be written (formal design is not commonly applied) but that is not delivered with the final software product. If drivers and stubs are kept simple, actual overhead is relatively low.
- Unfortunately, many components cannot be adequately unit tested with "simple" overhead software. In such cases, complete testing can be postponed until the integration test step (where drivers or stubs are also used).
- Unit testing is simplified when a component with high cohesion is designed. When only one function is addressed by a component, the number of test cases is reduced and errors can be more easily predicted and uncovered.

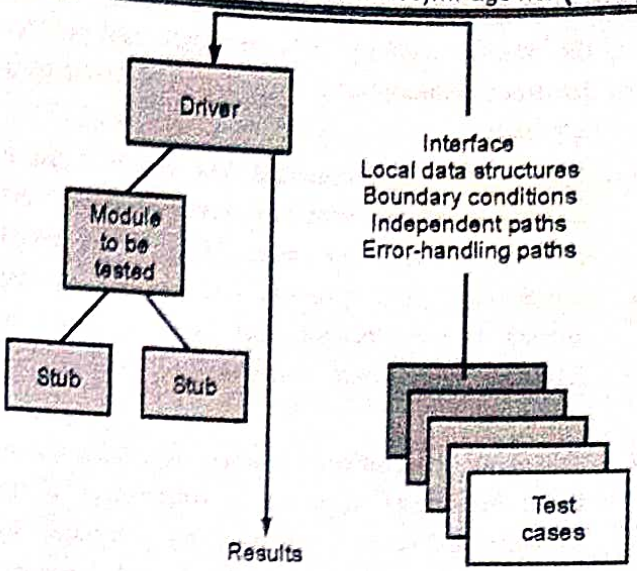


Fig. 5.7.2 : Unit Test Environment

Integration Testing

- A neophyte in the software world might ask a seemingly legitimate question once all modules have been unit tested: "If they all work individually, why do you doubt that they'll work when we put them together?" The problem, of course, is "putting them together"—interfacing.
- Data can be lost across an interface; one component can have an inadvertent, adverse effect on another; subfunctions, when combined, may not produce the desired major function; individually acceptable imprecision may be magnified to unacceptable levels; global data structures can present problems. Sadly, the list goes on and on.
- Integration testing is a systematic technique for constructing the software architecture while at the same time conducting tests to uncover errors associated with interfacing.
- The objective is to take unit-tested components and build a program structure that has been dictated by design. There is often a tendency to attempt nonincremental integration; that is, to construct the program using a "big bang" approach. All components are combined in advance.
- The entire program is tested as a whole. And chaos usually results! A set of errors is encountered. Correction is difficult because isolation of causes is complicated by the vast expanse of the entire program. Once these errors are corrected, new ones appear and

the process continues in a seemingly endless loop. Incremental integration is the antithesis of the big bang approach.

- The program is constructed and tested in small increments, where errors are easier to isolate and correct; interfaces are more likely to be tested completely; and a systematic test approach may be applied. In the paragraphs that follow, a number of different incremental integration strategies are discussed.
- **Top-down integration.** *Top-down integration testing* is an incremental approach to construction of the software architecture. Modules are integrated by moving downward through the control hierarchy, beginning with the main control module (main program). Modules subordinate (and ultimately subordinate) to the main control module are incorporated into the structure in either a depth-first or breadth-first manner.

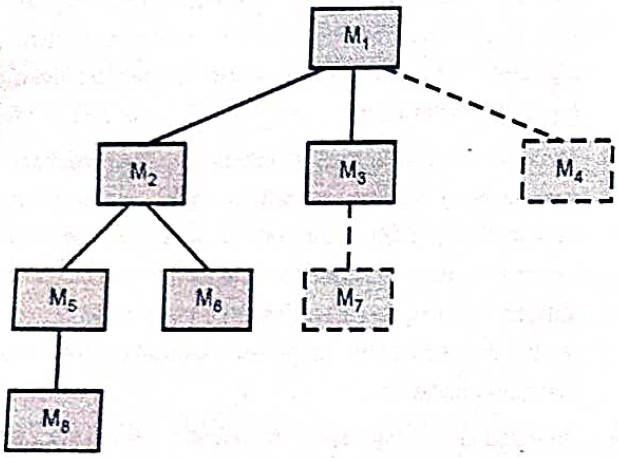


Fig. 5.7.3 : Top down integration

- Referring to Fig. 5.7.3, *depth-first integration* integrates all components on a major control path of the program structure. Selection of a major path is somewhat arbitrary and depends on application-specific characteristics. For example, selecting the left-hand path, components M_1, M_2, M_5 would be integrated first. Next, M_8 or (if necessary for proper functioning of M_2) M_6 would be integrated. Then, the central and right-hand control paths are built. *Breadth-first integration* incorporates all components directly subordinate at each level, moving across the structure horizontally. From the figure, components M_2, M_3, and M_4 would be integrated first. The next control level, M_5, M_6, and so on, follows. The integration process is performed in a series of five steps:

1. The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main control module.
2. Depending on the integration approach selected (i.e., depth or breadth first), subordinate stubs are replaced one at a time with actual components.
3. Tests are conducted as each component is integrated.
4. On completion of each set of tests, another stub is replaced with the real component.
5. Regression testing (discussed later in this section) may be conducted to ensure that new errors have not been introduced.

- The process continues from step 2 until the entire program structure is built. The top-down integration strategy verifies major control or decision points early in the test process.
- In a "well-factored" program structure, decision making occurs at upper levels in the hierarchy and is therefore encountered first. If major control problems do exist, early recognition is essential. If depth-first integration is selected, a complete function of the software may be implemented and demonstrated.
- Early demonstration of functional capability is a confidence builder for all stakeholders. Top-down strategy sounds relatively uncomplicated, but in practice, logistical problems can arise. The most common of these problems occurs when processing at low levels in the hierarchy is required to adequately test upper levels.
- Stubs replace low-level modules at the beginning of top-down testing; therefore, no significant data can flow upward in the program structure. As a tester, you are left with three choices: (1) delay many tests until stubs are replaced with actual modules, (2) develop stubs that perform limited functions that simulate the actual module, or (3) integrate the software from the bottom of the hierarchy upward.
- The first approach (delay tests until stubs are replaced by actual modules) can cause you to lose some control

over correspondence between specific tests and incorporation of specific modules. This can lead to difficulty in determining the cause of errors and tends to violate the highly constrained nature of the top-down approach.

- The second approach is workable but can lead to significant overhead, as stubs become more and more complex. The third approach, called bottom-up integration is discussed in the paragraphs that follow.
- **Bottom-up integration.** *Bottom-up integration testing*, as its name implies, begins construction and testing with *atomic modules* (i.e., components at the lowest levels in the program structure). Because components are integrated from the bottom up, the functionality provided by components subordinate to a given level is always available and the need for stubs is eliminated. A bottom-up integration strategy may be implemented with the following steps:

1. Low-level components are combined into clusters (sometimes called *builds*) that perform a specific software subfunction.
2. A *driver* (a control program for testing) is written to coordinate test case input and output.
3. The cluster is tested.
4. Drivers are removed and clusters are combined moving upward in the program structure.

- Integration follows the pattern illustrated in Fig. 5.7.4. Components are combined to form clusters 1, 2, and 3. Each of the clusters is tested using a driver (shown as a dashed block). Components in clusters 1 and 2 are subordinate to M_a. Drivers D_1 and D_2 are removed and the clusters are interfaced directly to M_a. Similarly, driver D_3 for cluster 3 is removed prior to integration with module M_b. Both M_a and M_b will ultimately be integrated with component M_c, and so forth.

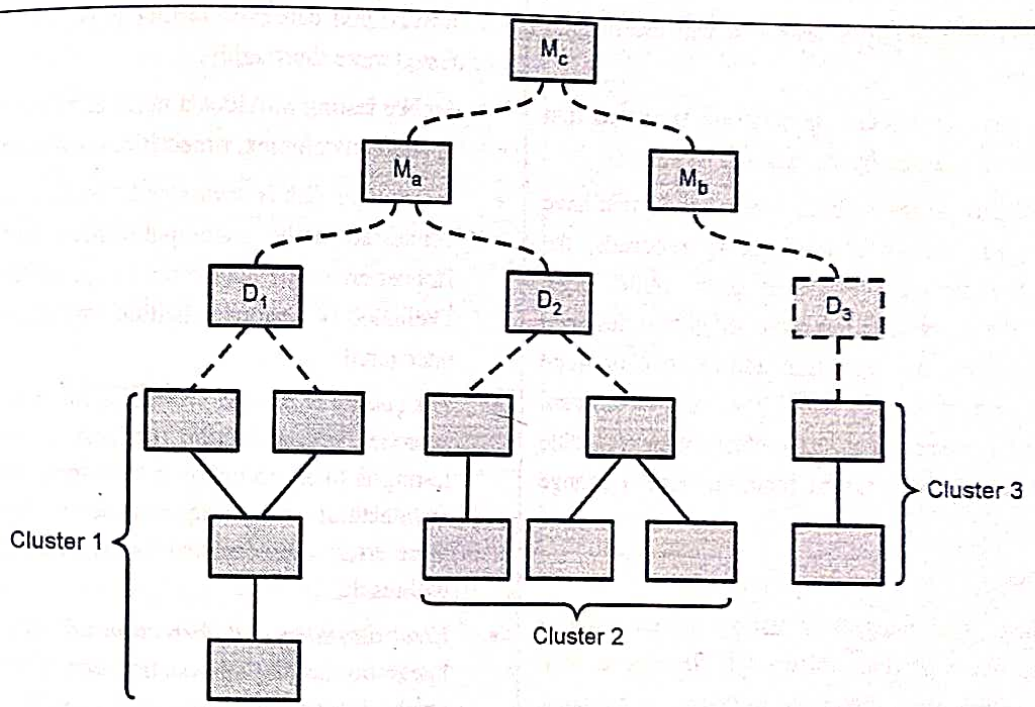


Fig. 5.7.4 : Bottom up integration

- As integration moves upward, the need for separate test drivers lessens. In fact, if the top two levels of program structure are integrated top down, the number of drivers can be reduced substantially and integration of clusters is greatly simplified.

Regression testing

Each time a new module is added as part of integration testing, the software changes. New data flow paths are established, new I/O may occur, and new control logic is invoked. These changes may cause problems with functions that previously worked flawlessly.

- In the context of an integration test strategy, *regression testing* is the reexecution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects. In a broader context, successful tests (of any kind) result in the discovery of errors, and errors must be corrected. Whenever software is corrected, some aspect of the software configuration (the program, its documentation, or the data that support it) is changed.
- Regression testing helps to ensure that changes (due to testing or for other reasons) do not introduce unintended behavior or additional errors. Regression testing may be conducted manually, by reexecuting a subset of all test cases or using automated capture/playback tools. *Capture/playback tools* enable the software engineer to capture test cases and results for subsequent playback and comparison. The *regression test suite* (the subset of tests to be executed) contains three different classes of test cases:
- A representative sample of tests that will exercise all software functions.
- Additional tests that focus on software functions that are likely to be affected by the change.
- Tests that focus on the software components that have been changed. As integration testing proceeds, the number of regression tests can grow quite large. Therefore, the regression test suite should be designed to include only those tests that address one or more classes of errors in each of the major program functions. It is impractical and inefficient to reexecute every test for every program function once a change has occurred.

Smoke testing

Smoke testing is an integration testing approach that is commonly used when product software is developed. It is designed as a pacing mechanism for time-critical projects, allowing the software team to assess the project on a frequent basis. In essence, the smoke-testing approach encompasses the following activities:

1. Software components that have been translated into code are integrated into a *build.* A build includes all data files, libraries, reusable modules, and engineered components that are required to implement one or more product functions.
2. A series of tests is designed to expose errors that will keep the build from properly performing its function. The intent should be to uncover "showstopper" errors that have the highest likelihood of throwing the software project behind schedule.
3. The build is integrated with other builds, and the entire product (in its current form) is smoke tested daily. The integration approach may be top down or bottom up. The daily frequency of testing the entire product may surprise some readers. However, frequent tests give both managers and practitioners a realistic assessment of integration testing progress. description of the smoke test in the following manner:

- The smoke test should exercise the entire system from end to end. It does not have to be exhaustive, but it should be capable of exposing major problems. The smoke test should be thorough enough that if the build passes, you can assume that it is stable enough to be tested more thoroughly.
- Smoke testing provides a number of benefits when it is applied on complex, timecritical software projects:
- *Integration risk is minimized.* Because smoke tests are conducted daily, incompatibilities and other show-stopper errors are uncovered early, thereby reducing the likelihood of serious schedule impact when errors are uncovered.
- *The quality of the end product is improved.* Because the approach is construction (integration) oriented, smoke testing is likely to uncover functional errors as well as architectural and component-level design errors. If these errors are corrected early, better product quality will result.
- *Error diagnosis and correction are simplified.* Like all integration testing approaches, errors uncovered during smoke testing are likely to be associated with "new software increments"—that is, the software that has just been added to the build(s) is a probable cause of a newly discovered error.
- *Progress is easier to assess.* With each passing day, more of the software has been integrated and more has been demonstrated to work. This improves team morale and gives managers a good indication that progress is being made.

- **Strategic options.** In general, the advantages of one strategy tend to result in disadvantages for the other strategy. The major disadvantage of the top-down approach is the need for stubs and the attendant testing difficulties that can be associated with them. Problems associated with stubs may be offset by the advantage of testing major control functions early.
- The major disadvantage of bottom-up integration is that "the program as an entity does not exist until the last module is added". This drawback is tempered by easier test case design and a lack of stubs. Selection of an integration strategy depends upon software characteristics and, sometimes, project schedule. In general, a combined approach (sometimes called sandwich testing) that uses top-down tests for upper levels of the program structure, coupled with bottom-up tests for subordinate levels may be the best compromise.
- As integration testing is conducted, the tester should identify critical modules. A critical module has one or more of the following characteristics: (1) addresses several software requirements, (2) has a high level of control (resides relatively high in the program structure), (3) is complex or error prone, or (4) has definite performance requirements. Critical modules should be tested as early as is possible. In addition, regression tests should focus on critical module function.
- **Integration test work products.** An overall plan for integration of the software and a description of specific tests is documented in a *Test Specification*. This work product incorporates a test plan and a test procedure and becomes part of the software configuration. Testing is divided into phases and builds that address specific functional and behavioral characteristics of the software. For example, integration testing for the *SafeHome* security system might be divided into the following test phases:
  - *User interaction* (command input and output, display representation, error processing and representation)
  - *Sensor processing* (acquisition of sensor output, determination of sensor conditions, actions required as a consequence of conditions)
  - *Communications functions* (ability to communicate with central monitoring station)
  - *Alarm processing* (tests of software actions that occur when an alarm is encountered) Each of these integration test phases delineates a broad functional category within the software and generally can be related to a specific domain within the software architecture. Therefore, program builds (groups of modules) are created to correspond to each phase.
  - The following criteria and corresponding tests are applied for all test phases:
    - *Interface integrity.* Internal and external interfaces are tested as each module (or cluster) is incorporated into the structure.
    - *Functional validity.* Tests designed to uncover functional errors are conducted.
    - *Information content.* Tests designed to uncover errors associated with local or global data structures are conducted.
    - *Performance.* Tests designed to verify performance bounds established during software design are conducted.
- A schedule for integration, the development of overhead software, and related topics are also discussed as part of the test plan. Start and end dates for each phase are established and "availability windows" for unit-tested modules are defined.
- A brief description of overhead software (stubs and drivers) concentrates on characteristics that might require special effort. Finally, test environment and resources are described. Unusual hardware configurations, exotic simulators, and special test tools or techniques are a few of many topics that may also be discussed.
- The detailed testing procedure that is required to accomplish the test plan is described next. The order of integration and corresponding tests at each integration step are described. A listing of all test cases (annotated for subsequent reference) and expected results are also included.
- A history of actual test results, problems, or peculiarities is recorded in a *Test Report* that can be

appended to the *Test Specification*, if desired. Information contained in this section can be vital during software maintenance.

- Appropriate references and appendixes are also presented. Like all other elements of a software configuration, the test specification format may be tailored to the local needs of a software engineering organization.
- It is important to note, however, that an integration strategy (contained in a test plan) and testing details (described in a test procedure) are essential ingredients and must appear.

Testing Strategies for Object oriented software

GQ. Discuss about strategies for object oriented software

- The objective of testing, stated simply, is to find the greatest possible number of errors with a manageable amount of effort applied over a realistic time span.
- Although this fundamental objective remains unchanged for object-oriented software, the nature of object-oriented software changes both testing strategy and testing tactics

Unit Testing in the OO Context

- When object-oriented software is considered, the concept of the unit changes. Encapsulation drives the definition of classes and objects. This means that each class and each instance of a class packages attributes (data) and the operations that manipulate these data.
- An encapsulated class is usually the focus of unit testing. However, operations (methods) within the class are the smallest testable units. Because a class can contain a number of different operations, and a particular operation may exist as part of a number of different classes, the tactics applied to unit testing must change.
- You can no longer test a single operation in isolation (the conventional view of unit testing) but rather as part of a class. To illustrate, consider a class hierarchy in which an operation X is defined for the superclass and is inherited by a number of subclasses.
- Each subclass uses operation X, but it is applied within the context of the private attributes and operations that have been defined for the subclass. Because the context in which operation X is used varies in subtle ways, it is necessary to test operation X in the context of each of the subclasses. This means that testing operation X in a stand-alone fashion (the conventional unit-testing approach) is usually ineffective in the object-oriented context. Class testing for OO software is the equivalent of unit testing for conventional software.
- Unlike unit testing of conventional software, which tends to focus on the algorithmic detail of a module and the data that flow across the module interface, class testing for OO software is driven by the operations encapsulated by the class and the state behavior of the class.

Integration Testing in the OO Context

- Because object-oriented software does not have an obvious hierarchical control structure, traditional top-down and bottom-up integration strategies have little meaning. In addition, integrating operations one at a time into a class (the conventional incremental integration approach) is often impossible because of the "direct and indirect interactions of the components that make up the class".
- There are two different strategies for integration testing of OO systems. The first, *thread-based testing*, integrates the set of classes required to respond to one input or event for the system. Each thread is integrated and tested individually.
- Regression testing is applied to ensure that no side effects occur. The second integration, approach, *use-based testing*, begins the construction of the system by testing those classes (called *independent classes*) that use very few (if any) *server* classes.
- After the independent classes are tested, the next layer of classes, called *dependent classes*, that use the independent classes are tested. This sequence of testing layers of dependent classes continues until the entire system is constructed. The use of drivers and stubs also changes when integration testing of OO systems is conducted.
- Drivers can be used to test operations at the lowest level and for the testing of whole groups of classes. A driver can also be used to replace the user interface so that tests of system functionality can be conducted prior to implementation of the interface.

- Stubs can be used in situations in which collaboration between classes is required but one or more of the collaborating classes has not yet been fully implemented.
- *Cluster testing* is one step in the integration testing of OO software. Here, a cluster of collaborating classes (determined by examining the CRC and object-relationship model) is exercised by designing test cases that attempt to uncover errors in the collaborations.

Testing: Strategies for WebApps

The strategy for WebApp testing adopts the basic principles for all software testing and applies a strategy and tactics that are used for object-oriented systems. The following steps summarize the approach:

1. The content model for the WebApp is reviewed to uncover errors.
2. The interface model is reviewed to ensure that all use cases can be accommodated.
3. The design model for the WebApp is reviewed to uncover navigation errors.
4. The user interface is tested to uncover errors in presentation and/or navigation mechanics.
5. Each functional component is unit tested.
6. Navigation throughout the architecture is tested.
7. The WebApp is implemented in a variety of different environmental configurations and is tested for compatibility with each configuration.
8. Security tests are conducted in an attempt to exploit vulnerabilities in the WebApp or within its environment.
9. Performance tests are conducted.
10. The WebApp is tested by a controlled and monitored population of end users. The results of their interaction with the system are

Validating Testing

GQ. Discuss about Validation Testing.

- Validation testing begins at the culmination of integration testing, when individual components have been exercised, the software is completely assembled as a package, and interfacing errors have been uncovered and corrected.
- At the validation or system level, the distinction between conventional software, object-oriented software, and WebApps disappears. Testing focuses on user-visible actions and user-recognizable output from the system.
- Validation can be defined in many ways, but a simple definition is that validation succeeds when software functions in a manner that can be reasonably expected by the customer.
- At this point a battle-hardened software developer might protest: "Who or what is the arbiter of reasonable expectations?" If a *Software Requirements Specification* has been developed, it describes all user-visible attributes of the software and contains a *Validation Criteria* section that forms the basis for a validation-testing approach.

Validation-Test Criteria

- Software validation is achieved through a series of tests that demonstrate conformity with requirements. A test plan outlines the classes of tests to be conducted, and a test procedure defines specific test cases that are designed to ensure that all functional requirements are satisfied, all behavioral characteristics are achieved, all content is accurate and properly presented, all performance requirements are attained, documentation is correct, and usability and other requirements are met (e.g., transportability, compatibility, error recovery, maintainability).
- After each validation test case has been conducted, one of two possible conditions exists: (1) The function or performance characteristic conforms to specification and is accepted or (2) a deviation from specification is uncovered and a deficiency list is created. Deviations or errors discovered at this stage in a project can rarely be corrected prior to scheduled delivery. It is often necessary to negotiate with the customer to establish a method for resolving deficiencies.

Configuration Review

- An important element of the validation process is a *configuration review*.
- The intent of the review is to ensure that all elements of the software configuration have been properly developed, are cataloged, and have the necessary detail to bolster the support activities.

Alpha and Beta Testing

- It is virtually impossible for a software developer to foresee how the customer will really use a program. Instructions for use may be misinterpreted; strange combinations of data may be regularly used; output that seemed clear to the tester may be unintelligible to a user in the field.
- When custom software is built for one customer, a series of acceptance tests are conducted to enable the customer to validate all requirements. Conducted by the end user rather than software engineers, an acceptance test can range from an informal "test drive" to a planned and systematically executed series of tests. In fact, acceptance testing can be conducted over a period of weeks or months, thereby uncovering cumulative errors that might degrade the system over time.
- If software is developed as a product to be used by many customers, it is impractical to perform formal acceptance tests with each one. Most software product builders use a process called alpha and beta testing to uncover errors that only the end user seems able to find.
- The *alpha test* is conducted at the developer's site by a representative group of end users. The software is used in a natural setting with the developer "looking over the shoulder" of the users and recording errors and usage problems. Alpha tests are conducted in a controlled environment.
- The *beta test* is conducted at one or more end-user sites. Unlike alpha testing, the developer generally is not present. Therefore, the beta test is a "live" application of the software in an environment that cannot be controlled by the developer.
- The customer records all problems (real or imagined) that are encountered during beta testing and reports these to the developer at regular intervals. As a result of problems reported during beta tests, you make modifications and then prepare for release of the software product to the entire customer base.
- A variation on beta testing, called *customer acceptance testing,* is sometimes performed when custom software is delivered to a customer under contract. The customer performs a series of specific tests in an attempt to uncover errors before accepting the software from the developer. In some cases (e.g., a major corporate or governmental system) acceptance testing can be very formal and encompass many days or even weeks of testing.

System Testing

- Software is only one element of a larger computer-based system. Ultimately, software is incorporated with other system elements (e.g., hardware, people, information), and a series of system integration and validation tests are conducted.
- These tests fall outside the scope of the software process and are not conducted solely by software engineers. However, steps taken during software design and testing can greatly improve the probability of successful software integration in the larger system.
- A classic system-testing problem is "finger pointing." This occurs when an error is uncovered, and the developers of different system elements blame each other for the problem. Rather than indulging in such nonsense, you should anticipate potential interfacing problems and (1) design error-handling paths that test all information coming from other elements of the system, (2) conduct a series of tests that simulate bad data or other potential errors at the software interface, (3) record the results of tests to use as "evidence" if finger pointing does occur, and (4) participate in planning and design of system tests to ensure that software is adequately tested.
- *System testing* is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that system elements have been properly integrated and perform allocated functions.

Recovery Testing

- Many computer-based systems must recover from faults and resume processing with little or no downtime. In some cases, a system must be fault tolerant; that is, processing faults must not cause overall system function to cease. In other cases, a system failure must be corrected within a specified period of time or severe economic damage will occur.

- *Recovery testing* is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed. If recovery is automatic (performed by the system itself), reinitialization, checkpointing mechanisms, data recovery, and restart are evaluated for correctness. If recovery requires human intervention, the mean-time-to-repair (MTTR) is evaluated to determine whether it is within acceptable limits.

Security Testing

- Any computer-based system that manages sensitive information or causes actions that can improperly harm (or benefit) individuals is a target for improper or illegal penetration. Penetration spans a broad range of activities: hackers who attempt to penetrate systems for sport, disgruntled employees who attempt to penetrate for revenge, dishonest individuals who attempt to penetrate for illicit personal gain.
- *Security testing* attempts to verify that protection mechanisms built into a system will, in fact, protect it from improper penetration. "The system's security must, of course, be tested for invulnerability from frontal attack—but must also be tested for invulnerability from flank or rear attack." During security testing, the tester plays the role(s) of the individual who desires to penetrate the system. Anything goes!
- The tester may attempt to acquire passwords through external clerical means; may attack the system with custom software designed to break down any defenses that have been constructed; may overwhelm the system, thereby denying service to others; may purposely cause system errors, hoping to penetrate during recovery; may browse through insecure data, hoping to find the key to system entry.
- Given enough time and resources, good security testing will ultimately penetrate a system. The role of the system designer is to make penetration cost more than the value of the information that will be obtained.

Stress Testing

- Earlier software testing steps resulted in thorough evaluation of normal program functions and performance. Stress tests are designed to confront programs with abnormal situations. In essence, the tester who performs stress testing asks: "How high can we crank this up before it fails?"
- *Stress testing* executes a system in a manner that demands resources in abnormal quantity, frequency, or volume. For example, (1) special tests may be designed that generate ten interrupts per second, when one or two is the average rate, (2) input data rates may be increased by an order of magnitude to determine how input functions will respond, (3) test cases that require maximum memory or other resources are executed, (4) test cases that may cause thrashing in a virtual operating system are designed, (5) test cases that may cause excessive hunting for disk-resident data are created. Essentially, the tester attempts to break the program.
- A variation of stress testing is a technique called *sensitivity testing*. In some situations (the most common occur in mathematical algorithms), a very small range of data contained within the bounds of valid data for a program may cause extreme and even erroneous processing or profound performance degradation.
- Sensitivity testing attempts to uncover data combinations within valid input classes that may cause instability or improper processing.

Performance Testing

- For real-time and embedded systems, software that provides required function but does not conform to performance requirements is unacceptable. Performance testing is designed to test the run-time performance of software within the context of an integrated system. Performance testing occurs throughout all steps in the testing process.
- Even at the unit level, the performance of an individual module may be assessed as tests are conducted. However, it is not until all system elements are fully integrated that the true performance of a system can be ascertained. Performance tests are often coupled with stress testing and usually require both hardware and software instrumentation. That is, it is often necessary to measure resource utilization (e.g., processor cycles) in an exacting fashion.
- External instrumentation can monitor execution intervals, log events (e.g., interrupts) as they occur, and

sample machine states on a regular basis. By instrumenting a system, the tester can uncover situations that lead to degradation and possible system failure.

Deployment Testing

- In many cases, software must execute on a variety of platforms and under more than one operating system environment. *Deployment testing*, sometimes called *configuration testing*, exercises the software in each environment in which it is to operate.
- In addition, deployment testing examines all installation procedures and specialized installation software (e.g., "installers") that will be used by customers, and all documentation that will be used to introduce the software to end users. As an example, consider the Internet-accessible version of *SafeHome* software that would allow a customer to monitor the security system from remote locations.
- The *SafeHome* WebApp must be tested using all Web browsers that are likely to be encountered. A more thorough deployment test might encompass combinations of Web browsers with various operating systems (e.g., Linux, Mac OS, Windows). Because security is a major issue, a complete set of security tests would be integrated with the deployment test.

Art of Debugging

GQ. Discuss about debugging.

GQ. Elaborate Different debugging strategies.

- Software testing is a process that can be systematically planned and specified. Testcase design can be conducted, a strategy can be defined, and results can be evaluated against prescribed expectations.
- *Debugging* occurs as a consequence of successful testing. That is, when a test case uncovers an error, debugging is the process that results in the removal of the error. Although debugging can and should be an orderly process, it is still very much an art.
- As a software engineer, you are often confronted with a "symptomatic" indication of a software problem as you evaluate the results of a test. That is, the external manifestation of the error and its internal cause may have no obvious relationship to one another. The poorly understood mental process that connects a symptom to a cause is debugging.

The Debugging Process

- Debugging is not testing but often occurs as a consequence of testing.s Referring to Fig. 5.7.5, the debugging process begins with the execution of a test case. Results are assessed and a lack of correspondence between expected and actual performance is encountered.
- In many cases, the noncorresponding data are a symptom of an underlying cause as yet hidden. The debugging process attempts to match symptom with cause, thereby leading to error correction.

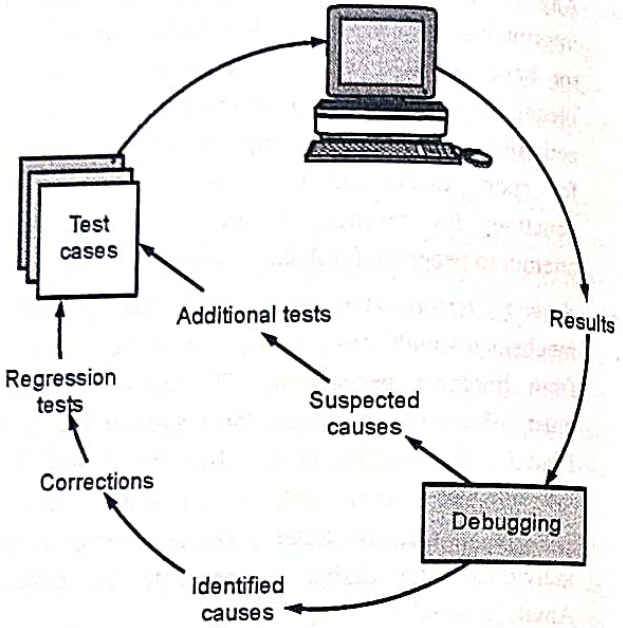


Fig. 5.7.5 : The Debugging process

- The debugging process will usually have one of two outcomes: (1) the cause will be found and corrected or (2) the cause will not be found. In the latter case, the person performing debugging may suspect a cause, design a test case to help validate that suspicion, and work toward error correction in an iterative fashion. Why is debugging so difficult? In all likelihood, human psychology has more to do with an answer than software technology. However, a few characteristics of bugs provide some clues:

1. The symptom and the cause may be geographically remote. That is, the symptom may appear in one part of a program, while the cause may actually be located at a site that is far removed. Highly coupled components (Chapter 8) exacerbate this situation.

2. The symptom may disappear (temporarily) when another error is corrected.
3. The symptom may actually be caused by nonerrors (e.g., round-off inaccuracies).
4. The symptom may be caused by human error that is not easily traced.
5. The symptom may be a result of timing problems, rather than processing problems.
6. It may be difficult to accurately reproduce input conditions (e.g., a real-time application in which input ordering is indeterminate).
7. The symptom may be intermittent. This is particularly common in embedded systems that couple hardware and software inextricably.
8. The symptom may be due to causes that are distributed across a number of tasks running on different processors. During debugging, you'll encounter errors that range from mildly annoying (e.g., an incorrect output format) to catastrophic (e.g., the system fails, causing serious economic or physical damage). As the consequences of an error increase, the , amount of pressure to find the cause also increases. Often, pressure forces some software developers to fix one error and at the same time introduce two more.

Psychological Considerations

- Unfortunately, there appears to be some evidence that debugging prowess is an innate human trait. Some people are good at it and others aren't.
- Although experimental evidence on debugging is open to many interpretations, large variances in debugging ability have been reported for programmers with the same education and experience.
- Debugging is one of the more frustrating parts of programming. It has elements of problem solving or brain teasers, coupled with the annoying recognition that you have made a mistake.
- Heightened anxiety and the unwillingness to accept the possibility of errors increases the task difficulty. Fortunately, there is a great sigh of relief and a lessening of tension when the bug is ultimately corrected.

Debugging Strategies

- Regardless of the approach that is taken, debugging has one overriding objective— to find and correct the cause of a software error or defect. The objective is realized by a combination of systematic evaluation, intuition, and luck.
- Debugging is a straightforward application of the scientific method that has been developed over 2,500 years. The basis of debugging is to locate the problem's source [the cause] by binary partitioning, through working hypotheses that predict new values to be examined.
- Take a simple non-software example: A lamp in my house does not work. If nothing in the house works, the cause must be in the main circuit breaker or outside; I look around to see whether the neighborhood is blacked out. I plug the suspect lamp into a working socket and a working appliance into the suspect circuit. So goes the alternation of hypothesis and test.
- In general, three debugging strategies have been proposed : (1) brute force, (2) backtracking, and (3) cause elimination. Each of these strategies can be conducted manually, but modern debugging tools can make the process much more effective.
- **Debugging tactics.** The *brute force* category of debugging is probably the most common and least efficient method for isolating the cause of a software error. You apply brute force debugging methods when all else fails. Using a "let the computer find the error" philosophy, memory dumps are taken, run-time traces are invoked, and the program is loaded with output statements. You hope that somewhere in the morass of information that is produced you'll find a clue that can lead to the cause of an error. Although the mass of information produced may ultimately lead to success, it more frequently leads to wasted effort and time. Thought must be expended first!
- *Backtracking* is a fairly common debugging approach that can be used successfully in small programs. Beginning at the site where a symptom has been uncovered, the source code is traced backward (manually) until the cause is found. Unfortunately, as the number of source lines increases, the number of potential backward paths may become unmanageably large.

- The third approach to debugging—*cause elimination*—is manifested by induction or deduction and introduces the concept of binary partitioning. Data related to the error occurrence are organized to isolate potential causes.
- A "cause hypothesis" is devised and the a forementioned data are used to prove or disprove the hypothesis. Alternatively, a list of all possible causes is developed and tests are conducted to eliminate each. If initial tests indicate that a particular cause hypothesis shows promise, data are refined in an attempt to isolate the bug.
- **Automated debugging.** Each of these debugging approaches can be supplemented with debugging tools that can provide you with semiautomated support as debugging strategies are attempted. Integrated development environments (IDEs) provide a way to capture some of the languagespecific predetermined errors (e.g., missing end-of-statement characters, undefined variables, and so on) without requiring compilation." A wide variety of
- bugging compilers, dynamic debugging aids ("tracers"), automatic test-case generators, and cross-reference mapping tools are available. However, tools are not a substitute for careful evaluation based on a complete design model and clear source code.

Correcting the Error

Once a bug has been found, it must be corrected. But, as we have already noted, the correction of a bug can introduce other errors and therefore do more three simple questions that you should ask before making the "correction" that removes the cause of a bug:

1. *Is the cause of the bug reproduced in another part of the program?* In many situations, a program defect is caused by an erroneous pattern of logic that may be reproduced elsewhere. Explicit consideration of the logical pattern may result in the discovery of other errors.
2. *What "next bug" might be introduced by the fix I'm about to make?* Before the correction is made, the source code (or, better, the design) should be evaluated to assess coupling of logic and data structures. If the correction is to be made in a highly coupled section of the program, special care must be taken when any change is made.
3. *What could we have done to prevent this bug in the first place?* This question is the first step toward establishing a statistical software quality assurance approach . If you correct the process as well as the product, the bug will be removed from the current program and may be eliminated from all future programs.

Software maintenance

GQ. Write in details about software maintenance.

- It begins almost immediately. Software is released to end users, and within days, bug reports filter back to the software engineering organization. Within weeks, one class of users indicates that the software must be changed so that it can accommodate the special needs of their environment. And within months, another corporate group who wanted nothing to do with the software when it was released now recognizes that it may provide them with unexpected benefit. They'll need a few enhancements to make it work in their world.
- The challenge of software maintenance has begun. You're faced with a growing queue of bug fixes, adaptation requests, and outright enhancements that must be planned, scheduled, and ultimately accomplished. Before long, the queue has grown long and the work it implies threatens to overwhelm the available resources.
- As time passes, your organization finds that it's spending more money and time maintaining existing programs than it is engineering new applications. In fact, it's not unusual for a software organization to expend as much as 60 to 70 percent of all resources on software maintenance.
- You may ask why so much maintenance is required and why so much effort is expended.
- Much of the software we depend on today is on average 10 to 15 years old. Even when these programs were created using the best design and coding techniques known at the time [and most were not], they were created when program size and storage space were principle concerns.

- They were then migrated to new platforms, adjusted for changes in machine and operating system technology and enhanced to meet new user needs—all without enough regard to overall architecture.
- The result is the poorly designed structures, poor coding, poor logic, and poor documentation of the software systems we are now called on to keep running.
- Another reason for the software maintenance problem is the mobility of software people. It is likely that the software team (or person) that did the original work is no longer around. Worse, other generations of software people have modified the system and moved on. And today, there may be no one left who has any direct knowledge of the legacy system.
- The ubiquitous nature of change underlies all software work. Change is inevitable when computer-based systems are built; therefore, you must develop mechanisms for evaluating, controlling, and making modifications. Throughout this book, I've emphasized the importance of understanding the problem (analysis) and developing a well-structured solution (design).
- In essence, *maintainability* is a qualitative indication of the ease with which existing software can be corrected, adapted, or enhanced. Much of what software engineering is about is building systems that exhibit high maintainability. But what is maintainability? Maintainable software exhibits effective modularity. It makes use of design patterns that allow ease of understanding.
- It has been constructed using well-defined coding standards and conventions, leading to source code that is self-documenting and understandable. It has undergone a variety of quality assurance techniques that have uncovered potential maintenance problems before the software is released.
- It has been created by software engineers who recognize that they may not be around when changes must be made. Therefore, the design and implementation of the software must "assist" the person who is making the change.

Software supportability

GQ. What is software supportability? Explain in details

- In order to effectively support industry-grade software, your organization (or its designee) must be capable of making the corrections, adaptations, and enhancements that are part of the maintenance activity. But in addition, the organization must provide other important support activities that include ongoing operational support, end-user support, and reengineering activities over the complete life cycle of the software.
- A reasonable definition of *software supportability* is. . . the capability of supporting a software system over its whole product life. This implies satisfying any necessary needs or requirements, but also the provision of equipment, support infrastructure, additional software, facilities, manpower, or any other resource required to maintain the software operational and capable of satisfying its function.
- In essence, supportability is one of many quality factors that should be considered during the analysis and design actions that are part of the software process.
- It should be addressed as part of the requirements model (or specification) and considered as the design evolves and construction commences. For example, the need to "antibug" software at the component and code level has been discussed earlier in the book.
- The software should contain facilities to assist support personnel when a defect is encountered in the operational environment (and make no mistake, defects *will* be encountered). In addition, support personnel should have access to a database that contains records of all defects that have already been encountered—their characteristics, cause, and cure. This will enable support personnel to examine "similar" defects and may provide a means for more rapid diagnosis and correction.
- Although defects encountered in an application are a critical support issue, supportability also demands that resources be provided to support day-to-day end-user issues. The job of end-user support personnel is to answer user queries about the installation, operation, and use of the application.

Reengineering

GQ. Explain Business Process Reengineering.

GQ. Explain BPR Model .

GQ. Discuss about Software Reengineering Process Model

GQ. Explain Software Reengineering Activities

GQ. What is reverse Engineering and forward engineering ?

- It is time to stop paving the cow paths. Instead of embedding outdated processes in silicon and software, we should obliterate them and start over. We should "reengineer" our businesses: use the power of modern information technology to radically redesign our business processes in order to achieve dramatic improvements in their performance.
- Every company operates according to a great many unarticulated rules. . . . Reengineering strives to break away from the old rules about how we organize and conduct our business.
- Like all revolutions, Hammer's call to arms resulted in both positive and negative changes. During the 1990s, some companies made a legitimate effort to reengineer, and the results led to improved competitiveness. Others relied solely on downsizing and outsourcing (instead of reengineering) to improve their bottom line. "Mean" organizations with little potential for future growth often resulted .
- By the end of the first decade of the twenty-first century, the hype associated with reengineering waned, but the process itself continues in companies large and small. The nexus between business reengineering and software engineering lies in a "system view."
- Software is often the realization of the business rules that Hammer discusses. Today, major companies have tens of thousands of computer programs that support the "old business rules."
- As managers work to modify the rules to achieve greater effectiveness and competitiveness, software must keep pace. In some cases, this means the creation of major new computer-based systems.2 But in many others, it means the modification or rebuilding of existing applications.
- In the sections that follow, I examine reengineering in a top-down manner, beginning with a brief overview of business process reengineering and proceeding to a more detailed discussion of the technical activities that occur when software is reengineered.

Business Process Reengineering

Business process reengineering (BPR) extends far beyond the scope of information technologies and software engineering.

❒ **Definition**

"the search for, and the implementation of, radical change in business process to achieve breakthrough results." But how is the search conducted, and how is the implementation achieved? More important, how can we ensure that the "radical change" suggested will in fact lead to "breakthrough results" instead of organizational chaos?

Business Processes

- A business process is "a set of logically related tasks performed to achieve a defined business outcome". Within the business process, people, equipment, material resources, and business procedures are combined to produce a specified result.
- Examples of business processes include designing a new product, purchasing services and supplies, hiring a new employee, and paying suppliers. Each demands a set of tasks, and each draws on diverse resources within the business.
- Every business process has a defined customer—a person or group that receives the outcome (e.g., an idea, a report, a design, a service, a product). In addition, business processes cross organizational boundaries. They require that different organizational groups participate in the "logically related tasks" that define the process.
- Every system is actually a hierarchy of subsystems. A business is no exception. The overall business is segmented in the following manner:

The business ã business systems ã business processes ã business subprocesses

- Each business system (also called *business function*) is composed of one or more business processes, and each business process is defined by a set of subprocesses. BPR can be applied at any level of the hierarchy, but as the scope of BPR broadens (i.e., as we move upward in the hierarchy), the risks associated with BPR grow

dramatically. For this reason, most BPR efforts focus on individual processes or ubprocesses.

A BPR Model

- Like most engineering activities, business process reengineering is iterative. Business goals and the processes that achieve them must be adapted to a changing business environment.
- For this reason, there is no start and end to BPR—it is an evolutionary process. A model for business process reengineering is depicted in Fig. 5.7.6.
- **The model defines six activities :** Business definition. Business goals are identified within the context of four key drivers: cost reduction, time reduction, quality improvement, and personnel development and empowerment. Goals may be defined at the business level or for a specific component of the business.

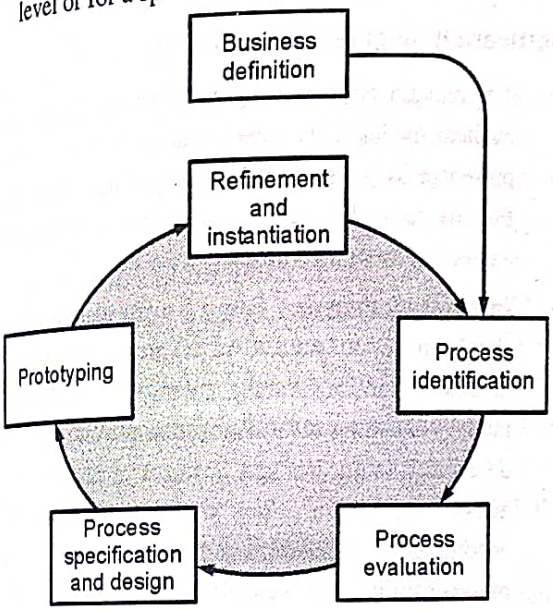


Fig. 5.7.6 : BPR Model

- **Process identification.** Processes that are critical to achieving the goals defined in the business definition are identified. They may then be ranked by importance, by need for change, or in any other way that is appropriate for the reengineering activity.
- **Process evaluation.** The existing process is thoroughly analyzed and measured. Process tasks are identified; the costs and time consumed by process tasks are noted; and quality/performance problems are isolated.
- **Process specification and design.** Based on information obtained during the first three BPR activities, use cases (Chapters 5 and 6) are prepared for each process that is to be redesigned. Within the context of BPR, use cases identify a scenario that delivers some outcome to a customer. With the use case as the specification of the process, a new set of tasks are designed for the process.
- **Prototyping.** A redesigned business process must be prototyped before it is fully integrated into the business. This activity "tests" the process so that refinements can be made.
- **Refinement and instantiation.** Based on feedback from the prototype, the business process is refined and then instantiated within a business system. These BPR activities are sometimes used in conjunction with workflow analysis tools. The intent of these tools is to build a model of existing workflow in an effort to better analyze existing processes.

Software reengineering

- The scenario is all too common: An application has served the business needs of a company for 10 or 15 years. During that time it has been corrected, adapted, and enhanced many times. People approached this work with the best intentions, but good software engineering practices were always shunted to the side (due to the press of other matters).
- Now the application is unstable. It still works, but every time a change is attempted, unexpected and serious side effects occur. Yet the application must continue to evolve. What to do? Unmaintainable software is not a new problem. In fact, the broadening emphasis on software reengineering has been spawned by software maintenance problems that have been building for more than four decades.

A Software Reengineering Process Model

- Reengineering takes time, it costs significant amounts of money, and it absorbs resources that might be otherwise occupied on immediate concerns. For all of these reasons, reengineering is not accomplished in a few months or even a few years.
- Reengineering of information systems is an activity that will absorb information technology resources for many years. That's why every organization needs a pragmatic strategy for software reengineering.

- A workable strategy is encompassed in a reengineering process model. I'll discuss the model later in this section, but first, some basic principles. Reengineering is a rebuilding activity.
- To better understand it, consider an analogous activity: the rebuilding of a house. Consider the following situation. You've purchased a house in another state. You've never actually seen the property, but you acquired it at an amazingly low price, with the warning that it might have to be completely rebuilt. How would you proceed?
- Before you can start rebuilding, it would seem reasonable to inspect the house. To determine whether it is in need of rebuilding, you (or a professional inspector) would create a list of criteria so that your inspection would be systematic.
- Before you tear down and rebuild the entire house, be sure that the structure is weak. If the house is structurally sound, it may be possible to "remodel" without rebuilding (at much lower cost and in much less time).
- Before you start rebuilding be sure you understand how the original was built. Take a peek behind the walls. Understand the wiring, the plumbing, and the structural internals. Even if you trash them all, the insight you'll gain will serve you well when you start construction.
- If you begin to rebuild, use only the most modern, long-lasting materials. This may cost a bit more now, but it will help you to avoid expensive and time-consuming maintenance later.
- If you decide to rebuild, be disciplined about it. Use practices that will result in high quality—today and in the future. Although these principles focus on the rebuilding of a house, they apply equally well to the reengineering of computer-based systems and applications.
- To implement these principles, you can use a software reengineering process model that defines six activities, shown in Fig. 5.7.7. In some cases, these activities occur in a linear sequence, but this is not always the case. For example, it may be that reverse engineering (understanding the internal workings of a program) may have to occur before document restructuring can commence.

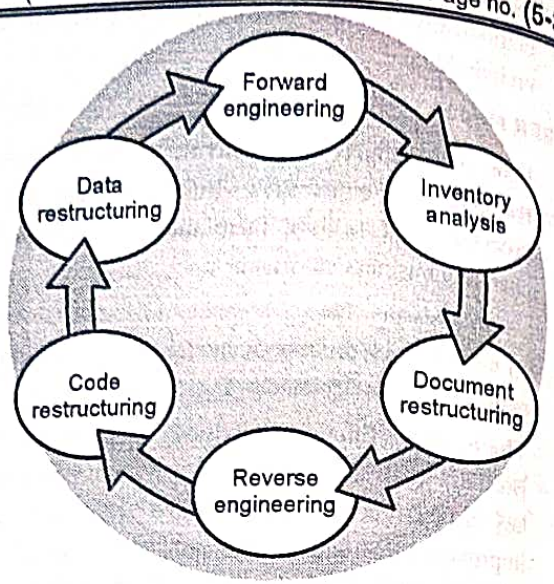


Fig. 5.7.7 : Software Reengineering process model

Software Reengineering Activities

- The reengineering paradigm shown in Fig. 5.7.7 is a cyclical model. This means that each of the activities presented as a part of the paradigm may be revisited. For any particular cycle, the process can terminate after any one of these activities.
- **Inventory analysis.** Every software organization should have an inventory of all applications. The inventory can be nothing more than a spreadsheet model containing information that provides a detailed description (e.g., size, age, business criticality) of every active application. By sorting this information according to business criticality, longevity, current maintainability and supportability, and other locally important criteria, candidates for reengineering appear. Resources can then be allocated to candidate applications for reengineering work.
- It is important to note that the inventory should be revisited on a regular cycle. The status of applications (e.g., business criticality) can change as a function of time, and as a result, priorities for reengineering will shift.
- **Document restructuring.** Weak documentation is the trademark of many legacy systems. But what can you do about it? What are your options?

1. *Creating documentation is far too time consuming.* If the system works, you may choose to live with what you have. In some cases, this is the correct approach. It

is not possible to re-create documentation for hundreds of computer programs. If a program is relatively static, is coming to the end of its useful life, and is unlikely to undergo significant change, let it be!

2. *Documentation must be updated, but your organization has limited resources.* You'll use a "document when touched" approach. It may not be necessary to fully redocument an application. Rather, those portions of the system that are currently undergoing change are fully documented. Over time, a collection of useful and relevant documentation will evolve.
3. *The system is business critical and must be fully redocumented.* Even in this case, an intelligent approach is to pare documentation to an essential minimum.

- Each of these options is viable. Your software organization must choose the one that is most appropriate for each case.
- **Reverse engineering.** The term *reverse engineering* has its origins in the hardware world. A company disassembles a competitive hardware product in an effort to understand its competitor's design and manufacturing "secrets." These secrets could be easily understood if the competitor's design and manufacturing specifications were obtained. But these documents are proprietary and unavailable to the company doing the reverse engineering. In essence, successful reverse engineering derives one or more design and manufacturing specifications for a product by examining actual specimens of the product.
- Reverse engineering for software is quite similar. In most cases, however, the program to be reverse engineered is not a competitor's. Rather, it is the company's own work (often done many years earlier).
- The "secrets" to be understood are obscure because no specification was ever developed. Therefore, reverse engineering for software is the process of analyzing a program in an effort to create a representation of the program at a higher level of abstraction than source code.
- Reverse engineering is a process of *design recovery.* Reverse engineering tools extract data, architectural, and procedural design information from an existing program.
- **Code restructuring.** The most common type of reengineering (actually, the use of the term reengineering is questionable in this case) is *code restructuring.* Some legacy systems have a relatively solid program architecture, but individual modules were coded in a way that makes them difficult to understand, test, and maintain. In such cases, the code within the suspect modules can be restructured.
- To accomplish this activity, the source code is analyzed using a restructuring tool. Violations of structured programming constructs are noted and code is then restructured (this can be done automatically) or even rewritten in a more modern programming language.
- The resultant restructured code is reviewed and tested to ensure that no anomalies have been introduced. Internal code documentation is updated.
- **Data restructuring.** A program with weak data architecture will be difficult to adapt and enhance. In fact, for many applications, information architecture has more to do with the long-term viability of a program than the source code itself.
- Unlike code restructuring, which occurs at a relatively low level of abstraction, data restructuring is a full-scale reengineering activity. In most cases, data restructuring begins with a reverse engineering activity. Current data architecture is dissected, and necessary data models are defined.
- Data objects and attributes are identified, and existing data structures are reviewed for quality. When data structure is weak (e.g., flat files are currently implemented, when a relational approach would greatly simplify processing), the data are reengineered. Because data architecture has a strong influence on program architecture and the algorithms that populate it, changes to the data will invariably result in either architectural or code-level changes.
- **Forward engineering.** In an ideal world, applications would be rebuilt using an automated "reengineering engine." The old program would be fed into the engine, analyzed, restructured, and then regenerated in a form that exhibited the best aspects of software quality.
- In the short term, it is unlikely that such an "engine" will appear, but vendors have introduced tools that provide a limited subset of these capabilities that addresses specific application domains (e.g.,

applications that are implemented using a specific database system). More important, these reengineering tools are becoming increasingly more sophisticated.

- Forward engineering not only recovers design information from existing software but uses this information to alter or reconstitute the existing system in an effort to improve its overall quality. In most cases, reengineered software reimplements the function of the existing system and also adds new functions and/or improves overall performance.

Reverse engineering

- Reverse engineering conjures an image of the "magic slot." You feed a haphazardly designed, undocumented source file into the slot and out the other end comes a complete design description (and full documentation) for the computer program.
- Unfortunately, the magic slot doesn't exist. Reverse engineering can extract design information from source code, but the abstraction level, the completeness of the documentation, the degree to which tools and a human analyst work together, and the directionality of the process are highly variable.
- The *abstraction level* of a reverse engineering process and the tools used to effect it refers to the sophistication of the design information that can be extracted from source code. Ideally, the abstraction level should be as high as possible. That is, the reverse engineering process should be capable of deriving procedural design representations (a low-level abstraction), program and data structure information
- (a somewhat higher level of abstraction), object models, data and/or control flow models (a relatively high level of abstraction), and entity relationship models (a high level of abstraction).
- As the abstraction level increases, you are provided with information that will allow easier understanding of the program. The *completeness* of a reverse engineering process refers to the level of detail that is provided at an abstraction level. In most cases, the completeness decreases as the abstraction level increases. For example, given a source code listing, it is relatively easy to develop a complete procedural design representation.
- Simple architectural design representations may also be derived, but it is far more difficult to develop a complete set of UML diagrams or models.
- Completeness improves in direct proportion to the amount of analysis performed by the person doing reverse engineering. *Interactivity* refers to the degree to which the human is "integrated" with automated tools to create an effective reverse engineering process.
- In most cases, as the abstraction level increases, interactivity must increase or completeness will suffer.
- If the *directionality* of the reverse engineering process is one-way, all information extracted from the source code is provided to the software engineer who can then use it during any maintenance activity. If directionality is two-way, the information is fed to a reengineering tool that attempts to restructure or regenerate the old program.

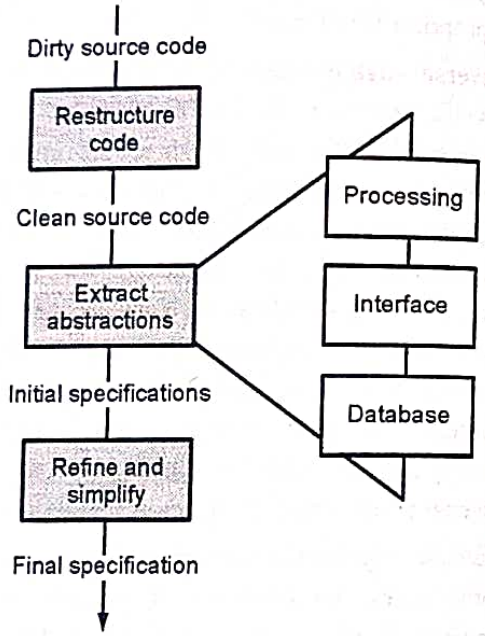


Fig. 5.7.8 : The reverse engineering process

- The reverse engineering process is represented in Fig. 5.7.8. Before reverse engineering activities can commence, unstructured ("dirty") source code is restructured so that it contains only the structured programming constructs.
- This makes the source code easier to read and provides the basis for all the subsequent reverse engineering activities.
- The core of reverse engineering is an activity called *extract abstractions*. You must evaluate the old

program and from the (often undocumented) source code, develop a meaningful specification of the processing that is performed, the user interface that is applied, and the program data structures or database that is used.

Reverse Engineering to Understand Data

- Reverse engineering of data occurs at different levels of abstraction and is often the first reengineering task. At the program level, internal program data structures must often be reverse engineered as part of an overall reengineering effort.
- At the system level, global data structures (e.g., files, databases) are often reengineered to accommodate new database management paradigms (e.g., the move from flat file to relational or object-oriented database systems). Reverse engineering of the current global data structures sets the stage for the introduction of a new systemwide database.
- **Internal data structures.** Reverse engineering techniques for internal program data focus on the definition of classes of objects. This is accomplished by examining the program code with the intent of grouping related program variables.
- In many cases, the data organization within the code identifies abstract data types. For example, record structures, files, lists, and other data structures often provide an initial indicator of classes.
- **Database structure.** Regardless of its logical organization and physical structure, a database allows the definition of data objects and supports some method for establishing relationships among the objects. Therefore, reengineering one database schema into another requires an understanding of existing objects and their relationships.
- The following steps may be used to define the existing data model as a precursor to reengineering a new database model: (1) build an initial object model, (2) determine candidate keys (the attributes are examined to determine whether they are used to point to another record or table; those that serve as pointers become candidate keys), (3) refine the tentative classes, (4) define generalizations, and (5) discover associations using techniques that are analogous to the CRC approach. Once information defined in the preceding steps is known, a series of transformations can be applied to map the old database structure into a new database structure.

Reverse Engineering to Understand Processing

- Reverse engineering to understand processing begins with an attempt to understand and then extract procedural abstractions represented by the source code.
- To understand procedural abstractions, the code is analyzed at varying levels of abstraction: system, program, component, pattern, and statement. The overall functionality of the entire application system must be understood before more detailed reverse engineering work occurs. This establishes a context for further analysis and provides insight into interoperability issues among applications within the system.
- Each of the programs that make up the application system represents a functional abstraction at a high level of detail. A block diagram, representing the interaction between these functional abstractions, is created.
- Each component performs some subfunction and represents a defined procedural abstraction. A processing narrative for each component is developed. In some situations, system, program, and component specifications already exist.
- When this is the case, the specifications are reviewed for conformance to existing code. Things become more complex when the code inside a component is considered. You should look for sections of code that represent generic procedural patterns. In almost every component, a section of code prepares data for processing (within the module), a different section of code does the processing, and another section of code prepares the results of processing for export from the component.
- Within each of these sections, you can encounter smaller patterns; for example, data validation and bounds checking often occur within the section of code that prepares data for processing.
- For large systems, reverse engineering is generally accomplished using a semiautomated approach. Automated tools can be used to help you understand the semantics of existing code. The output of this process is then passed to restructuring and forward

engineering tools to complete the reengineering process.

Reverse Engineering User Interfaces

- Sophisticated GUIs have become de rigueur for computer-based products and systems of every type. Therefore, the redevelopment of user interfaces has become one of the most common types of reengineering activity. But before a user interface can be rebuilt, reverse engineering should occur.
- To fully understand an existing user interface, the structure and behavior of the interface must be specified. Three basic questions that must be answered as reverse engineering of the UI commences:
  - What are the basic actions (e.g., keystrokes and mouse clicks) that the interface must process?
  - What is a compact description of the behavioral response of the system to mthese actions?
  - What is meant by a "replacement," or more precisely, what concept of equivalence of interfaces is relevant here?
  - Behavioral modeling notation can provide a means for developing answers to the first two questions. Much of the information necessary to create a behavioral model can be obtained by observing the external manifestation of the existing interface. But additional information necessary to create the behavioral model must be extracted from the code.
- It is important to note that a replacement GUI may not mirror the old interface exactly (in fact, it may be radically different). It is often worthwhile to develop a new interaction metaphor. For example, an old UI requests that a user provide a scale factor (ranging from 1 to 10) to shrink or magnify a graphical image. A reengineered GUI might use a slide-bar and mouse to accomplish the same function.

Restructuring

- Software restructuring modifies source code and/or data in an effort to make it amenable to future changes. In general, restructuring does not modify the overall program architecture.
- It tends to focus on the design details of individual modules and on local data structures defined within modules. If the restructuring effort extends beyond module boundaries and encompasses the software architecture, restructuring becomes forward engineering.
- Restructuring occurs when the basic architecture of an application is solid, even though technical internals need work. It is initiated when major parts of the software are serviceable and only a subset of all modules and data need extensive modification.

Code Restructuring

- *Code restructuring* is performed to yield a design that produces the same function but with higher quality than the original program. In general, code restructuring techniques model program logic using Boolean algebra and then apply a series of transformation rules that yield restructured logic.
- The objective is to take "spaghetti-bowl" code and derive a procedural design that conforms to the structured programming philosophy. Other restructuring techniques have also been proposed for use with reengineering tools.
- A resource exchange diagram maps each program module and the resources (data types, procedures and variables) that are exchanged between it and other modules. By creating representations of resource flow, the program architecture can be restructured to achieve minimum coupling among modules.

Data Restructuring

- Before data restructuring can begin, a reverse engineering activity called *analysis of source code* should be conducted. All programming language statements that contain data definitions, file descriptions, I/O, and interface descriptions are evaluated.
- The intent is to extract data items and objects, to get information on data flow, and to understand the existing data structures that have been implemented. This activity is sometimes called *data analysis*.
- Once data analysis has been completed, *data redesign* commences. In its simplest form, a *data record standardization* step clarifies data definitions to achieve consistency among data item names or physical record formats within an existing data structure or file format.
- Another form of redesign, called *data name rationalization*, ensures that all data naming conventions conform to local standards and that aliases are eliminated as data flow through the system.
- When restructuring moves beyond standardization and rationalization, physical modifications to existing data structures are made to make the data design more effective. This may mean a translation from one file

format to another, or in some cases, translation from one type of database to another.

Forward engineering

- A program with control flow that is the graphic equivalent of a bowl of spaghetti, with "modules" that are 2000 statements long, with few meaningful comment lines in 290,000 source statements and no other documentation must be modified to accommodate changing user requirements. You have the following options:

1. You can struggle through modification after modification, fighting the ad hoc design and tangled source code to implement the necessary changes.
2. You can attempt to understand the broader inner workings of the program in an effort to make modifications more effectively.
3. You can redesign, recode, and test those portions of the software that require modification, applying a software engineering approach to all revised segments.
4. You can completely redesign, recode, and test the program, using reengineering tools to assist you in understanding the current design. There is no single "correct" option. Circumstances may dictate the first option even if the others are more desirable.

- Rather than waiting until a maintenance request is received, the development or support organization uses the results of inventory analysis to select a program that (1) will remain in use for a preselected number of years, (2) is currently being used successfully, and (3) is likely to undergo major modification or enhancement in the near future. Then, option 2, 3, or 4 is applied.
- At first glance, the suggestion that you redevelop a large program when a working version already exists may seem quite extravagant. Before passing judgment, consider the following points:

1. The cost to maintain one line of source code may be 20 to 40 times the cost of initial development of that line.
2. Redesign of the software architecture (program and/or data structure), using modern design concepts, can greatly facilitate future maintenance.
3. Because a prototype of the software already exists, development productivity should be much higher than average.
4. The user now has experience with the software. Therefore, new requirements and the direction of change can be ascertained with greater ease.
5. Automated tools for reengineering will facilitate some parts of the job.
6. A complete software configuration (documents, programs, and data) will nexist upon completion of preventive maintenance.

- A large in-house software developer (e.g., a business systems software development group for a large consumer products company) may have 500 to 2000 production programs within its domain of responsibility.
- These programs can be ranked by importance and then reviewed as candidates for forward engineering. The forward engineering process applies software engineering principles, concepts, and methods to re-create an existing application. In most cases, forward engineering does not simply create a modern equivalent of an older program. Rather, new user and technology requirements are integrated into the reengineering effort. The redeveloped program extends the capabilities of the older application.

Forward Engineering for Client-Server Architectures

- Over the past few decades, many mainframe applications have been reengineered to accommodate client-server architectures (including WebApps). In essence, centralized computing resources (including software) are distributed among many client platforms. Although a variety of different distributed environments can be designed, the typical mainframe application that is reengineered into a client-server architecture has the following features:
  - Application functionality migrates to each client computer.
  - New GUI interfaces are implemented at the client sites.
  - Database functions are allocated to the server.
  - Specialized functionality (e.g., compute-intensive analysis) may remain at the server site.
  - New communications, security, archiving, and control requirements must be established at both the client and server sites.
- It is important to note that the migration from mainframe to client-server computing requires both business and software reengineering. In addition, an

"enterprise network infrastructure" should be established.

- Reengineering for client-server applications begins with a thorough analysis of the business environment that encompasses the existing mainframe. Three layers of abstraction can be identified.
- The *database sits* at the foundation of a client-server architecture and manages transactions and queries from server applications. Yet these transactions and queries must be controlled within the context of a set of business rules (defined by an existing or reengineered business process). Client applications provide targeted functionality to the user community.
- The functions of the existing database management system and the data architecture of the existing database must be reverse engineered as a precursor to the redesign of the database foundation layer.
- In some cases a new data model is created. In every case, the client-server database is reengineered to ensure that transactions are executed in a consistent manner, that all updates are performed only by authorized users, that core business rules are enforced (e.g., before a vendor record is deleted, the server ensures that no related accounts payable, contracts, or communications exist for that vendor), that queries can be accommodated efficiently, and that full archiving capability has been established.
- The business rules layer represents software resident at both the client and the server. This software performs control and coordination tasks to ensure that transactions and queries between the client application and the database conform to the established business process.
- The client applications layer implements business functions that are required by specific groups of end users. In many instances, a mainframe application is segmented into a number of smaller, reengineered desktop applications. Communication among the desktop applications (when necessary) is controlled by the business rules layer.

Forward Engineering for Object-Oriented Architectures

- Object-oriented software engineering has become the development paradigm of choice for many software organizations. But what about existing applications that were developed using conventional methods? In some cases, the answer is to leave such applications "as is." In others, older applications must be reengineered so that they can be easily integrated into large, object-oriented systems.
- Reengineering conventional software into an object-oriented implementation uses many of the same techniques. First, the existing software is reverse engineered so that appropriate data, functional, and behavioral models can be created.
- If the reengineered system extends the functionality or behavior of the original application, use cases are created. The data models created during reverse engineering are then used in conjunction with CRC modeling to establish the basis for the definition of classes.
- Class hierarchies, object-relationship models, object-behavior models, and subsystems are defined, and object-oriented design commences. As object-oriented forward engineering progresses from analysis to design, a CBSE process model can be invoked. If the existing application resides within a domain that is already populated by many object-oriented applications, it is likely that a robust component library exists and can be used during forward engineering.
- For those classes that must be engineered from scratch, it may be possible to reuse algorithms and data structures from the existing conventional application. However, these must be redesigned to conform to the object-oriented architecture.

Chapter Ends...
□□□

MODULE VI

CHAPTER 6

IT Project Management and Project Scheduling

Syllabus

Introduction, 4 P's, W5HH Principle, Need for Project Management, Project Life cycle and ITPM, Project Feasibility, RFP, PMBOK Knowledge areas, Business Case, Project Planning, Project Charter and Project Scope. Project Scheduling: Defining a Task Set for the Software Project, Timeline chartsWBS, Developing the Project Schedule, Network Diagrams (AON, AOA), CPM and PERT, Gantt Chart, Tracking the Schedule, Earned Value Analysis.

6.1 Introduction to Project Management 6-3

GQ. Why is an increased need felt for effective and efficient project management? 6-3

6.2 What is a Project ? 6-4

GQ. What is a Project? Discuss the salient features of an IT project. 6-4

6.2.1 Attributes of a Project 6-4

GQ. State the attributes of a project. 6-4

6.3 4P's 6-5

GQ. Enumerate the 4P's of project management 6-5

6.3.1 People 6-5

6.3.2 Product 6-5

6.3.3 Process 6-5

6.3.3(A) Project Management Process Groups 6-6

GQ. Briefly describe the project management process groups 6-6

6.3.4 Project 6-8

GQ. State the five most important goals of a project 6-8

6.4 W5HH 6-9

GQ. State W5HH principle 6-9

6.5 Need for Project Management 6-9

6.6 Project Lifecycle 6-10

GQ. Briefly describe the various phases of a generic project lifecycle. 6-10

6.7 IT Project Methodology 6-12

6.7.1 Project Management Methodology 6-12

6.7.2 Methodology Development 6-13

6.7.3 Phases in the Project Management Methodology 6-14

GQ. Describe the phases in IT Project Management Methodology. 6-14

6.8 Project Feasibility 6-16

GQ. Why is it essential to undertake a feasibility study before project initiation? 6-16

6.9 RFP 6-17

GQ. What is RFP? 6-17

6.10 PMBOK Knowledge Areas 6-18

GQ. State the areas covered under project feasibility. 6-18

GQ. Enumerate PMBOK Knowledge Areas. 6-18

6.10.1 PMBOK Knowledge Areas 6-19
6.11 Business Case 6-20
GQ. What is a Business Case? State the steps in developing the business case. 6-20
6.11.1 Developing the Business Case 6-20
6.12 Project Planning 6-22
GQ. Write short note on: Project Planning. 6-22
6.13 Project Charter and Project Scope 6-25
GQ. Write short note on : Project Charter and Project Scope. 6-25
6.14 Work Breakdown Structure 6-28
GQ. "WBS assists in accomplishing project objectives and create required project products". Explain. ..6-28
GQ. Write short notes on :1. Delivery Definition Table (DDT) 2. Project Scope Statement 6-28
6.14.1 Benefits of WBS 6-29
GQ. State the benefits of WBS. 6-29
6.14.2 Developing the WBS 6-29
6.14.3 WBS Diagram and Numbering System 6-30
6.15 Developing the Project Schedule 6-31
GQ. Briefly describe the process of developing the project schedule. 6-31
6.15.1 Project Activities, Attributes and Milestones 6-33
6.16 Project Schedules 6-35
GQ. What purpose does the project schedule serve? 6-35
GQ. Enumerate the tools available with the project manager in identifying risks 6-35
6.16.1 Inputs to Project Schedule Development 6-36
6.16.2 Schedule Development – Tools and Techniques 6-37
6.16.3 Outputs of Project Schedule Development 6-37
6.17 Formulating a Network Model 6-38
6.17.1 Evolution of Network Planning 6-40
6.17.2 Advantages of Network Planning 6-40
6.17.3 Steps in drawing a Network - Activity-On-Arrow 6-40
6.17.4 Events, Activities and their Inter-relationship in Network Diagram 6-41
6.17.5 Numbering the Events 6-43
6.17.6 Use of Dummy Arrows for Dummy Activities 6-43
6.17.7 Tips for Drawing Network Diagrams 6-44
6.17.8 Drawing Networks – Illustrative Examples 6-45
6.18 Network Analysis 6-47
6.18.1 Scheduling Activities 6-47
6.18.2 Critical Path 6-49
GQ. What is a critical path? Why is the longest path the critical path? 6-49
6.18.3 Type of Floats 6-51
GQ. What is a float ? State the types of float. 6-51
6.19 Introduction to Critical Path Method (CPM) and Program Evaluation Review Technique (PERT) 6-64
GQ. Explain the three time estimates required for each activity under PERT. 6-64
6.19.1 Significance of Using CPM/PERT 6-65
6.19.2 Basic Difference between CPM and PERT 6-65
GQ. Differentiate between CPM and PERT. 6-65
6.20 Precedence Network Analysis (AON) 6-66
GQ. Briefly describe Precedence Diagram Method. ..6-66
6.20.1 Need for Precedence Diagramming 6-66
6.20.2 History of Precedence Diagramming Method 6-67
6.20.3 Notations used in Precedence Diagramming Method 6-67
6.20.4 Types of Logical Relationships or Dependencies in Precedence Network Analysis 6-67
6.20.5 Precedence Network Representations 6-68
6.20.6 Network Calculations 6-69
6.20.7 Illustrative Examples 6-70
6.21 Program Evaluation Review Technique (PERT) 6-73
GQ. "PERT is an event oriented technique." Explain. 6-73
6.22 Network Planning Models 6-76
6.22.1 Gantt Chart 6-76
GQ. What is Gantt chart? How is it created? 6-76
GQ. State the advantages and limitations of Gantt chart. 6-76
6.22.2 Milestone Chart 6-78
GQ. Write short note on : Milestone Chart. 6-78
6.23 Tracking the Schedule 6-78
6.24 Earned Value Analysis 6-79
• **Chapter Ends** 6-79

Syllabus Topic : Introduction

6.1 INTRODUCTION TO PROJECT MANAGEMENT

Q. Why is an increased need felt for effective and efficient project management?

You may feel that why is such a fuss being made of the management of projects and the need for the adoption of proper project management tools and techniques. With increased competition and globalization a failed project could have more wide ranging implications that could affect the overall prospects of the organization. Project managers are expected to deliver results on time, within budget, while adhering to quality parameters and ensuring that the project results cover the entire scope of the project. Hence, a disciplined project management process is important to any project.

Following are a few reasons why an increased need is felt for effective and efficient project management:

1. **To control scope of project and manage change** - Although the project deliverables are defined at the outset of the project, small changes in project deliverables are common. These changes are demanded by customers, stakeholders, management, suppliers or the project team itself. Individually, these change demands may seem acceptable and manageable, but collectively these change demands can lead to a significant expansion in the project scope and can lead to an overrun in schedule and budget. However, with project management, if the manager effectively manages the scope of their project, they have a better chance of effectively managing project resources and change.

   To deliver projects on time and within budget - The project management process includes cost calculations such as return on investments (ROI). Once ROI is established it is for the project manager to ensure that the project schedule and budget are adhered to else the project will fail to deliver the expected results.

   To ensure the focus of the project team – It is common for the project team to drift from the main tasks and spend unnecessary longer time on other tasks. Hence, it is the responsibility of the project manager to ensure that the project team focuses on the right tasks by using a clear and concise project charter and that there are no interferences.

4. **To collect user requirement from disparate sources** – The project manager at the initiation phase should collect user requirement, project constraints and conduct a feasibility study to build a strong business case justification. The primary advantage of collecting input from various sources is that the project manager is able to avoid future dissent from users and is able to communicate project benefits.

5. **To define the critical path to optimally complete the project** - Every project is made up of connected activities each having their individual constraints. By using the critical path method technique the project manager is able to identify the critical path and thus ensure the successful completion of the project.

6. **To provide a process for estimating project resources, time and costs** - Solid project management tools and techniques and past experience will enable the project manager to correctly estimate the project resources requirement as well as determine the time required for the completion of the project and the likely expenditure. Such, estimation at the initiation phase will ensure that the project receives adequate commitment from the management and its success is ensured.

7. **To communicate project progress, risks and changes** – The stakeholders of the project need to be kept updated on the project progress, hurdles encountered and changes incorporated. Every good project management plan has a communication plan that addresses communication issues, provides formats and lays process for execution of the communication plan.

8. **To explore project assumptions** – Although ample ground work is done for a project there are bound to be a few assumptions on which the project is based. Hence, a good project manager has to delve deeper into user requirements, project constraints and management expectations to understand the hidden project requirements. A

project based on too many assumptions could ruin its chances of success.

9. **To prepare for unexpected project issues** – Howsoever, one may be prepared there are bound to be a few issues which may suddenly surface. Hence, the project manager should always be prepared with an alternate plan.
10. **To document the knowledge gained from the project** – The last phase of the project involves the documentation of all that has been learnt at each phase in the project. These documented experiences provide guidance to other project manager in other projects.

6.2 WHAT IS A PROJECT ?

GQ. What is a Project? Discuss the salient features of an IT project.

- To study project management it is necessary to first understand the concept of a project.
- A project is defined as an undertaking of a non-routine, non-repetitive nature having prescribed objectives in terms of scope, time, quality and cost.
- A project is undertaken to create something new, unique, or to enhance an existing product, service, or system.
- A project is a temporary endeavor undertaken to create a unique product, service, or result.
- A project is an endeavor to accomplish specific objectives through a unique set of interrelated tasks and effective utilization of resources.
- IT projects involve hardware, software and networks to create product, service, or result. A few examples of IT projects are;
- A new feature is added to an existing software application.
- New system is developed to increase sales force productivity, improve customer relationship management and enhance supply chain management.
- A firm decides to implement an integrated ERP project to consolidate its information systems.
- Replacing the company's manual time keeping system with a web based system within a particular time frame.

6.2.1 Attributes of a Project

GQ. State the attributes of a project.

A few typical project attributes are as follows :

- **Purpose** - A project has a unique purpose and well defined objectives. It begins with rough ideas and an initial plan which is updated as more information emerges. Projects are undertaken to accomplish something of value to the company, maybe, a system or software. The sole purpose of undertaking a project is to produce a tangible product of value to the company.
- **Goal** - A goal drives a project. It is the sole motivating factor and defines each activity, task, work, schedule and budget of the project. It provides direction to the team. Hence, the project goal should be clearly defined, ambiguity in defining the project goal leads to a project with no end.
- **Time frame** - Since a project is a temporary endeavour it has to have a definitive start and end. The time frame for achieving the project goal is estimated based on the duration of the various project activities. The completion date of the project is set accordingly. However, for projects where the completion date is fixed, such as the Y2K problem, the starting date has to be set by working backwards.
- **Interrelated tasks and resources** - A project is composed of interrelated tasks and utilizes resources such as people, software, hardware and other assets of the company. The primary asset for an IT project is people who cross departmental and other boundaries to achieve this unique purpose. Most projects demand the service of people outside the organization such as consultants. Resources, however, are expensive and limited hence should be used effectively to meet project objectives. A project is unique and hence it is difficult to estimate the time, budget and resources requirement. Hence, a project involves uncertainty.
- **Ownership** - A project has various stakeholders but only one primary customer known as the project sponsor. The project sponsor is the "go to" entity for funds, direction and approvals regarding all project related matters. For the project manager the project sponsor is the owner of the project.

- **Risks and Assumptions** - Every project has certain risk element, the more complex and big the project greater is the risk. Risk is directly proportional to the size and complexity of the project. Risks can emanate from sources both internal and external to the project team. Internal risks could be the resignation of a key project member, while external risks could arise from vendors and suppliers of key hardware and software. Assumptions are those unknown variables which are used to estimate key facets of the project. Therefore, the project manager has to identify all the risks and assumptions that could impact the project.

Syllabus Topic : 4P's

6.3 4P'S

GQ. Enumerate the 4P's of project management.

Similar to any other engineering project the primary focus for effective IT project management are the 4 P's namely people, product, process and project.

6.3.1 People

- In software project management the people's component is probably the most crucial as they form the backbone of the project.
- Although, other resources such as materials, facilities, tools and equipments are also of vital importance, the primary resource in IT projects is its people. IT projects are completed by people and the project manager will need the right people to accomplish project objectives.
- The complexity of modern IT projects makes it necessary to employ the services of individuals with varying skills and knowledge. Following are the key players in an IT project.
  - **Project manager** - The project manager is the captain of the team and is responsible for the success or failure of the project. The project manager has to ensure that all development requirements and processes are in place and are carried out as per the specifications and quality parameters specified.
  - **Project sponsor** - The project sponsor could be the client or an organizational appointee who is the "go to" person for all project related resource requirement and direction whenever required.
  - **Subject Matter Expert (SME)** - It is not necessary for the project manager and team members to be experts in the core subject of the project. Hence, every project needs a subject matter expert (SME) who will support the project by providing vital inputs on the subject matter. The SME could be user, client, or a subject expert who has specifically been roped in for the project.
  - **Project team member** - The project team is formed according to the requirement of the project and could comprise of analyst, programmers, network specialists, graphic designers, etc.

6.3.2 Product

- This is the first step in project conceptualization and should be undertaken even before the project is planned. The product is the output of the project and hence its objectives and scope should be defined, alternative solutions should be considered and project constraints in terms of technical and management issues should be identified.
- Once this has been done it is possible to estimate cost of the project with reasonable accuracy, effectively assess risks to the project, break the project into tasks and manage the project schedule.
- The objectives of the project identify the overall goals for the product without considering how they will be attained. The product scope identifies the primary data, functions and behaviors that characterize the product. Once the product objectives and scope have been understood and alternative solutions have been properly evaluated then a solution can be arrived at.

6.3.3 Process

- Project Process is the application of knowledge, skill, tools and techniques to project activities to meet project requirements. Project management is accomplished through processes, which support all the activities necessary to implement the project, using project management skills, tools and techniques that receive inputs and deliver outputs.

- In order for the project to be successful the project manager should select appropriate processes within the process groups that are required to meet the project objectives.
- The project manager has to adapt the product specifications and plans to meet the requirements of the project and product.
- A process is a set of interrelated actions and activities that are performed to achieve a pre-specified set of products. The project management processes are undertaken by the project team and fall under one of the following category.
  - (i) The project management processes are common to most of the projects and are associated with each other by their performance for a common purpose. The common purpose is to initiate, plan, execute, monitor, control, and close a project.
  - (ii) Product oriented processes specify and create the project's products. Product oriented processes are defined by the project lifecycle and vary by application areas. Project management processes and product management processes interact throughout the project.
- Project management processes focus on defining, coordinating and controlling the activities needed to manage the project while product management processes focus on the final product i.e. the tangible output of the project, namely, the information system.
- Unlike, project management processes product management processes necessitate specific domain knowledge to complete the work.
- Thus, every project will require a subject matter expert who has the requisite knowledge, tools and techniques to handle the product processes.
- However, there must be a balance between project management processes and product management processes. For successful project management the project management and product management need to be properly aligned and connected with each other's processes to facilitate coordination.
- Balance is the key, a solo focus on project management processes will not enable the project team to define the project scope or develop a quality system and on the other hand a solo focus on product management processes will not ensure the completion of work as required.

6.3.3(A) Project Management Process Groups

GQ. Briefly describe the project management process groups.

- Project management process groups have been termed according to the nature of integration between the processes, the interactions within them and the purposes they serve.
- These processes have been grouped into five groups, defined as the project management process group. These process groups overlap between the different phases of the project lifecycle since the output of one process group becomes the input for the next or another process group.
- These five process groups are dependent and are performed in the same sequence on each project independent of the industry.
- A process group constitutes project management processes that are linked by their respective inputs and outputs i.e. the output of one process group becomes the input for the other.
- However, the process groups should not be mixed up with project phases. Large or complex projects may be divided into distinct phases or sub-projects and therefore all process group processes should be repeated for each phase or sub-project. The five process groups are as follows :

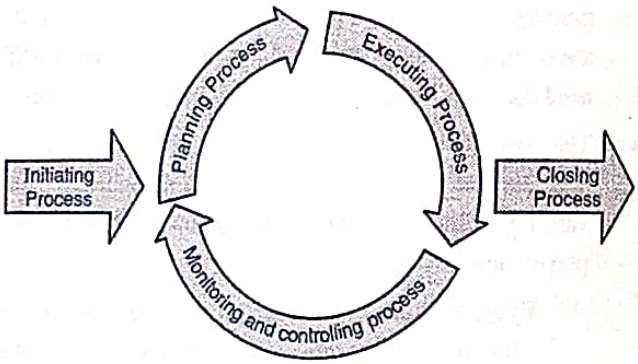


Fig. 6.3.1 : Project Management Processes

(i) Initiating Process Group

- The initiating process group signals the formal authorization to start a new project or project phase. Initiating process group processes are done external to the project's scope of control by the organization. Even

before beginning the initiating process group activities the organizations business needs are identified and documented.

- The business case is developed to establish the feasibility of the new project through a process of evaluating alternatives to pick the best one that can support the organizations strategy and goals. Along with the documentation of the objectives of the organization, the reason why this project best meets the objectives is also documented.
- Developing the business case needs time and money commitment from the organization i.e. there is a cost attached to it. This means that the organization has to make a commitment to the project even before its actual implementation.
- Once the business case receives the green signal from the management it becomes a project and requires greater commitment from the organization.

(ii) Planning Process Group

- The planning process group's processes and activities are used by the project team to successfully plan and manage the project. The planning process is the most critical part of the project management process. The planning process group gathers information from various sources and develops the project management plan.
- The efforts expended on planning should be proportional to the size and complexity of the project i.e. larger and complex projects demand more planning efforts while smaller and simpler projects could do away with less efforts. Although, the efforts on planning may vary from project to project, planning is required at each phase of the project more so during the development of project charter and project plan.
- The planning processes also identify, define, and mature the project scope, project cost and project schedule. The planning process is iterative in nature and is subject to constant change and revision.
- Project planning changes as more information emerges, additional dependencies are discovered, new requirements evolve, and new risks, opportunities, assumptions are identified. Hence, significant changes occur throughout the project lifecycle making it mandatory to revisit the planning process and in some cases the initiating process again and again. However, experience and good judgement enable the project team to overcome most of the difficulties during the planning process.
- Supporting processes include a basic description of the project scope, the deliverables, project duration, resource planning, activity planning, cost estimating, schedule estimating, organizational planning and procurement planning.

(iii) Executing Process Group

- The developed and approved project plan needs to be executed in order to bring the project to realty. The executing process group consists of the processes used to complete the work defined in the project plan.
- The project team should determine which of the processes are required for the implementation of the plan. The product oriented processes play an important role in the completion of the project plan activities.
- The executing process group involves coordinating people and resources as well as integrating and performing activities in accordance with the project plan. This process group also addresses the scope defined in the project scope statement and implements changes approved by the project team.
- The supporting processes in this process group include perform quality assurance, acquire and develop project team, risk management, and implementation plan.

(iv) Monitoring and Controlling Process Groups

- The monitoring and controlling process group consists of those processes that observe and control project execution so that potential problems can be identified in a timely manner and corrective action can be initiated. This process group also measures the progress made in the direction of the project's MOV as well as the scope, cost, schedule and quality objectives.
- The key benefit of this process group is that project performance is observed and measured regularly to identify variances from the project plan. This group also includes proposing controlling changes and preventive action in anticipation of problems.
- Supporting processes include scope control, schedule control, quality control, budget control and communications plan.

(v) Closing Process Group

- The closing process group includes all the processes used to formally terminate all the activities of a project or a project phase. In case the project has been successfully completed the project team should ensure that all deliverables have been completed to the satisfaction stakeholders and that the project sponsor accepts the project's products.
- The project team should also ensure that the final product has been successfully integrated into the day-to-day operations of the organizations. In case the project has failed the project team has to document all the relevant information with the project sponsor and then close the cancelled project. This process group when completed indicates that all the processes within all other process groups have been completed and the project or project phase is ready to be closed.
- There are primarily two types of closures; contract closure and administrative closure. Contract closure indicates that all agreed upon project deliverables and agreed upon terms have been completed and that the project can now officially end.
- It also paves the way for all the specifically assigned resources to be reassigned to new projects and dues under the contract to be paid. On the other hand, administrative closure involves documenting of all project related information that would prove valuable for future projects. Although each phase has a closing process, the major emphasis is during the final project closure process.

6.3.4 Project

GQ. State the five most important goals of a project.

- The project manager as a key person is expected to carry out various tasks and lead the project team. The role of a project manager involves handling all aspects of the project such as planning, brainstorming and overseeing the overall completion of the project.
- Therefore, for a project manager the trick for successful project management is to focus on the five most important goals associated with project management. Meeting these five goals of a project will ensure project success. Project goals keep the focus on what is important for the success of the project.
- The five most important goals of every project are :

(i) Ensure that the project is completed within the deadlines set

- Every project has a deadline within which it has to be completed and it is the duty and responsibility of the project manager to ensure that the deadline is adhered to.
- Although, the project manager and his team may have a role to play in deciding the schedule and at the outset of the project it may have looked feasible but as time passes and requirements change the project may require new strategies and planning making it difficult to complete the project within the stipulated time frame. However, the project manager should never lose sight of the deadline and keep pushing the team to complete the project within the time frame originally set.
- Hence, the project manager before committing to the deadline should take into account all eventualities which he and his team may have to encounter. It is always better to have a buffer period in hand.

(ii) Ensure that the project is completed within budget –

- Just as every project has a schedule it also has a budget. A budget is a forecast of the expenditure that will be incurred on the project.
- It is essential that the budget is prepared with considerable thought to all the activities and resources that would be utilized in the project. Hence, a budget is prepared after considerable research and after comparison of prices to get the best possible deals.
- Like the schedule a budget should leave no stone unturned. However, as requirements change there are chances that the original budget may go astray. It is therefore the responsibility of the project manager to ensure that things stay within budget.

(iii) Ensure user satisfaction

- Although user satisfaction should be the utmost priority for any project manager and is achieved by delivering the project on time it should in no way be achieved by compromising on the quality of the results.
- Quality of the project output is another project goal of which the project manager should never lose sight of.

v) Ensure that all user requirements are duly met

Another very critical goal for the project manager is to ensure that all the user requirements from the project are duly met. This would mean getting very detailed inputs from the user on his requirements from the project.

) Ensure team management

Along with ensuring that all project goals are met the project manager has to also ensure that his team is happy and contented.

It is the responsibility of the project manager to provide encouragement, incentives and rewards for the hard work put in by the team in the project. Remember, no project is complete without a team and hence it is the responsibility of the manager to ensure the well being of all the team members.

Achievement of these goals is no mean task as it is akin to juggling with five pins at the same time. The project manager should ensure that these goals are outlined to the project management team at the very beginning, there in no way for the delivery of the goals to be delayed in any way as everyone will always be aware of what they need to achieve and by when.

Syllabus Topic : W5HH Principle

6.4 W5HH

Q. State W5HH principle.

Barry Boehm suggested an approach that addresses project objectives, milestones and schedules, responsibilities, management and technical approaches and required resources. This is called W5HH principle. The questions that are answered in this principle are :

- Why is the system being developed?
- What will be done by When?
- Who is responsible for a function?
- Where they are organizationally located?
- How will the job be done technically and managerially?
- How much of each resource is needed?
- Why is the system being developed?
  - It enables the parties to assess the validity of business reasons for the software work. It justifies the expenditure of people, time, and money.
  - What will be done?
  - It specifies the task set required for the project.
  - When will it be done?
  - It helps to determine the project schedule. It helps in determining when tasks are conducted and when milestones are reached.
  - Who is responsible for a function?
  - It helps to accomplish the role and responsibilities of each member of the software team.
  - Where they are organizationally located?
  - The software team does not contain all the roles and responsibilities. The customers, users and stakeholders also have responsibilities.
  - How will the job be done technically and managerially?
  - The management and technical strategy of project is defined once the scope of the product is established.
  - How much of each resource is needed?
  - It helps in deriving estimates based on the answers to the above questions.

Syllabus Topic : Need for Project Management ?

6.5 NEED FOR PROJECT MANAGEMENT

- You may feel that why is such a fuss being made of the management of projects and the need for the adoption of proper project management tools and techniques.
- With increased competition and globalization a failed project could have more wide ranging implications that could affect the overall prospects of the organization.
- Project managers are expected to deliver results on time, within budget, while adhering to quality parameters and ensuring that the project results cover the entire scope of the project.
- Hence, a disciplined project management process is important to any project.
- Following are a few reasons why an increased need is felt for effective and efficient project management.

(i) **To control scope of project and manage change** – Although the project deliverables are defined at the outset of the project, small changes in project deliverables are common. These changes are demanded by customers, stakeholders, management, suppliers or the project team itself. Individually, these change demands may seem acceptable and manageable, but collectively these change demands can lead to a significant expansion in the project scope and can lead to an overrun in schedule and budget. However, with project management, if the manager effectively manages the scope of their project, they have a better chance of effectively managing project resources and change.

(ii) **To deliver projects on time and within budget** – The project management process includes cost calculations such as return on investments (ROI). Once ROI is established it is for the project manager to ensure that the project schedule and budget are adhered to else the project will fail to deliver the expected results.

(iii) **To ensure the focus of the project team** – It is common for the project team to drift from the main tasks and spend unnecessary longer time on other tasks. Hence, it is the responsibility of the project manager to ensure that the project team focuses on the right tasks by using a clear and concise project charter and that there are no interferences.

(iv) **To collect user requirement from disparate sources** – The project manager at the initiation phase should collect user requirement, project constraints and conduct a feasibility study to build a strong business case justification. The primary advantage of collecting input from various sources is that the project manager is able to avoid future dissent from users and is able to communicate project benefits.

(v) **To define the critical path to optimally complete the project** – Every project is made up of connected activities each having their individual constraints. By using the critical path method technique the project manager is able to identify the critical path and thus ensure the successful completion of the project.

(vi) **To provide a process for estimating project resources, time and costs** – Solid project management tools and techniques and past experience will enable the project manager to correctly estimate the project resources requirement as well as determine the time required for the completion of the project and the likely expenditure. Such, estimation at the initiation phase will ensure that the project receives adequate commitment from the management and its success is ensured.

(vii) **To communicate project progress, risks and changes** – The stakeholders of the project need to be kept updated on the project progress, hurdles encountered and changes incorporated. Every good project management plan has a communication plan that addresses communication issues, provides formats and lays process for execution of the communication plan.

(viii) **To explore project assumptions** – Although ample ground work is done for a project there are bound to be a few assumptions on which the project is based. Hence, a good project manager has to delve deeper into user requirements, project constraints and management expectations to understand the hidden project requirements. A project based on too many assumptions could ruin its chances of success.

(ix) **To prepare for unexpected project issues** – Howsoever, one may be prepared there are bound to be a few issues which may suddenly surface. Hence, the project manager should always be prepared with an alternate plan.

(x) **To document the knowledge gained from the project** – The last phase of the project involves the documentation of all that has been learnt at each phase in the project. These documented experiences provide guidance to other project manager in other projects.

Syllabus Topic : Project Lifecycle

6.6 PROJECT LIFECYCLE

GQ. Briefly describe the various phases of a generic project lifecycle.

- Projects do not form the core activity of the organization they are temporary activities with a definitive objective. Today, the project lifecycle

pproach is used in construction, aerospace, overnment agencies, research and development, anufacturing, electronics and many other industries; owever the term project lifecycle may mean different ings to different people depending upon the particular pplication.

he basic project lifecycle principles are applicable to a ide variety of projects such as new applications, ajor maintenance and repair projects, relocations and -organizations, creating new web site, or establishing new data base centre. Irrespective of the type of roject the basic framework shared by every project emains the same.

he project lifecycle describes the various logical hases in the life of a project from its beginning to its nd in order to deliver the final product of the project.

he idea of breaking the project into phases is to ensure at the project becomes manageable, activities are ranged in a logical sequence, and risk is reduced.

he phases are so created that each phase at its end rovides one or more deliverables. Deliverables are ngible and verifiable products of work that are roduced at the end of each phase and define the sources needed for the completion of each phase.

he review of the deliverables at the end of each phase ables the project manager to evaluate the erformance at the end of each phase and take ecessary corrective action if felt necessary.

lthough, it is recommended to undertake the phases in linear sequence, phases could be overlapped to save me. However, overlapping of resources is risky and ould be justified.

this section we shall be studying the generic project fecycle that comprises of phases common to most rojects.

he **Project Goal :** The first step for any project, respective of its size and complexity, is defining its verall goal. Every project undertaken aims to provide siness value to the organization hence the goal of the oject should focus on doing the same. Now, the goal the project needs to be defined initially and clearer it provides the project team with a clear focus and ides it through each phase of the project. The project hazy and seems risky at the start, but as the project goal gets defined and starts making progress, things start to look brighter and the probability of success increase. At the beginning of the project, the requirement of resources in terms of staff and cost is low; however, it increases as the project makes progress and reduces at the completion stage of the project.

- **The Project Plan :** The project's goal provides the guidelines and makes it easier for the development of the project plan also known as baseline plan. The project plan is developed to provide answers to various project related queries such as;
  - What the project aims to achieve? – The project deliverables
  - How does the project team aim to achieve it? – The tasks and activities
  - Who all will be involved in the project? – The stakeholders and the project team
  - How much will it cost? – The project budget
  - How much time will it take? – The project schedule
  - What are the risks involved? – Risk identification
- The initial plan serves as a base to gauge the progress made by the project throughout its lifecycle.
- **Project Plan Execution :** The project plan thus developed needs to now be executed. The project's progress should be compared with the baseline plan and communicated to all of the project's stakeholders. The project manager has to constantly monitor the projects' progress, schedule, budget, and resources.
- **Project Closure :** Project closure marks the culmination of the project; the project may have completed all the work as planned or the project sponsor may have called of the project due to various reasons which we shall study later on in this book. However, in the event of the project having completed all the work as planned, the project sponsor should formally acknowledge its successful completion, discharge all resources from the ongoing project, and make necessary payment of dues of any external consultants and suppliers. The project closure is also marked by the presentation of a project report that documents all the project deliverables.

- **Project Evaluation**
  - It is not possible to evaluate the true value of any IT project immediately after its implementation. Sometimes the goals are long term and the results of the project can only be gauged after the passage of a certain period of time. Merely evaluating the success achieved in implementing the hardware and software peripherals of the project does not amount to having succeeded in the project.
  - These are merely means to an end and hence time only will tell whether the project has succeeded in accomplishing what it was set in motion for. However, this does not mean that evaluation should not be undertaken immediately.
  - For starters, the project team should document all its experiences - things it did right, things it should have done differently, and all the lessons learned during the entire course of the project. This document proves useful in the execution of future projects for the organization. Also, the project manager and his team should be evaluated for their performance.
  - While, the project manager should evaluate each team member and provide feedback to the member as well as the management, the performance of the project manager should be evaluated by a senior manager.
  - The feedback proves useful to the team members in honing their skills, understanding where their erred in their role and the mistakes they made. The feedback proves useful in future assignment, promotions and incentives.
  - Evaluation of the project manager is carried out in terms of; the project delivering the promised work, abiding by budgetary provisions, following established processes, and ability to manage men and resources.

Syllabus Topic : ITPM

6.7 IT PROJECT METHODOLOGY

- As seen earlier in the chapter there are different types of IT projects such as CRM, SCM, ERP, Electronic Commerce, etc, that are undertaken by an organization. Each project has different products to deliver and hence will require different tools, approaches, processes, and knowledge areas. However, the idea behind developing a standard IT project management methodology was to provide the project manager with guidelines while undertaking these projects.
- The objective of the Information Technology Project Management Methodology is to develop a standard method and devise guidelines to ensure that IT projects are conducted in a consistent and well managed environment that assures the delivery of quality and ensures that the projects are completed on time and within budget.
- A project management methodology provides a generic strategic level plan for managing and controlling IT projects. The project management methodology is akin to developing a template for the process groups (initiating, planning, executing, controlling and closing) of an IT project.
- Now, IT projects are different i.e. the products that are attained on the completion of the projects are different, however it is the products which are different and not the process of managing the projects.
- Hence, it is possible to devise a methodology that will provide guidelines to the phases, processes, tools, deliverables and knowledge areas that will support any
- IT project. The objective of the methodology is to merely provide guidelines as each IT project may require different tools and approaches.

6.7.1 Project Management Methodology

- After spending considerable amount of time and money on training for developing general project management skills, organizations have discovered that project managers are still not able to tailor their project management skills to the organizations particular needs.
- Therefore, to overcome this problem, organizations have developed their own information technology project management methodology.
- The information technology project management methodology is a standard methodology describing the

best organization practices for managing projects and the manner in which project activities should be undertaken.

- IT project management methodologies provide the project manager with guidelines for implementing project lifecycles. Standard project management methodologies provide a common language for all those involved with the project, such as project manager, team members, stakeholders, sponsor, etc, to communicate more effectively.
- The project management methodology also enables comparison between various projects undertaken by the organization as it provides common parameters and language across all the projects.
- The project language, planning and reporting are the same across all the projects thus enabling the management to develop a better understanding which would also lead to better decision making with respect to allocation of resources and funds.
- The project management methodology should be flexible enough to adapt to the changing needs of the project over time. A flexible methodology would enable the project manager to use different modelling approaches during different phases of the development lifecycles.

6.7.2 Methodology Development

- The project management methodology and all associated activities are independent of the specific project development lifecycle selected. The project management methodology model is not intended as a specific project development lifecycle model (waterfall, spiral, etc).
- The model is a generic methodology for project management that accommodates various development approaches and a variety of detailed execution procedures that are defined by individual organizations.
- The foundation for the project management methodology comprises of:
  - **Project Management Processes** - Initiating, Planning, Executing, Controlling and Closure.
  - **Project Management Objectives** - Schedule, Budget, Scope, and Quality.
  - **Project Management Areas** - Time management, Communication Management, HR Management, Scope Management, Risk Management, Integration Management, Quality Management and Procurement Management.
- The development approach depends upon factors such as organizational experience in managing IT projects, the knowledge and experience of the project manager and the project team, and the very nature of the project in terms of size, complexity and the time frame within which it is to be delivered.
- The development approach is also influenced by the infrastructure support to the project provided by the organization and the management's commitment to the project. The development methodology should be a part of the overall project management methodology.
- The project management methodology has a general applicability to the management of all information technology projects. However, the degree to which the organization should apply the methodology depends on project and the risks associated with the project.
- The methodology identifies responsibilities and activities that are assigned and performed on information technology projects. The flexibility of the methodology permits rigorous application of management processes for large and complex projects than for small, well defined projects with readily achievable goals.
- Developing a methodology is not an overnight job it means inculcating all the experience gained from past and current projects in the development methodology to increase the chances of future project success.
- The development methodology over time incorporates all the best project management practices that suit the organization and the kind of projects that it undertakes. Once, the methodology has fully matured the likelihood of project success increases and there is a marked decrease in the wastage of resources.
- Organizations that have adapted and developed the approach observe a higher success rate in project implementation which in turn increases their competitive advantage, efficiency and effectiveness.

6.7.3 Phases in the Project Management Methodology

GQ. Describe the phases in IT Project Management Methodology.

The IT project management methodology comprises of five phases which are as follows :

- **Phase I : Project Conceptualization and Initialization** - This is the first stage in the project methodology and is responsible for preparing the ground work for further development. In this stage the idea behind the project is hatched and the goal or purpose of the project is defined. The primary project goal is very important in project methodology as it provides a basis for future project decisions and aids in defining the project scope.
- The project goal also serves as a parameter for evaluating the project's success after its completion. If the primary goal of the project is achieved the project could be termed as a success, however, there are other factors like the time and cost which also are also to be considered while determining project success.
- Project Initiation is the first official step in the project methodology and marks the commencement of the project. Project Initiation is based on the business needs that justify the allotment of resources and the expenses that will be incurred on it.
- The primary idea behind Project Initiation is to ensure that the business need is properly understood by the project manager and is kept in mind throughout the lifecycle of the project. Remember, the business need drives the project hence deviation from it is not acceptable. The business need is the direct outcome of the objectives and goals of the organization.
- The Project Initiation stage enables the creation of the project charter that is the official document authorizing the project manager to undertake the project within the organization. As everyone who is party to the project or who is likely to be affected by it is part of the project initiation stage it is easy for the manager to identify the stakeholders.
- The business need which drives the project could vary from organization to organization. While some organizations may feel the need to increase productivity others may feel the need to reach out to more customers or to increase efficiency.
- The project manager needs to understand the business need that is driving the project and also how the project will help the organization in meeting this business need. As stated earlier the organizational objectives, goals and mission are behind the business need hence the project manager should understand the objectives and goals of the organization to understand its business needs better. The project manager also needs to understand how the outcome of the project will be used by its stakeholders.
- **Phase II : Developing Project Plan and Charter** - A project is defined as an undertaking of non-routine nature to create a new or unique product, service or result. As the undertaking is new or unique and never been done before in the organization, planning is essential. The planning process should complement the size and complexity of the project i.e. the larger and more complex the project the greater the planning effort while small routine project require very less planning effort.
- Although, planning is an intrinsic part of each phase of the project methodology it is all the more relevant in this phase where the project plan and charter are developed. Another very critical feature of planning is that it is never over, planning is an iterative process, and may require constant changes in its budget, scope, schedule or quality, as per the requirement of the client, stakeholder or the management. The project manager will have to depend on his wisdom and experience in developing a good and pragmatic project plan.
- The project manager needs to focus on the project specifics and narrow the project description. It is the time to draft the Project Charter. A project charter is a detailed official document prepared in line with the company's vision and goal describing in detail the finer nuances of the project and chalking out deadlines for the milestones within the project. The Project Charter serves as a road map for the project manager and states the goals that are to be achieved from the project.
- A Project Charter gives a clear definition of the project, its attributes, the end results and the project authorities. Project authorities are the people who are responsible for the implementation and success of the project.

These people are namely the project in charge/project sponsor, project manager and the project team leaders.

- A project charter is the final official authorization for the commencement of the project to the project manager. It is a green signal to the project manager to commence work on the project.
- **Phase III : Project Execution and Control** - After having developed the project plan it is time to execute the plan. There are two types of processes, one which result in products and other which form the supporting processes.
- The product oriented process play an important role during the execution phase of the project. Quality assurance, risk management, team development, etc form the core of the supporting processes in the project execution. Although, the execution process is a part of every project phase it is more active and prominent during the execution and control phase of the project methodology.
- The second part of this phase is the controlling process. The primary objective of the controlling process is to measure and manage the project activities and ensure that they are on the right track towards the goal and adhere to the scope, budget, schedule, and quality parameters.
- The control process identifies deviation from the plan and makes it possible to take corrective action. Although, controlling process is present in each of the phases it has more emphasis in this phase.
- **Phase IV : Project Closure** - The primary objective of the closing process group is to ensure that the project reaches its logical conclusion and to bring the project to an orderly completion. The project closure phase is reached when all the project deliverables have been achieved and accepted by the project sponsor. At this phase the project team has to also ensure that the project integrates with the day-to-day operations of the organization and delivers information products as required.
- The closure of a project is marked by contract and administrative closure. Contract closure indicates that all the project deliverables have been successfully completed and all the agreed upon terms and conditions of the contract have been adhered too. Contract closure paves the way for the settlement of dues of outside parties, namely, suppliers and consultants.
- Administrative closure involves the documentation of all project activities and experiences for future references. Although, each project process group has closure of its activities, the major project closing process occurs in this phase.
- **Phase V : Project Evaluation** - The project evaluation phase focuses on evaluating the previous four project process group. The project review conducted by the project manager and his team should focus on assessing the positive and negative outcome of the project, things that worked in favour of the project and what went against the project. The experiences gained from the project should be well documented for future references.
- The project manager should also identify the best practices from the project that could be incorporated in the project management methodology of the organization. Inculcation of such best practices leads to the evolution of the methodology.
- The second part of the evaluation should take place between the project manager and the project team members and should focus on reviewing individual performance of team members as well as that of the project manager. The review should provide an honest and dispassionate feedback to the individual.
- The third part of the review should be conducted by a neutral party who should review the project, the project manager and the project team. The review should focus on the performance of the team in delivering what was promised to the sponsor, the ability of the team in meeting the objectives of the project scope, budget, schedule, and quality, overall client satisfaction, risks and challenges faced by the project team, the performance of the team as a unit, and the professional and ethical manner in which project activities were handled.
- The last part of the evaluation process should determine whether the project succeeded in providing value to the organization. Although, it may take time to gauge the full implications of the project, this part of the evaluation is essential to determine whether the project was an overall success.

Syllabus Topic : Project Feasibility

6.8 PROJECT FEASIBILITY

GQ. Why is it essential to undertake a feasibility study before project initiation?

- Information Technology projects are expensive and hence before project initiation a preliminary study called a feasibility study needs to be conducted to determine the validity of the project.
- The primary objective of the feasibility study is to ensure that the organization is on the right track and is focusing on the right problem with respect to the project.
- A feasibility study is undertaken to determine the information needs of the prospective users and the resource requirements, costs, benefits and the technical viability of the project.
- A feasibility study is a report of the research that the project manager has undertaken. It helps in determining the validity or scope of the entire project or a part of the project. As the title indicates the feasibility study tells the management if a problem is solvable or a business opportunity is realizable. Some feasibility studies may also cover the financial implications of undertaking the project.
- The financial part of the feasibility study will tell the management whether it makes financial sense in undertaking the project. It will indicate the Return on Investment (ROI).
- The project manager while preparing the feasibility report should refrain from expressing his opinion about the feasibility of the project.

  The report should be highly factual and down to the point. It should cover all the aspects of the project, its ability to address the problem or realize the business opportunity, financial implications, and the value that it will add to the organization.
- The project manager should be fair in his assessment and should not be tempted to impose new technology merely for the sake of technology. Many a times in IT projects it has been observed that project managers utilize technology merely to display their technical expertise.
- Technology for the sake of technology is no good; it should add value to the organization. The project manager, in his report, should express how the new technology will benefit the organization.
- The feasibility study comprises of the following :
  - **Executive Summary** - The purpose of the executive summary is to provide the reader with a brief summary of the findings of the feasibility study.
  - **Define the Business Problem or Opportunity** - In this section the business problem that the organization is encountering and its impact on the functioning of the organization is discussed. In case there is an opportunity that the organization wants to utilize that too will have to be defined and how it will benefit the organization has to be stated. The business goals or objectives can be utilized to link the project to the opportunity or problem. Also, the benefits of the proposed technology are to be stated in the report. The report should also identify the areas or people who are likely to be affected by the introduction of the new technology.
  - **Purpose of the Feasibility Study** - Although the primary purpose of any feasibility study is to determine the practicality of an opportunity or the solubility of a problem there are number of other reasons for which a feasibility study can be undertaken.
- To determine whether a particular product should be bought or made in house. It's a typical make-or-buy problem.
  - Compare various software and hardware solutions.
  - Determine capability resource gap with the new technology
  - The purpose of this section is to describe the intent of the feasibility study.
- **Assessment of Alternatives** - A feasibility study involves various alternative solutions for the business problem or opportunity. The project manager has to state in the report the alternatives evaluated, basis for selection of the alternatives, and the manner in which the alternatives differ from one another.

- **Impacted Areas** - This section of the feasibility study identifies the impacted people, addresses the issues concerning the users, and determines the capability resource gap. Some of issues addressed are;
  - Probably downtime that the users will experience on account of the implementation of the project.
  - Phased introduction of the new technology within the organization.
  - Assessment of requirement for training, the number of users requiring training, the training period, and the resources that will needed to impart training.
  - Learning curve of the new software.
  - How will the new software integrate with the organizations existing software
  - Does the new software require specific hardware?
  - Compatibility of new software with the existing operating system.
  - Period before an upgrade to the new software will be available.
  - This section will also cover the process of implementation of the new technology.
- **Financial Issues** - This section of the feasibility report will address all the financial issues related to the project. The issues addressed here are;
  - Price of the new technology
  - License fee and renewal period
  - Cost of training
  - Additional financial burden arising out of requirement for trained personnel
  - Cost of labour to implement the technology
  - Technical support from vendor
  - Loss incurred for not adopting the new technology
  - Return on Investment analysis
- **Project Manager's Recommendation** - This is the final section in the feasibility report and will contain the project manager's personal recommendation for or against the implementation of the proposed technology. The project manager should cover the working of the technology, its implementation process, and the resources required. The section should also cover the other alternatives that could be used and a comparison of how they measure up to the proposed technology.

Syllabus Topic : RFP

6.9 RFP

GQ. What is RFP?

- Projects require resources, products and services that are not always available within the organization and for which the organization has to resort to external consultants, suppliers and agencies. To procure these products, resources and services the project manager has to correctly identify the needs of the project that cannot be met internally and then specify it in a document called the Request for Proposal or RFP.
- A Request for Proposal (RFP) is a document that is used to solicit bids, quotes or proposals from prospective suppliers. On the other hand, a proposal is a document prepared by suppliers when there are different approaches for meeting the requirement of the organization. In the RFP the supplier may propose the use of various software, hardware and networking solutions to meet the requirements of the organization.
- Developing an RFP is a time consuming process and begins by determining the project needs that cannot be fulfilled internally and for which the organization will need to avail the services of outside suppliers.
- The task of determining the needs that can be met internally and those that cannot be met need to be left with the project manager and the project team. Along with determining the needs for which external suppliers are required the project manager and his team should also determine when, where, and how much of these resources would be required.
- The idea behind an RFP is to prepare a level playing field for all the prospective suppliers by stating in clear terms the requirements of the organization and all other technical specifications.
- An unambiguous document enables all suppliers to send their proposals clearly and also enables the organization to compare and evaluate proposals of various suppliers.
- Although, price is an important factor in determining the choice of supplier there are other factors such as experience, technical capability, track record, financial

stability, after sales service, etc which should be taken into account while determining the choice of supplier.

- The organization should obtain proposals from a reasonable number of reputed suppliers. To invite proposals the organization should either directly contact reputed suppliers and send them the RFP or could release an advertisement in the newspaper, internet or the trade journal and get suppliers to send their proposals.
- The project manager should also invite prospective suppliers for a meeting and update them on the requirement and provide answer to their queries.
- The elements of the supplier's proposal will include the following :
  - These items are a general framework and more or less should be a part of any proposal regardless of the type of work being done.
  - Summary of project : Identify the scope of work.
  - Overall budget and costs : Identify what the overall project budget looks like. At this point the IT professional will have to negotiate with the organization and give a written estimate of total cost.
  - Breakdown of each task and associated costs : Identify the specifics of each task and justify associated costs. Some costs may be fixed while others may be more variable. Some of the costs may be speculative and hence need to be indicated. Making sure the client knows this ahead of time will make it easier should unexpected costs arise.
  - Payment Schedule : States the manner in which the payment for the work undertaken is expected.
  - Provisions : Identify any provisions that are specific to the project. Provisions are an extremely important piece of the proposal as they allow the IT professional to set his terms in a legally-binding manner. The provisions included within the project proposal should include any that are specific to the project as well as any that are general to the business. Those provisions will be restated within the contract along with a general summary of the proposal.
  - Signatures : Even though the proposal likely won't be a legally binding contract once it has been negotiated and agreed upon, it is still good to sign off and have the client sign off on it as well.
  - Once, the proposals have been received the buying organization can begin the process of comparing, analyzing, and evaluating each proposal. The buying organization should prepare beforehand the criteria for selection of the supplier. As stated earlier, although price is an essential factor, other factors which could affect the outcome of the project should also be taken into consideration before selection of the supplier.

Syllabus Topic : PMBOK Knowledge Areas

6.10 PMBOK KNOWLEDGE AREAS

GQ. State the areas covered under project feasibility.

GQ. Enumerate PMBOK Knowledge Areas.

- The Project Management Body of Knowledge Guide (PMBOK Guide) prepared by the Project Management Institute, USA, includes proven traditional as well as well as innovative practices in the area of project management.
- The purpose of the PMBOK Guide is to identify the "good practices" that are applicable to most projects, most of the time and that there is wide spread consensus about their value and usefulness. However, "good practices" does not mean that knowledge should be applied uniformly to all the projects; the project management team should be responsible for determining what is appropriate for their project.
- The PMBOK Guide is used by various Project Management Institutes as a foundation for their professional development program. However, this guide only addresses project management practices that are generally recognized as "good practices", there are other external standards on organization project management that are also recognized as "good practices" but are not addressed by the guide. Hence, to develop an overall view of project management external additional information should also be refereed.

- The PMBOK Guide proves useful to;
  - Senior Executives
  - Project Managers
  - Functional Managers
  - Project Team Members
  - Stakeholders
  - Trainers
  - Consultants
  - Researchers

6.10.1 PMBOK Knowledge Areas

- The PMBOK defines nine knowledge areas for understanding project management and are as follows:
  - **Project Integration Management Knowledge Area** - This area includes the processes and activities needed to identify, define, combine, unify and coordinate the various processes and activities within the project management process groups. Integration activities are crucial to project completion. Integration management focuses on coordinating the project plans development, execution, and control.
  - **Project Scope Management Knowledge Area :** This area includes all the processes that are needed to ensure that project scope includes all the work required to be completed by the team, and only the work required to complete the work successfully. Scope management assures that project work expected from the team is defined accurately and completely and that the work is undertaken and completed as planned. Also, it is essential for Scope management to include proper scope management procedures.
  - **Project Time Management Knowledge Area :** This knowledge area includes the processes required to develop, monitor, and manage the project schedule such that the timely completion of the project is accomplished. Time management identifies project phases and activities and then estimates and sequences them. The assignment of resources to ensure the timely completion of the project while ensuring that the project scope and objectives are met is also included in this area.
  - **Project Cost Management Knowledge Area :** This includes the processes involved in planning, estimating, budgeting and controlling costs so that the project can be completed within the approved budget.
  - **Project Quality Management Knowledge Area :** Includes all the activities of the organization that determine qualities policies, objectives, and responsibilities so that the project will satisfy or exceed the needs of the stakeholders for whom the project had been undertaken.
  - **Project Human Resource Management Knowledge Area** - This area focuses on managing the most important resource on a project. It includes the processes that organize and manage the project team.
  - **Project Communication Management Knowledge Area :** This area focuses on developing processes that are required to ensure timely, accurate, and appropriate generation, collection, distribution, storage, retrieval, and disposition of project information.
  - **Project Risk Management Knowledge Area :** this includes all the processes associated with risk management planning, identification, response and monitoring on a project.
  - **Project Procurement Management Knowledge Area :** All resources i.e. people, hardware or software, needed for the project may not be available within the organization and it may need to procure some from outside sources. The procurement management area includes the processes to purchase or acquire these products or services to perform the work.
- The concepts associated with information systems and software engineering when integrated with the nine knowledge areas outlined in PMBOK provide the foundation for IT project management. Since, the PMBOK guideline are generic to any project, the integration of information systems management and software engineering with PMBOK concepts helps distinguish IT projects from other projects such as construction and engineering.

Syllabus Topic : Business Case

6.11 BUSINESS CASE

GQ. What is a Business Case? State the steps in developing the business case.

- The desire to utilize information technology to improve the efficiency and effectiveness of the organization has lead to an increase in the number of IT projects undertaken within organizations. However, the utilization of information technology without a thorough understanding of its risk and cost implications has lead to the failure of many IT projects.
- The haphazard utilization of information technology merely because it is available is not judicious. The rampant failure of IT projects on account of the inability of projects to deliver returns as compared to the time and resources invested in them is an indicator of the need to develop some sort of analysis or feasibility study which would indicate the true value of the project for the organization.
- The business case arises out of this need and provides an analysis of the technical feasibility, costs, risks, returns, and organization value of various projects. It provides the basis on which informed decision on the projects can be taken.
- Although, the business case may sound very similar to a project plan or budget it should not be mixed up with one. The business case is a document that provides the top management with all the information needed to select the projects that are to be funded.
- Although creation of the business case document is very much similar to the feasibility report in most of the cases it is a separate document. Like the feasibility study the business case too helps the management in justifying the cost that will be incurred on the project and its return on investment.
- The business case is built on the relevance of the business goals and objectives and the cost of the proposed technology that can get the organization there.
- The business case takes into account the cost of the solution, break-even point, return on investment, and maintenance cost. Along with the quantitative issues the business case may also address qualitative issues such as; working comfort, increased efficiency, improved morale, etc.
- The business case is very similar to an investment proposal. Hence, the developer of the business case has to present compelling facts and figures in favour of the project and base his argument in the most logical manner. However, the business case developer in no way should digress from the facts of the project and the analysis should be as dispassionate as possible. A good business case should;
  - Detail all possible impacts, costs and benefits.
  - Be clear and logical in comparing the cost-benefit impact of each project alternative.
  - Include all pertinent information.
  - Systematically summarize all the findings.

6.11.1 Developing the Business Case

- Although, IT projects are undertaken for a plethora of reasons such as; improve customer service, reduce costs, improve communication, integrate customers, suppliers and partners, improve decision making, etc, the underlying objective behind each reason is to create organizational value by way of improving efficiency and effectiveness.
- The priority amongst the various reasons for undertaking IT projects differs from organization to organization. It is up to the top management to evaluate the various project proposals and select those which will create maximum organizational value. Therefore, the objective of the business case is to evaluate and advocate how the IT project will create value for the organization and improve its efficiency and effectiveness.

Steps in developing the business case

- **Step I :** Forming the team - developing the business case should not be a single man's responsibility. On the other hand, a team comprising of stakeholders, managers, users and IT specialists should be formed. The team so formed should bring in all the requisite knowledge, experience, information and expertise required to develop the business case. Of the team, the stakeholders are the people who are going to be affected

y the project and hence their point of view needs to be esented in the business case document. Business anagers bring in a higher level of perspective which is essential for the business case. The users are the ones ho understand the requirements that the project has to lfil. While the IT specialists understand the risks and nefits associated with IT and should present their rspective in the business case.

According to Schmidt, there are several advantages of having a team develop the business case :

- **Credibility** - As a team is made up of individuals from various organizational areas they possess expertise, knowledge and experience that may not be readily available outside the realms of their operations. A team also provides different perspectives to the project and provides inputs that an individual developer may tend to overlook.
- **Alignment with organizational goals** - The managers in the core team can align the business case with the long term strategic goals of the organization. Aligning the project with the strategic goals of the organization further strengthens the business case. The business case should also highlight how the successful completion of the project will help achieve the overall goals and objectives of the organization.
- **Access to real costs** - The team members selected should have specific expertise and should be privy to information that will assist in building a realistic estimate of the costs.

The other advantage of forming a team from various areas of the organization is that enables the project manager to manage resistance during project implementation.

tep II :Developing Measurable Organizational alue (MOV) - Measurable organizational value MOV) is the IT projects overall goal and its measure of ccess. For any project the MOV should align with the ganization's overall mission, objectives and goals. The rm Measurable Organizational Value was coined by ck Marchewka as an alternative to the more popular turn on investment (ROI). According to Jack larchewka the projects MOV should be:

- Measurable
- Provide value to the organization
- Agreed upon by all the core team members.
- Verifiable at the end of the project.
- Guide the project throughout its lifecycle.
- Align with the organizations strategy and goals.

A clear MOV will enable the team to know where the project should go, it will be like the road that the whole project lifecycle should take. In case the project deviates from its path, the relevant decisions and adjustments will be based on the MOV which can be vital in achieving the overall goal.

Steps in developing the MOV

(i) **Identifying the desired area of impact :** The desired area of impact expected from the IT project is the primary reason for undertaking the project. To identify the desired area of impact the project manager should find out from the project sponsor how the idea of the project came about. Although, the answers could be vague it would provide the project manager with some elementary background as to how decisions are made in the organization.

(ii) **Identifying the desired value of the project :** The value that a project brings to an organization could be in terms of doing something faster, better, cheaper, or on increasing the market share. The project manager should identify and highlight what the project could potentially deliver to the organization.

(iii) **Developing an appropriate metric :** Once the desired value of the project has been identified it is time to quantify the value that would be delivered. So for example a proposed business information system would enable a company to process customer orders faster, the management would be better off in learning the reduced customer order cycle time. Therefore, the value should be quantified and expressed in terms of a metrics i.e. in terms of time, money, percentage or a specific value.

(iv) **Setting a time frame for achieving the MOV :** The time frame for achieving the MOV should also be mentioned. Some MOV's are immediately achievable on the completion of the project, for example the reduction in customer order cycle time, however, some like increase in market share may take time.

(v) **Verifying with stakeholders :** Getting the metric value and time frame verified and approved from the stakeholders adds value to claims made in the business case.

(vi) **Summarize the MOV in a clear and concise statement :** The MOV should be mentioned in a clear and concise statement.

- **Step III : Identifying Alternatives :** All the alternative solutions to the problem or opportunity need to be delved upon in the business case. These alternatives should also enable the company to achieve the desired MOV. The alternative of maintaining status quo i.e. not doing anything and continuing with the present situation could also be a solution that needs to be looked into. However, the business case should put forth compelling reasons to bring about change and the cost that would be incurred in continuing with the existing process, systems or product. The cost incurred could be in terms of downtime, maintenance cost and systems failure.
- **Step IV : Defining Feasibility and Assess Risk :** The feasibility and risk associated with each alternative solution should be analysed. Feasibility is the probability of successfully implementing an alternative while risk focuses on what can go wrong and what must go right. Feasibility and risk analysis will enable the project manager to identify alternatives that are not worth pursuing. Feasibility is viewed in terms of economic feasibility, technical feasibility, and organizational feasibility. Risk analysis focuses on its identification, assessment, and response.
- **Step V : Defining Total Cost of Ownership :** The total cost of ownership of the application needs to be accounted for before any decision on implementing it is to be taken. Total cost of ownership is over and above the cost of purchasing or developing the application and as such includes cost of purchasing/developing, training, maintaining and supporting the application over its entire lifetime. Thus, total cost of ownership calculation is complex and hence the project manager has to authenticate his calculation with data sources, assumptions and methods for arriving at the cost.
- **Step VI : Defining Total Benefits of Ownership :** The total benefits of ownership includes direct and indirect benefits associated with each alternative. Benefits could be in terms of increased efficiency, improved productivity, improved customer service, improved accuracy and efficiency, or improved decision making. However, not all benefits all easy to identify and quantify. Every alternative has certain tangible and intangible benefits. Tangible benefits are easy to identify and quantify as they lead to cost savings. On the other hand, though intangible benefits are identifiable they are difficult to quantify. The project manager should try and quantify intangible benefits by linking them to tangible benefits.
- **Step VII : Analysing alternatives :** Once costs and benefits have been identified it is time to compare all the alternatives and arrive at one that best meets the requirement of the organization. Financial models such as ROI, Payback and Net Present Value and Scoring Models are used to analyse the alternatives.
- **Step VIII : Recommend Solution :** After analysing each alternative the one that best suits the company should be recommended for approval.

Syllabus Topic : Project Planning

6.12 PROJECT PLANNING

GQ. Write short note on: Project Planning.

- Although, project management processes are presented as discrete components with well defined interfaces in reality the overlap and interact in every way possible.
- The need for integration in project management is all the more evident in situations where processes interact. For example, cost estimate needed for contingency plan would need the integration of project cost management processes, project time management processes and project risk management processes.
- Thus, project integration management is one of the most important knowledge areas because it facilitates coordination amongst all the other eight knowledge areas throughout the project lifecycle.
- Project Integration Management includes the processes and activities needed to identify, define, combine, unify and coordinate the various processes and activities within the project management process groups.
- The project manager should ensure that all the project activities and processes are properly coordinated such that the project is geared to meet any problems and achieve success in implementation. Integration activities are crucial to successful project completion.

- Project integration management focuses on coordinating the project plan development, project plan execution, and monitoring and change control.

Project Plan Development

- Project plan development is the longest and most important activity of project management. The purpose of the project plan development is to create a flexible yet concrete document that will guide the activities of the project.
- Thus, the project plan document is like a blueprint for the project. Without proper planning, a project has a poor chance of success. Team members must decide on a budget, set timelines, and identify resources and any roadblocks to success.
- The team verifies the availability of resources, materials and expertise critical to on-time project completion. Project teams should spend the necessary time planning a project and make any plan changes before moving on to the next phase. The team must put project plans in writing to clarify roles, responsibilities and project deadlines.
- Howsoever, simple it may seem Project plan development is not the forte of many project managers, IT professionals, and freelancers. From the project managers point of view project planning is the first step in the implementation of the project.
- A project plan is an iterative process that communicates the approach and intent of the project manager. A project plan will provide the details of the processes that will be used in the project and how the project work will be executed, controlled, and commissioned.
- The first draft of the project plan is based on the business case and any other additional information that may be available at that time. The project manager could refer to documents related to past projects to understand how the approach adopted by these projects fared in terms of accuracy and completeness of their assumptions and estimates.
- The first step in Project Planning is to research the business opportunity or problem that the project aims to address. A good research will enable the project manager to develop a thorough understanding of the problem. The research can be done by interviewing the key stakeholders. The project manager needs to understand why the project is being initiated and what it aims to accomplish.
- The success of the project hinges on a clear understanding on the part of the project manager and key stakeholders as to the motive and key deliverables of the project.
- The vision for the end result of the project should be mutually developed by the key stakeholders and project manager.
- While developing the project plan the project manager has to take into account the policies and procedures of the organization. Each organization has its own policies and procedures which are kind of an unwritten rule.
- In case the project team and manager are from within the organization i.e. they are internal members then they are well aware of the policies and procedures that are expected to be followed. However, an outside project manager or team is also expected to learn these policies and procedures and follow them because they have a big impact on the outcome of the project.
- The project manager should approach the project plan development stage cautiously. He should be aware of the resources in hand, the people assisting him, and the time he should give to the planning phase. Although, planning is an iterative process that the project manager will be revisiting time and again during the entire course of the project implementation, he should decide in advance the time that would be spent on the planning phase.
- The usually practice in determining the time one should spend on planning is that it should be directly proportional to the size and relevance of the project.
- Also, every constraint and assumption taken into consideration while developing the project plan has to be documented. Constraints are factors which limit the functionality of the project and have an impact on the scope, schedule, budget, or quality of the project.
- On the other hand assumptions form the basis of the project plan and must go as per planned for the project to succeed. Assumptions and constraints are considered while developing the risk management plan which is a key constituent of the project plan.
- The project plan enables controlling the various activities of the project and ensuring that they are in

accordance with the plan. Also, the control mechanism facilitates coordinating changes across the entire project.

Project Plan Execution

- After having developed the project plan and set appropriate project baselines, it's time to execute the project plan. The project plan has to be realistic as during its execution most of the resources assigned to the project will be expended.
- Project execution or implementation is the phase in which the plan designed in the prior phases is put into action.
- The purpose of project execution is to produce the project deliverable and other direct outputs. Typically, this is the longest phase of the project lifecycle and where majority of the resources are expended. The project plan should not only guide the usage of the resources assigned to the project but also facilitate the monitoring the progress made towards the goal.
- The first step in execution is the formation of a team with the skills and knowledge required to deliver the project's products.
- The project manager should ensure that his team has the requisite skills and knowledge before being a part of the team or have made arrangements to acquire them in due time.
- The project execution phase requires the project manager and his team to perform multiple activities to execute the project plan and accomplish the work defined in the project scope statement. However, the project plan has a work authorization system that authorizes team members to undertake specific activities and ensures the sequence of the activities.
- The project execution plan ensures that project activities are carried out as per the plan. Without a project execution plan in place, each team member would undertake activities as per their own practice and experience. This may jeopardise the complete project.
- Project execution depends on the plans developed in the planning phase. In the project execution phase the project manager has to monitor the execution of other plans such as the communication plan, risk management plan, and procurement plan via daily interaction with the project team and stakeholders.

- **Information Distribution :** Ensuring that right information is available at the right place at the right time is very crucial for the success of the project.
- **Status Reviews :** The project communication plan is an important factor in the execution plan. The communication plan should define which people need what information, when they would need the information, and how they would get the information. Joint project reviews serve as a good tool to bring visibility to all areas of the project. Joint Project Reviews provide an opportunity to discuss pressing issues and make crucial decisions on the project. These reviews involve the project manager, team members, stakeholders, and project sponsor. The communication plan will also define the frequency of the meetings, the topics to be discussed, and the means of communication.
- **Project Administration :** Project administration is a part of the project planning phase and is implemented and modified as necessary during the project execution phase. Modification may be needed because of;

- Changes in scope arising out of changes in the functionality of the deliverables.
- New estimates of work to be completed arising out of revelation of new or additional information.
- Changes in resource arising out of change in the project scope.
- Project administration also involves monitoring the various activities of the execution phase, risk management, and authorizing project changes.

- **Procurement** – In case there is a need to purchase products outside the resource pool of the organization the procurement plan is set into motion. Every organization has devised its own set of guidelines and policies for purchasing which should be integrated in the purchase plan. The project manager should follow these guidelines and policies while undertaking procurement of new products or services that were not planned for in the planning phase. The project manager will be responsible for vendors meeting their contractual agreements.

- **Scope Verification** – Scope verification is the responsibility of the project manager. He will have to ensure that the products created during the execution process are as per the stated requirement in the baseline scope with agreed upon tolerance limits. Scope verification is achieved through formal reviews of the deliverables.
- **Documentation** – Information on the work status of the project deliverables needs to be collected and regularly updated for effective management. The information documented includes the current status of the work on the deliverables, quality standards, and the cost that is incurred.
- **Work Authorization** – The work authorization is the formal procedure for sanctioning project work. Work authorization is a written document stating the commencement and completion time and the sequence that is to be followed.

Overall Change Control

- The Change Control process ensures that changes as a result of project corrective actions and other controlling factors are managed across the project knowledge areas. Change control takes place throughout the project, from project initiation through project closure.
  - Approved change requests
  - Rejected change requests
  - Updates to the Project Management Plan
  - Updates to the Project Scope Statement
  - Approved corrective and preventive actions
  - Approved defect repair
  - Validated defect repair
  - Deliverables

Syllabus Topic : Project Charter and Project Scope

6.13 PROJECT CHARTER AND PROJECT SCOPE

GQ. Write short note on : Project Charter and Project Scope.

- The first step in the project initiation stage was the determination of the business need and identification of key stakeholders. However, till now all the project discussions were broadly based and were general descriptions.
- The project manager needs to focus on the project specifics and narrow the project description. It is the time to draft the Project Charter.
- A project charter is a detailed official document prepared in line with the company's vision and goal describing in detail the finer nuances of the project and chalking out deadlines for the milestones within the project.
- The Project Charter serves as a road map for the project manager and states the goals that are to be achieved from the project.
- A Project Charter gives a clear definition of the project, its attributes, the end results and the project authorities. Project authorities are the people who are responsible for the implementation and success of the project.
- These people are namely the project in charge/project sponsor, project manager and the project team leaders. A project charter is the final official authorization for the commencement of the project to the project manager. It is a green signal to the project manager to commence work on the project.
- The project charter and the project plan provide a tactical plan for the execution of the project. As stated earlier, the project charter is an official document authorizing the project manager and team to undertake the project.
- It is an agreement between the project sponsor and the project manager and his team and documents the project's MOV, defines the infrastructure, summarizes the project plan, defines roles and responsibilities, and states project control mechanism.
- The project charter and the project plan should be developed simultaneously as the summary of the plan has to be included in the project charter. Also, the infrastructure identified in the project charter will influence the project schedule and estimates in the project plan.
- As done in the business case the development of the project charter and project plan should include the project manager, the project team, and the project sponsor. This will ensure that they all agree and

subscribe to the assumptions and constraints in the project plan and that the goals and objectives of the project are achievable.

- The project charter covers the following :
  - **Project's MOV :** In the previous chapter we have seen the project's MOV in detail in the business case. The MOV now needs to be agreed upon by both the project sponsor and the project manager and his team. As seen, the project's MOV drives the project and once agreed upon should not be changed. The MOV drives every decision in the project, the project planning process, and the requirement of resources.
  - **Project Infrastructure :** The project charter states all the key resources, people, technology, methods, processes that go into the making of the project i.e. all that is needed to see the project through. This is the project infrastructure that is available with the project manager. However, the available infrastructure needs to be taken into account even while developing the project plan. The available infrastructure will enable the project manager determine the schedule and the budget for the project.
  - **Summary of the project plan :** The project charter should include a brief summary of the project scope, budget, schedule, quality objectives, key deliverables, and major milestones. Thus, the project charter should serve as a source of information to all those seeking details of the project. It is more like a ready reckoner that can be referred to by anyone seeking information on the project.
  - **Roles and Responsibilities :** Not only should the project charter identify all those involved in the project but all define their individual roles and responsibilities, and the line of command.
  - **Project Control Mechanism :** As the project progresses and new data and information emerges a need to make changes to the project scope, schedule, and budget will undoubtedly be required. However, incorporating these changes is not an easy task and may make the project team lose focus. Therefore, to facilitate change the project charter should outline a process for change management.

Purpose of the Project Charter

- A project charter serves the following purpose;
- Defines the business need
- Identifies the project sponsor
- Authorizes the project
- Identifies the project manager, grants authority and makes him responsible for the management of the project

Elements of the Project Charter

The depth of the project charter depends on the size and complexity of the project, however, the project charter should address the fundamental project management processes and areas included in the project. The project charter contains all the relevant information related to the project and includes;

I. **Official Project Title :** Every project has a name by which it is identified. The name is usually based on the kind of work the project is undertaking. Naming a project is especially necessary in organizations which have multiple projects underway at the same time. Also, the project name gives a sense of identity to the project team and stakeholders.

II. **Project Sponsor :** The name, designation and contact details of the person who has authorized the project. The project sponsor is the major stakeholder in the project and takes ownership of the project after its completion. The project sponsor is the person authorized by the management to take all decisions in relation to the project, he has the authority to sign the project charter and grant resources. Every change in the project has to be authorized by the project sponsor.

III. **Project Manager :** The name, designation and contact details of the person who is responsible for the implementation of the project. The project manager is another major stakeholder in the project. However, it is not always that the project manager is a part of the same organization he may also be a free lance project consultant who has been hired by the company specifically for the project.

IV. **Description of the Project :** Every project that is undertaken addresses some problems. Hence, the

charter should spell out the need or purpose behind the project. The purpose or description of the project enables a person totally unfamiliar with the project to understand the project and see the larger picture. The project description covers the problem or opportunity that has become a catalyst or purpose for undertaking the project. The description should also state how the project aligns with the organization's goal and strategy. The description of the project should be a business analysis and not a technical analysis.

V. **Measurable Organizational Value (MOV) :** Although the MOV has been discussed at length in the Business Case it should again be clarified and agreed upon by all the stakeholders in the project charter.

VI. **Project Scope :** The project scope statement is the most important document in the business charter. Project scope is the difference between the envisaged business goals and objectives and the current position of the organization. The project scope statement is based on the project requirement, feasibility study, business goals and objectives and the business case.

- The project scope statement defines the project boundaries, project deliverables and the work needed to be done by the project team to accomplish those deliverables.
- The project scope statement is the output of the joint effort of the project manager, project sponsor, and the key stakeholders. Since so many people are involved in preparing and approving the statement it undergoes considerable refinement before it gets the final nod from the involved parties. Howsoever, time consuming the process of creating the project scope statement maybe it is important to involve all the stakeholders and get their approval.
- The project scope statement serves as the primary document for the remainder of the stages in the planning process. The deliverables that the project will create can be used as a reference point for future project decisions.
- Another, very important component of the project scope statement is the definition of the project boundaries. The project boundaries not only define what the project will do but will also define what the project is not capable of doing. Hence, it is very important to have utmost clarity on the project boundaries.
- Another purpose of the project scope statement is that it defines key performance indicators against which the performance of the project is measured. The performance is usually measured at the key project phases otherwise known as milestones.
- The components of the project scope statement are :
  - **Project Scope Description** - Is the description of the deliverables of the project that the client will get upon successful completion of the project.
  - **Project Acceptance Criteria** - The project acceptance criteria will defines what the project should create to be accepted by the organization.
  - **Miscellaneous Deliverables** - Apart from the key deliverables the project scope statement should also define those deliverables that are part and parcel of the project.
  - **Project Exclusions** – Are those deliverables that are not included in the project scope. To avoid ambiguity the project boundaries should be clearly defined.
  - **Project Constraints** - The constraints that limit the actions of the project manager need to be defined.
  - **Project Assumptions** - Every project is based on certain assumptions which need to stand the test. Some of the assumptions may be; the continued support of management for the project, software and hardware compatibility, etc.

VII. **Road Map for Work :** The road map will contain of the approach that the project manager has adapted.

VIII. **Project Schedules :** Timeline for the completion of the major milestone stages in the project.

IX. **Project Budget :** The total cost of the project should be summarized in the project charter.

X. **Project Resources :** Specifies the budget for the various stages of the project and other key resources and players of the project. The charter also specifies the party responsible for providing these resources. There should be no ambiguity as to who is providing the project resources. Resources include people,

technology and infrastructure needed to support the project team.

XI. **Assumptions and Constraints :** Specifies the assumptions and constraints of the project. Assumptions are those factors which form the basis of the project and are expected to go as per the plan for the project to be successful. This could include the availability of people, technology or facilities. On the other hand, constraints are those factors which limit the scope of the project.

XII. **Risks :** every project has certain risk factor associated with it, hence it is always beneficial for the project manager to identify the risk factors associated with the project and be ready with solutions to tackle them.

XIII. **Project Administration :** Project administration focuses on those controls that support the project. Thus project administration will include, the project communication plan, project human resource plan, project change management plan, project quality management plan, and the project scope management plan.

XIV. **Acceptance :** Although the project charter is developed mutually by the project sponsor and the project manager and his team and serves as an agreement between the two of them it is necessary for the two of them to sign the project charter. Signing of the business charter indicates the acceptance of the terms of the charter.

Syllabus Topic : Work Breakdown Structure

6.14 WORK BREAKDOWN STRUCTURE

GQ. "WBS assists in accomplishing project objectives and create required project products". Explain.

GQ. Write short notes on :

1. Delivery Definition Table (DDT)
2. Project Scope Statement

Introduction

- The primary requirement for project planning is the Work Breakdown Structure (WBS) for which the approved project scope statement is required. The WBS is a deliverable/product oriented hierarchical decomposition of the work to be executed by the project team, to accomplish the project objectives and create the required deliverables/project products.
- Decomposition indicates the breaking down of the project scope statement into smaller, more manageable components i.e. the project deliverables.
- The project scope statement defines all the project deliverables. These defined deliverables are then clubbed together to form the Delivery Definition Table (DDT).
- The Delivery Definition Table contains the sequence of the delivery of the deliverables. The Delivery Definition Table is then used to create the Delivery Structure Table (DSC) which contains the work packages which is then further used to create the Work Breakdown Structure.
- Thus, WBS is similar to Bill of Materials (BOM), wherein a product is broken into its smallest component for which estimation of time and cost is possible. The idea behind WBS is to divide each component till its reaches its smallest component, the work package.
- The work package is the smallest component in the WBS that facilitates estimation of cost, schedule, resources, and monitoring and control.
- Each descending level of the WBS represents an increasingly detailed definition of the project work. Thus, WBS provides the necessary framework for detailed cost estimating, guidance for schedule development, and control.
- At the top the WBS offers high-level deliverables each of which are then broken down into more detailed and well defined deliverables known as the work packages. Further on these work packages are decomposed into activities in the project schedule.
- The sum of all the work packages comprising of activities should be equal to the project scope. The completion of the project scope will also result in the completion of the product scope. Completion of both scopes would mean the completion of the project.

14.1 Benefits of WBS

tate the benefits of WBS.

viding complex projects to simpler and manageable the main purpose of developing WBS. This method by project managers to simplify project execution as work packages and activities are easier to manage. ng are the benefits for developing WBS in a project :

BS serves as an input to key project management ivities, namely, cost budgeting, resource planning, k management, activity definition, and schedule nning.

BS illustrates the project scope, so that every team mber as well as stakeholder can have a better derstanding of it.

BS provides an accurate and readable project ganization.

BS facilitates accurate assignment of responsibilities the project team.

BS indicates the project milestones and control ints.

6.14.2 Developing the WBS

- Project managers, project team and Subject Matter Experts (SME) typically develop a WBS as a precursor to a detailed project schedule and budget estimate.
- Although, there are different methods of decomposing project work and creating WBS, the starting point for deriving a WBS would be identifying the main deliverables of the project.
- The next step would be to convene a meeting of all team members and subject matter experts, who would then brainstorm all the work required to complete project deliverables successfully.
- The involvement of all team members and subject matter experts increases the likelihood of the resulting WBS being comprehensive.
- Team members and subject matter experts identify all project deliverables and milestones and then decompose them one at a time into a detailed and sequential list of activities required to complete them

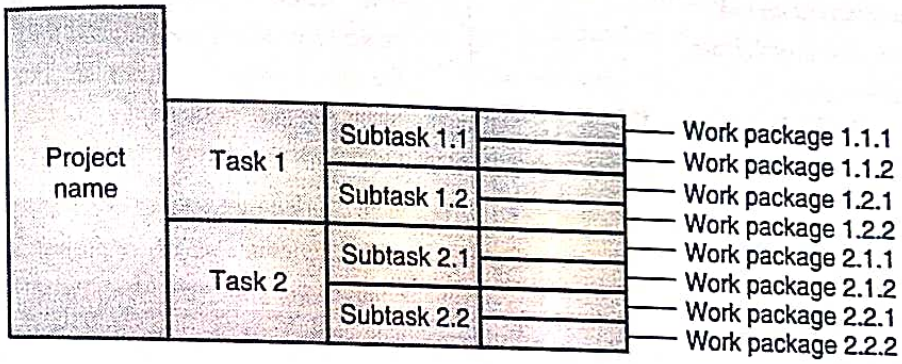


Fig. 6.14.1

is logical to create a WBS based on the nature of the oject work. However, this is possible only in projects here it is possible to identify the phases of project. In ojects where the focus of the project execution is not ear, the project manager would be better off in tegorizing the project components and then ecomposing.

here are no hard and fast rules to the level and manner f breakdown of the WBS. Some project managers reak down the WBS till it is not possible to break any rther. However, this method is not recommended as the tasks created would be too small to manage, measure, and assign. Therefore, for a small project the work could be broken down into a few days of work while for a large project the work could be broken down into weeks of work.

- The general rule is the "two weeks" rule; no task is broken to anything less than two weeks of work i.e. the smallest task in the WBS will require a period of two weeks to complete. Another rule in creating the WBS is the "8/80" rule. This rule suggests that the smallest work package in the project should take no more than

80 hours to complete or no less than 8 hours to complete. In short, no task should be smaller than 8 hours of work and should not be larger than 80 hours of work.

- The WBS should be accompanied by a WBS Dictionary, which lists and defines WBS elements. The WBS dictionary defines each work package in the WBS.
- The dictionary identifies each component of the WBS and how they relate to the project scope. The description of each element of the WBS is stated in simple language that everyone can understand. The WBS dictionary contains the following :
  (i) Code of accounts
  (ii) Description of the task
  (iii) Person responsible for the task
  (iv) Task dependencies
  (v) Schedule of the task
  (vi) Resources required for the task
  (vii) Cost to create the task
  (viii) Quality requirements of the task
  (ix) Technical details of the task
  (x) Acceptance criteria of the task
  (xi) Contract information

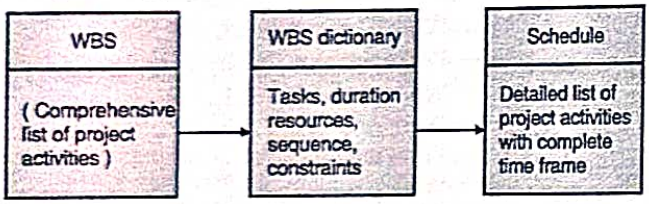


Fig. 6.14.2

- WBS and WBS dictionary are not the schedule but the building blocks for the schedule. The WBS and WBS dictionary are not static documents and are subject to change as more information related to the project emerges.
- A Project Team that has substantial changes to its WBS should reference the project's Change Management Plan for guidance on management of changes to project scope.

6.14.3 WBS Diagram and Numbering System

- The WBS numbering system or the "code of accounts" is a system that shows the different level of WBS components and identifies which component belongs to which part of the WBS. Every level has a unique number assigned so that the work can be identified and tracked over time.
- A WBS may have varying numbers of decomposition levels, but there is a general scheme for how to number each level so that tasks are uniquely numbered and correctly summarized. Below is the general convention for how tasks are decomposed :
  - **Level 1 :** Designated by 1.0. This level is the top level of the WBS and is usually the project name. All other levels are subordinate to this level.
  - **Level 2 :** Designated by 1.X (e.g., 1.1, 1.2). This level is the summary level.
  - **Level 3 :** Designated by 1.X.X (e.g., 1.1.1, 1.1.2). This third level comprises the subcomponents to each level 2 summary element.
- This effort continues down until progressively subordinate levels are assigned for all work required for the entire project. If tasks are properly subordinated, most project scheduling tools will automatically number tasks using the above convention.
- A WBS diagram starts with a graphic box at the top, which represents the entire project. Then, there are sub-components under the box. These boxes represent the deliverables of the project. Under each deliverable, there are sub-elements listed.
- These sub-elements are the activities that should be performed in order to achieve the deliverables. In addition to the general use of WBS, there is specific objective for deriving a WBS as well. WBS is the input for Gantt charts, a tool that is used for project management purpose. Gantt chart is used for tracking the progression of the tasks derived by WBS.

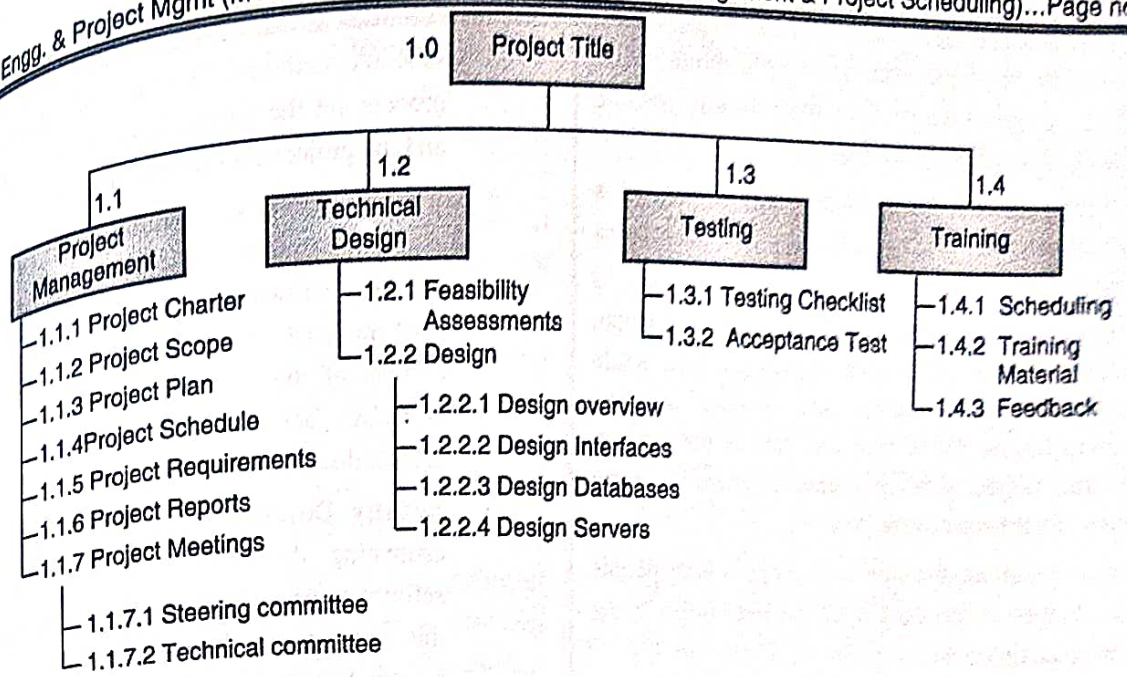


Fig. 6.14.3

Syllabus Topic : Developing the Project Schedule

6.15 DEVELOPING THE PROJECT SCHEDULE

GQ. Briefly describe the process of developing the project schedule.

- In the previous section, we have seen that the top level of the WBS offers high level deliverables each of which are decomposed into detailed and well defined deliverables known as work packages.
- These work packages are further decomposed to form activities in the project schedule. Activities or tasks are a key input of the project schedule. A list of all the activities to be included in the project schedule is tabulated.
- The activity list to be included on the project schedule includes the activity name, the activity code/identifier/number, and a brief description of the activity.
- On the other hand, the activity attributes contains more schedule related information about each activity such as predecessor and successor activity, logical relationship between activities, resource requirement, constraints, dates, and assumptions related to each activity.
- The data for the preparation of the activity list and activity attribute is the WBS and WBS dictionary. As information on the requirements of various activities emerges it is updated in the activity attributes. Activity information is a necessary input for other processes and in activity sequencing, resource allocation, estimating duration, and developing the schedule.
- The project team should get the activity list and attributes reviewed by the project stakeholders before undertaking project scheduling.
- A milestone is a significant event in the life of a project. A milestone in a project is accomplished after the completion of various activities. Although, the milestone has no duration, it marks the completion of various activities and is considered a key event.
- The primary purpose of setting milestones is that they prove useful in setting schedule goals and monitoring the progress of the project activities. Milestones might include completion of certain software modules, installation of hardware, etc.
- Thus, milestone marks the completion of an important stage in the life of the project and the culmination of activities undertaken to reach that milestone.
- However, this does not mean that every project deliverable is a milestone. A milestone is a special event, something which is important for the project and takes it closer to its final destination.
- Project management is a balancing act - balancing scope, time and cost, necessarily in the same order. As

we have seen the project team and the stakeholders first define the project's scope, then the schedule or time and finally the cost of the project.

- The order in which the three primary aspects of a project are addressed reflects the order of the processes in project time management.
- The activity definition further defines the project scope, the activity sequencing further defines the time while activity resource estimating and activity duration estimating further define time and cost of the project. These four project time management processes form the basis of the project schedule.
- The most important resource in IT projects is its people and the biggest component of IT project budget is the cost incurred on them.
- This cost is calculated on an hourly basis and hence the time spent on the project needs to be monitored and managed effectively. In the truest sense, the adage "time is money" is applicable to IT projects.
- Time management is the process of recording and controlling time spent by staff on the project. As time is a scarce and a costly resource within projects, each team member should record time spent undertaking project activities on a timesheet form.
- This will enable the project manager to monitor the amount of time spent undertaking each activity within the project. A timesheet register is also completed, providing a summary of the time spent on the project in total so that the project plan can always be kept up to date.
- Project time management includes the processes required to accomplish the timely completion of the project. Project time management includes the following :
  - **Activity Definition :** This includes identifying the activities in the schedule that need to be performed to accomplish project deliverables. The activities or tasks are found in the project's work breakdown structure (WBS) and have expected duration, cost and resource requirement. The output of this process is an activity list, activity attributes, milestone list and project management plan updates.
  - **Activity Sequencing :** This includes identifying and documenting the dependencies amongst the schedule activities. The primary outputs of this process are the project schedule network diagram and the project documents updates.
  - **Activity Resource Estimating :** This includes estimating the type and quantities of each resource (people, software and hardware) required to perform each schedule activity. The primary outputs of this process are resource breakdown structure, activity resource requirements, and project document updates.
  - **Activity Duration Estimating :** This includes estimating the duration required by schedule activity to be completed. The primary outputs of this process are the activity duration plan and project document updates.
  - **Schedule Development :** This includes analyzing the activity sequences, durations, resources, and schedule constraints to create the project schedule. The output of this process is the project schedule.
  - **Schedule Control :** Controlling changes to the project schedule.
- Although, these processes are shown to be discrete components with well defined boundaries and interfaces in reality they overlap and interact not only with each other but also with processes from other knowledge areas. Each process within the project time management needs the active participation of one or more than one person depending on the size and complexity of the project. The processes will need to be repeated for each phase if the project is divided into phases, if not it will need to be undertaken just once.
- The six processes of project time management are preceded by a planning effort on the part of the project team which produces the schedule management plan.
- The schedule management plan is responsible for developing and controlling the project schedule. The schedule management plan is a subsidiary of project management plan and includes the project time management processes and its various tools and techniques.
- Simply put, schedule management plan involves determining the policies, procedures and documents that will be used for planning, executing and controlling the project schedule.

6.15.1 Project Activities, Attributes and Milestones

Activity Sequencing

- The next step in project time management is determining dependencies and sequencing the activities.
- The inputs to activity sequencing include; project schedule management plan, activity list and attributes, milestone list, project scope statement, and organizational resources. To sequence activities it is necessary to understand their dependencies.
- Dependencies or relationship refers to interlinking of activities where it is not possible to start one activity before the previous activity is completed. Therefore, it is very important to determine the dependency or relationship amongst activities before scheduling.
- The types of dependencies are as follows:
  - **Mandatory dependencies :** These kinds of dependencies are the logical dependencies in nature of the work. Example: it is not possible to create WBS before developing the project scope.
  - **Discretionary dependencies :** Although these form of dependencies are also logical in the nature of work, the project manager may override the logic to commence the activity. Example: the project team should not commence designing activity until the customer has approved the analysis work. However, such form of discretionary dependencies should be managed with care as they may create scheduling problems at a later stage of the project.
  - **External dependencies :** This type of dependency exists between project and non-project activity. However, it is a matter of debate whether any activity that is remotely related to any project activity should be considered to be a non-project activity. Therefore, it is judicious to include the non-project activity in the project scope to establish its relationship with the project activity and its ability to impact the schedule.
- It is necessary to determine dependencies before utilizing any of the project scheduling tools. We shall be studying these tools later on in the chapter.

Activity Resource Estimating

- Resources needed for a project are people, material, tools, equipment, facilities and services. For a project to be successful it is beneficial to know in advance the resources required and when they would be required.
- Resources are required to carry out project tasks and include people, capital, material, facilities, tools, and equipments.
- Lack of resources will be a constraint on the completion of the project. However, the biggest resource in IT Projects is People. Projects require people with the right qualification, experience, and skill set to accomplish the project objectives.
- Depending on the nature of the project, deadline for its completion, and the scope of project especially WBS, the project manager is able to identify the resource requirement for its success. The resource can then be mapped to the WBS to create a Resource Breakdown Structure.
- The Resource Breakdown Structure indicates the moment the resource would be required and the project deliverable it would be creating. As we have seen that there are various projects vying for scarce resources in an organization.
- The resource breakdown structure will be able to utilize these resources optimally by assigning resources to project activities. The project schedule will have to consider the availability of project resources to create deliverables.
- In case the project sponsor has set a deadline even before the project is assigned to the project manager, the project manager should verify the resource requirement based on the project deadline. Projects that fail or are delayed can usually be traced to poor planning, insufficient resources or unrealistic deadlines.
- The time taken to complete a project depends on the number of hours a resource works in a week, the number of days they will be able to work, and the time they can allocate to this project.
- The project manager's first concern during planning is to uncover the skills needed for the project, which tasks will require the most hours and the skills needed to complete those tasks.

- This will enable him to propose assigning a resource with the right skills and the highest availability. To determine availability, first consider the resource's project availability percentage, then adjust for concurrent project demands and planned out-of-office time.
- Find the right balance to solve the resource equation, and the project manager will be able to shorten the project's planned duration and accelerate the finish date.
- While selecting his team the project manager may rely on multiple methods to assess the skills of the proposed team member.
  - **Prior Experience :** The project manager has worked with the team member on prior projects and is very well aware of his capabilities and shortcomings.
  - **Prior Track record :** Although, the project manager may not have directly worked with this team member his track record in the organization makes him a good prospect for being a part of the team.
  - **Management Recommendation :** Not all members are selected by the project manager some may be in the team on recommendation of the management. Either management may feel that they are suitable for the job or wants them to get the experience of working on the project.
  - **Team Member Recommendation :** Existing team members whom you fully trust may recommend others on to the team.
- The project manager might be able to hit the sponsor's initial deadline by using creative resource allocation, such as a trainer with development skills, or a desktop technician who can set up a server, when traditional resources aren't available and delaying isn't a practical option.
- The key to making these kinds of choices, and settling on the most aggressive yet realistic deadline, is to communicate with the resources' managers. Make sure they agree to commit their people to the effort and hold them accountable for their active participation on the project team and timely completion of their assigned tasks.

Activity Duration Estimating

- Activity duration should not be mixed up with effort. Effort is the number of work hours or days that a person or team will take to complete a task it indicates the actual time worked on the task, while duration includes the actual time worked on the task plus the elapsed time.
- For example, even though it may take 10 hours complete the task, the duration estimate might be three days to allow time for other non-project activities that the person may be associated with. Also, the capability of the persons and resources assigned to the task affect the task duration estimate.
- Hence, duration estimate relates to the time estimate on the calendar and not the effort estimate. However, the duration and effort estimate are related and hence both should be taken into account while creating duration estimates.
- The people who will be undertaking the activity should be consulted while estimating the activity duration. The project team should also consult project experts and refer to past experiences of similar activities while estimating activity duration.
- Along with referring to past experiences with similar activities the project team should also review the accuracy of the estimates so far in the project. This review indicates the general pattern that the duration estimates are following. In case the pattern indicates a deviation, appropriate corrections should be carried out in future estimates.
- Another critical factor that needs to be considered while estimating is the availability of resources, especially human resources.
- The project manager needs to analyze the skill requirement of the activities, the skills possessed by the people assigned to the project and the number of people available to work on the project at a given time.
- The activity duration estimates are based on the schedule management plan, activity list, activity attribute, project scope statement, resource requirement, resource breakdown structure, risk register; etc. Based on all these inputs the project manager arrives at a three point estimate for all activities.

- The three point estimate includes an optimistic estimate, a realistic estimate and a pessimistic estimate. The optimistic estimate is based on the best case scenario while the pessimistic estimate is based on the worst case scenario.

Syllabus Topic : Project Schedules

6.16 PROJECT SCHEDULES

GQ. What purpose does the project schedule serve?

GQ. Enumerate the tools available with the project manager in identifying risks.

- Effective Planning and analyzing the plans from different perspectives increase the chances of completing the project on time and within budget. However, before setting the plan in motion it is necessary to work out a proper project schedule and plan the procurement activities listed in the plan.
- Just as every project must have a definitive set of deliverables which mark the closure of the project; it should have a designated start and finish date.
- A project with no starting and finishing date is less likely to succeed. As far as possible the deadlines set for the project should be inflexible. Consider the Y2K bug problem surely this project implemented by every company had an inflexible deadline.
- Although, not all projects have such a demanding and inflexible deadline, however the project manager should set a realistic deadline to ensure that the project meets its logical end. A project missing its deadlines is a sure shot indication of poor planning and a lack of research.

Perhaps one of the biggest challenge and area of conflict for any project manager is to deliver projects on time. Problems of schedule are common, easy to measure and remember.

Project stakeholders and sponsors remember the project's scheduled completion date and quickly estimate the delay in the project without giving a thought to the approved changes in the project that may have lead to the overrun in date.

Thus, time is one variable that has very less flexibility and needs to be respected no matter what happens on the project. Hence, with all possibilities of schedule conflicts it is important for project managers to adopt good project time management.

- All the previously mentioned project time management processes, namely, activity definition, activity sequencing, activity resource estimating, and activity duration estimating, act as inputs in the development of the schedule.
- These processes have to undergo several iterations before the project schedule is finalized. The project schedule determines all the start and end dates of all activities and the project. Schedule development continues throughout the project as work progresses, the project management plan changes, and anticipated risk events occur or disappear as new risks are identified.
- The objectives of developing a project schedule are :
  - To identify the project activities that need to be completed
  - To estimate the duration of the project activities
  - To determine the sequence and time when the project activities will need to happen
  - To identify the resources needed for the activities and when they would be needed
  - To determine how the project schedule can be adjusted for the best possible outcome for the project
  - To monitor and control the project schedule
- The Project Scope Statement enables the creation of the Work Breakdown Statement (WBS) which in turn facilitates creation of the project schedule.
- A project schedule comprises of the various tasks within the project and the resources that would be needed for the completion of the task.
- A project schedule enables the project manager to determine the time that would be needed for the project and accordingly the timing for procurement of the resources required for the completion of the job.
- Based on the criticality of work involved in the task the time period for its completion is estimated by the project manager.
- While estimating the project manager may rely on previous project experience or may discuss the time

frame with the people who will responsible for the completion of the task.

- However, when the task is undertaken it is the responsibility of the project manager to ensure that the requisite resources are made available.
- The estimation of time required for the completion of tasks should be measured in units of time such as days, weeks, or months.
- After the unit of time required for completion of each of the task in the project has been estimated, the project manager can estimate the time required for the completion of the project by merely adding the units of time of each task. Based on the duration estimated a deadline can be set for the completion of the project.
- However, in the real world, project deadlines are determined by management way before the actual project even commences. The delivery dates for the project are given to the project manager who then has to plan backwards.
- Despite the duration of the project having been decided in advance the project manager has to still undertake all the activities of work decomposition and assignment of resources to complete the tasks just as he would have done with a project with no pre-defined deadline.
- Creating a network diagram, also referred to as a logic diagram, is a great way for starting to understand and visualize how the project will proceed.
- It's a simple concept successfully used in applications outside of project management planning; commonly used in manufacturing industries, as well as service industries requiring mapping and understanding work flows.
- In the context of project planning, a network diagram is a sequence of tasks (activities), commonly represented by blocks, that are linked together in the logical sequence they need to be carried out. Producing a network diagram follows the completion of the project WBS.
- Once the logic diagram has been completed, a clear picture of the required sequence of events and activities for the project emerges. The logic rules affecting activities, for example you cannot start building the house foundations until you have first prepared the ground, are evident.
- By including the project team in the development of the network diagram, the project manager obtains a consensus of understanding of how the project should be implemented, at the same time providing a forum for challenging accepted ways of doing things, which may need amending to suit the specific constraints associated with the project at hand.
- Project Schedule serves the following purpose :
  - Project schedule simplifies the project plan such that it can be understood by all those concerned and initiates prompt action.
  - Project schedule validates the time objective by verifying the deadlines set for tasks.
  - The implications of a delay in a particular task can be gauged and proper corrective action can be initiated to avoid repercussions.
  - Project schedule enables the project manager to identify and focus on those activities that could prove to be bottleneck in the future.
  - Project schedule serves as a basis for project monitoring and control.
  - Project schedule ensures the optimum utilization of resources and helps in resource levelling by working on the floats.
  - Project schedule identifies scheduling constraints and enables the project manager to work within the framework of the constraints.

6.16.1 Inputs to Project Schedule Development

- **Organizational Process Assets :** Organizational process assets refers to those assets that can be used to influence the project's success. Organizations involved in project work have their formal and informal policies, procedures, plans and guidelines for conducting project work. Organizational process assets also represent the knowledge and experience gained from previous projects. These policies, procedures, plans and guideline include :
  - Safety and health policy.
  - Quality policy and procedure.
  - Standardized guidelines, work instructions and performance measurement criteria.

- Communication plan.
- Project closure guideline.
- Financial control procedures.
- Issue management procedure.
- Risk control procedure.
- Work authorization procedure.

Organizational process assets include the organizational knowledge base for storing and retrieving information.

- Process measurement database to measure data on processes and products.
- Project files relating to scope, cost, quality baselines, schedule, performance measurement baselines, project calendars, project schedule network diagram, risk registers, etc.
- Knowledge and experience gained from previous projects.
- These organizational process assets can be used in schedule development.

Project Scope Statement : The project scope statement contains assumptions and constraints that can impact the development of the project schedule. Assumptions are those schedule related factors that are considered to be true for schedule development purpose. Constraints are those factors that limit the options when performing schedule network analysis. There are two major time constraints that need to be considered during project schedule development.

Dates imposed on the commencement and completion of activities restricts their start or finish to occur either no earlier than or no later than a specified date. Date constraints may arise due to contract commitments, technology limitations etc.

The key stakeholders of the project often dictate the dates of completion of major milestones of the project. Once these dates have been scheduled it is expected to be adhered and can only be shifted through approved changes. Milestones are often used to indicate interfaces with outside agencies which are not in the ambit of the project. Milestones with constraint dates provide the appropriate schedule interface.

- Project activity list and attributes
- Project schedule network diagram
- Project activity resource estimating
- **Project resource calendar :** A project resource calendar documents working and non-working days, that help determine the dates on which a particular resource, whether a person or material is available for project work. Remember, resources are more than people: material, tools and facilities are resources too. The project resource calendar identifies resource specific holidays and resource availability periods. Thus, the project resource calendar defines when the project resources are needed, when the resources are available, and when the resource will be utilized in the project. Not only does the resource calendar indicate the availability of resource on a particular date it also indicates the quantity of the resource available.
- Project activity duration estimates
- **Project management plan :** The project management plan comprises of the project schedule management plan, project scope management plan, cost management plan, and the risk management plan. These plans and their components support project schedule development.

6.16.2 Schedule Development – Tools and Techniques

Several tools and techniques assist in project schedule development such as Gantt Chart, Critical Path Analysis, PERT, etc. We shall be studying some of these techniques in detail in the coming sections.

6.16.3 Outputs of Project Schedule Development

- The outputs of the project schedule development are as follows:
- **Project Schedule :** The primary objective of a project schedule is to determine a start and finish date for each project activity. The project start and finish date cannot be established until resource assignments have been confirmed and happen only after the completion of the project management plan. The project schedule with the target start and finish dates for each schedule

activity can be presented in either, summary form, tabular form, or graphical form. The graphical form of project schedule uses one or more of the following formats.

- Project schedule network diagram
- Bar charts
- Milestone charts

- We shall be studying these formats in detail in the coming sections.
- **Schedule supporting data** : The supporting data for the project schedule includes milestones, activities, attributes, resource requirement, assumptions and constraints.
- **Updated resource requirement and activity attributes :** Resource levelling has a significant impact on the requirement of resources. The change in resource requirement on account of resource levelling needs to be updated. Similarly the activity attributes are also updated to include any revised resource requirement and other approved changes.
- **Updated project resource calendar** : A change in resource requirement and activity attributes is bound to have an effect on the project resource calendar. The project resource calendar will need to be updated to include the changes in the changed requirements of resources.
- **Requested changes :** The schedule development process is bound to create some requests for changes which need to be processed through the integrated change control process.

Updated project management plan - The project management plan is updated to include the changes approved by the integrated change control process.

6.17 FORMULATING A NETWORK MODEL

Network Planning is the categorisation of activities involved in project implementation in a sequential order followed by a schematic presentation of the activities necessary for the entire project.

The steps in network planning are as follows

- Identify and list the category of activities involved from the start to the completion of the project. The activitie are grouped in categories which are different from eac other.
- Arrange the list of activities in sequential order of thei performance. There may be activity which can be started only after the completion of some other activity, whereas there may also be some other independent activity which can be started simultaneously.
- Also, some activities may be started after the lapse of certain time of a certain previous activity. It is essential that the project manager studies each activity, understands their interrelationship and the duration each activity would take.
- In network planning, such independent and inter-dependent activities are laid down along with their estimated time schedule, i.e. the duration estimated from the start to the completion of the activity.
- Network planning of the project schedule makes the monitoring and controlling of the project easier than to look around the list of activities and locate lapses, if any.
- The network planning as detailed here is a tool available to the project management for a systematic project scheduling.
- Under such method the inter-relationship of the various activities involved in the project schedule will be visible in the total plan and, as such, steps can be taken to economise the consumption of resources, wherever possible.
- Classical approach to project planning such as the Gantt Charts lacked completeness in three fronts.
- These charts failed to show the logical sequence of activities and their interdependence.
- These charts could not indicate how a failure of performance in one task would affect others and the overall project in terms of time and cost.
- These charts could not show the possibilities of reducing time by expediting a limited number of specified tasks and thereby aim at project cost trade off.
- To overcome these three basic limitations of Gantt charts the technique of Network Planning was developed. These limitations of Gantt charts are the strengths of Network Planning.

- Also, the rapid advancement in technology, increased complexity of project planning and the requirement for speedy execution prompted the need for a more versatile project planning technique.
- Network Planning techniques have proved to be an effective management tool for planning, scheduling and controlling of complex projects involving the interlinking of numerous tasks.
- Network Planning is a systematic quantitative structured approach to problem solving in project management. Network Planning is easy to understand and analyze and hence a very popular tool in project planning.
- Network Planning often referred to as logic diagrams are a great way to understand and visualize how the project will proceed.
- Network Planning is a sequence of steps (activities), commonly represented by blocks, that are lined together in the logical sequence in which they are to be performed.
- The network/logic diagram follows the completion of the work breakdown structure. Once the logic diagram has been completed a clear picture of the required sequence of events and activities for the project emerge.
- The project team is able to clearly see the logic rules affecting activities.
- The following logic-network diagram will make things more clear. The example taken to explain the concept is from our daily lives and hence is simple to understand. We shall be studying the making of a cup of tea through the logic-network diagram.

Network Planning explained through the simple process of Tea making

- Imagine that you are making tea. We could break the process down into several jobs or activities, some of which depend on others to have finished before they can start.
- The activities each need one person to do them. No-one can do two things at once. Here is a list of all the jobs:
- Clean cup, boil water, put bag in cup, pour water into cup, add sugar, add milk, drink tea.
- The problem is that you can't pour water into a dirty cup. You must clean the cup before pouring in the water. We can summarize the order in which we do the activities in a precedence table.

Table 6.17.1

Activity	Predecessor Activity
Clean Cup and Saucers (A)	-
Pour water in kettle and boil (B)	-
Put tea bag in cup (C)	A
Put water in cup (D)	B and C
Add Sugar (E)	A
Remove Tea Bag (F)	D
Add Milk (G)	F
Drink Tea (H)	E and G

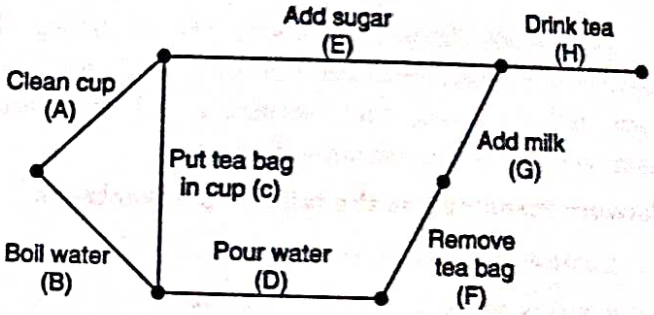


Fig. 6.17.1

- The network developer should identify a few things within the project activities before attempting the network diagram.

Parallel workflows

Activities which can be started at the same time. In the above example getting the cups and saucers and pouring water in the kettle and boiling can start simultaneously. None of these activities rely on each other.

Logical sequence

- The activities should all fit together in a logical sequence. For example milk should not be added until the tea bag has been removed.
- The logical links are referred to as the Finish/Start logic links i.e. the succeeding activities cannot start until the preceding activity has finished.

- The finish/start logic forces a deeper level of understanding of the events and activities which must be carried out.

6.17.1 Evolution of Network Planning

- In the 60's an entirely new approach to planning was evolved by the US Navy assisted by Lockheed Missile Systems division.
- This came to be known as Program Evaluation Review Technique (PERT). On the other hand Remington Rand produced the Critical Path Method (CPM) and this was the first in the stream of industrial applications.
- Over the years the nomenclature of these approaches changed to eventually result into network analysis. Network Analysis is a whole family of which CPM, PERT and Precedence Network Analysis are members.

6.17.2 Advantages of Network Planning

In Network Planning, sequencing and scheduling of activities of a project are treated in a composite frame. This frame sets out tasks, their sequences and also their interrelationship through an arrow diagram.

Network Planning has the following advantages

- It enables thorough planning.
- It increases coordination amongst tasks.
- It identifies bottleneck tasks in advance and enables application of remedial measures.
- It helps to revise thinking and increases the user's awareness of the problems involved and their relative importance in the overall operation.
- It enables the project manager to focus his attention on those activities that are in difficulty rather than on those activities that are progressing smoothly.
- It indicates optimum start and finish time for each activity in an operation.
- It enables faster modifications in the plan to accommodate changing circumstances.
- It permits reporting and facilitates corrective action.
- The most important advantage is that it enables the project manager to control operations and secure completion of the project by keeping a continuous watch on the critical events.
- Reduction in costs by recourse to alternative resource's planning is another added advantage. Thus the advantages provided by Network Analysis prove its superiority over traditional techniques such as Gantt and Milestone chart.
- Critical Path Method (CPM), Program Evaluation and Review Technique (PERT) and Precedence Network Analysis (PNA) are the most widely used network planning techniques.

Syllabus Topic : Network Diagram (AOA)

6.17.3 Steps in drawing a Network - Activity-On-Arrow

- **Step 1 :** Study the project with a proper perspective.
- **Step 2 :** Define broad activities of the project. An activity is the smallest unit of work and has definite duration, definite start and end, logical relationship with other activities in the project, consumes resource, and has an associated cost.

  Activity is denoted by an arrow. Arrow direction shows the project progress. Name of activity is written above the arrow while duration is mentioned below it. Example: for construction of a building procuring material, laying foundation, plumbing, electric wiring, etc are basic activities.
- **Step 3 :** Assign an arrow for each activity with a starting and finishing node termed as start and end events. The beginning or completion of an activity is termed as an event.

  It is also known as node. It indicates a particular instant of time at which some milestone has been achieved e.g. excavation started, concreting finished. It is shown by a circle. Obviously activity starts with an event and ends also with an event.

  Event does not consume any time or any other resource. Usually nodes are numbers, such as 1, 2, 3 etc., to maintain the uniqueness in their identification from left to right in the direction of the progress of the project.
- **Step 4 :** Determine the position of a particular arrow through three basic queries.

  - Which activities must immediately precede this arrow?
  - Which activities cannot be started until the completion of this activity e.g. laying foundation cannot start unless the required material in procured.

- Which activities can be done side by side with this activity e.g. procurement of pipes, paints, etc can be undertaken simultaneously.
- Thus we have three basic forms of arrow diagram :

Arrow Diagram I

Activity A precedes activity B or activity B succeeds activity A. Example: the site will have to be cleaned before the line out can be undertaken.

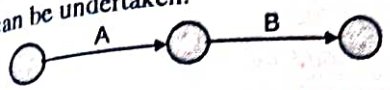


Fig. 6.17.2

Arrow Diagram II

Activity A precedes activities B and C which can run concurrently. Example: once the structure of the house (Activity A) is ready, plumbing (Activity B) and electric wiring (Activity C) can run simultaneously.

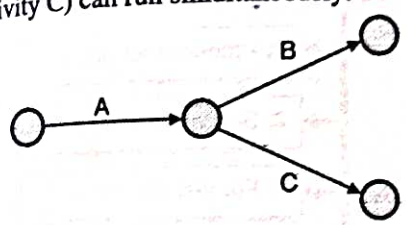


Fig. 6.17.3

Arrow Diagram III

- Activities A and B can run simultaneously. Activity C cannot be undertaken unless both A and B are completed.
- Example: Completion certificate (Activity C) of a new house cannot be obtained unless the finishing and cleaning activities are complete.

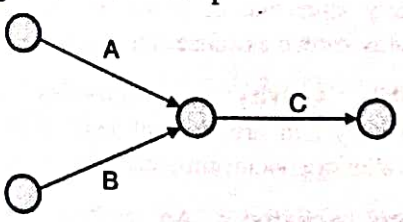


Fig. 6.17.4

Syllabus Topic : Network Diagram (AON)

6.17.4 Events, Activities and their Inter-relationship in Network Diagram

Events in the project diagram represent project milestones, such as the start or the completion of an activity (task) or activities, and occur at a particular instant of time at which some specific part of the project has been or is to be achieved. Events are commonly represented by circle (nodes) in the network diagram.

Events can be classified into following categories

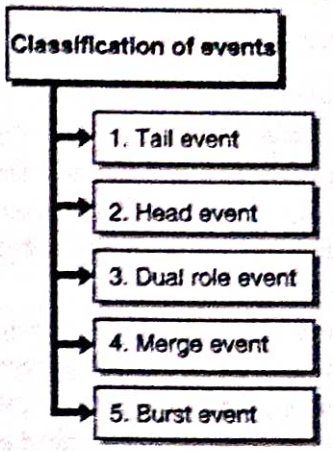


Fig. 6.17.5 : Classification of Events

1. Tail Event

An event which defines the beginning of an activity is called a tail event. It is called so as it is at the tail end of the arrow.

2. Head Event

An event which marks the completion of an activity is called a head event. It is located at the head of the arrow.

3. Dual Role Event

- A node which plays both the roles of head as well tail event is known as dual role event.

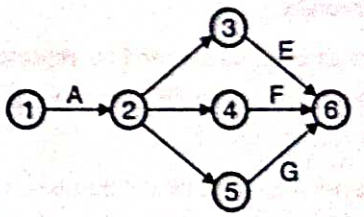


Fig. 6.17.6

- In the Fig. 6.17.6 Node 1 is the tail event of activity A and Node 2 represents the head event or the completion of activity A.
- On the other hand Node 2 is the tail event of activities B, C, & D and is thus a dual role event.
- Further on Nodes 3, 4, & 5 are head events for activities B, C, & D while they are also tail events for E, F, & G thus making them also dual role events.

However, Node 6 is only a head event for activities E, F, & G.

4. Merge Event

An event which represents the joint completion of more than one activity is known as merge event.

5. Burst Event

- An event which represents the initiation (beginning) of more than one activity is known as burst event.
- Events in the network diagram are identified by numbers. Each event should be identified by number higher than allotted to its immediately preceding event to indicate progress of work.
- The numbering of events in the network diagram must start from left (start of the project) to the right (completion of the project) and top to the bottom. Care should be taken that there is no duplication in the numbering.

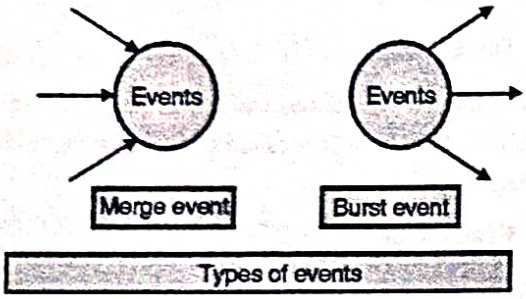
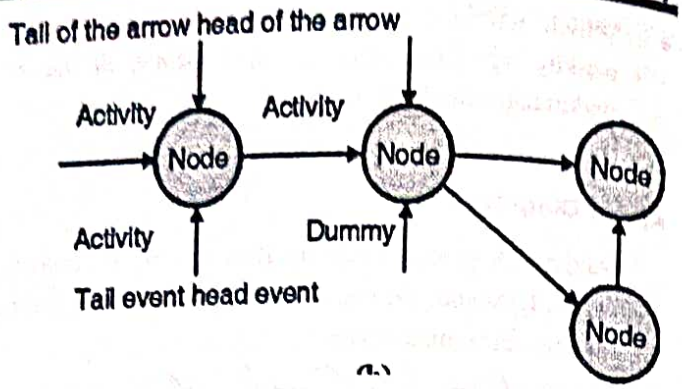


Fig. 6.17.7

Activities

- Activities in the network diagram represent project operations or tasks to be conducted. As such each activity except dummy consumes time and resources and incurs costs.
- An arrow is commonly used to represent an activity with its head indicating the direction of progress in the project.
- Activities are identified by the numbers of their starting (tail or initial) event and ending (head or terminal) event. An arrow (i,j) between two events; the tail event i represents that start of the activity and the head event j represents the completion of the activities shown in the Fig. 6.17.8.

(a)



(b)

Fig. 6.17.8 (a) and (b) : Activity-Node Relationship in Network Diagram

The activities can be further classified into the following three categories

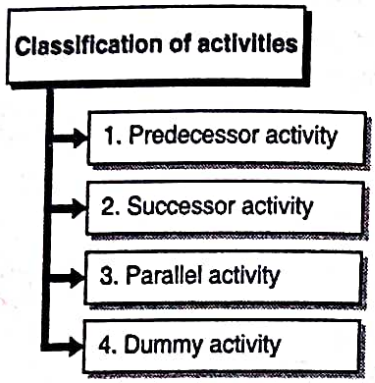


Fig. 6.17.9 : Classification of Activities

1. Predecessor Activity : An activity which must be completed before one or more other activities start is known as predecessor activity.

2. Successor Activity : An activity which starts immediately after one or more of other activities are completed is known as successor activity.

3. Parallel Activity : Activities that occur simultaneously and are independent of each other are called parallel or concurrent activities.

4. Dummy Activity : An activity which doesn't consume either any resource and/or time is known as dummy activity.

- A dummy activity in the network is added only to establish the given precedence relationship among activities of the project and is needed when (a) two or more parallel activities in a project have same head and tail events, or (b) two or more activities have some (but not all) of their immediate predecessor activities in common. A dummy activity is depicted by dotted line in the network diagram.

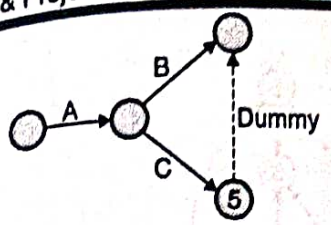


Fig. 6.17.10

6.17.5 Numbering the Events

- It is a practice in Network Analysis to number the events in such a way that the event at a start of an activity has a smaller number than the event at the finish.
- The purpose of such a procedure is two-fold. It enables to identify activities by the numbers of start and end events and helps in the systematic calculation at scheduling stage. Example: activities A to D can be identified by numbers of start and end events.

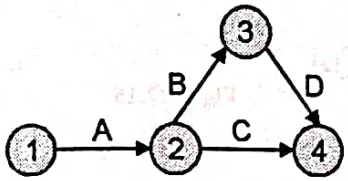


Fig. 6.17.11

Activity	Identified as
A	(1,2)
B	(2,3)
C	(2,4)
D	(3,4)

Fulkerson's Rule for Numbering Events

- The starting event (having no predecessor activity is numbered as 1) other events are numbered in increasing order from 1 to rightwards.
- If there are more than 1 initial event found in a diagram, anywhere they are to be numbered from top to bottom in increase order. No two events can have the same number in any case.

6.17.6 Use of Dummy Arrows for Dummy Activities

- There are two basic needs of dummy or dotted arrows. One is for the uniqueness in identifying the activities.
- **Example :** Suppose activities B, C and D can all start as soon as A is complete and activity E has to wait until B, C, and D are complete. The activities with the event numbers can be shown in the following manner.

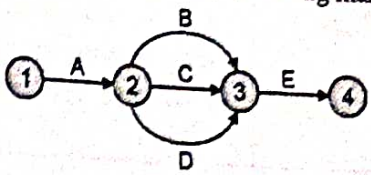


Fig. 6.17.12

- Though the sequence of activities is correctly shown in the Fig. 6.17.12, this would mean that activities B, C, and D would all be denoted by (2, 3). This would create an ambiguity and hence to clear this ambiguity two dummy arrows are introduced into the Fig. 6.17.13.

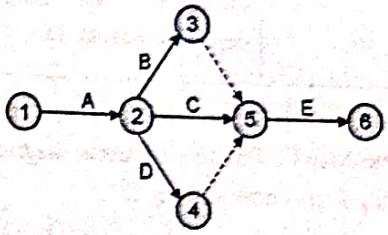


Fig. 6.17.13

- Now activities B, C and D are represented without ambiguity by (2, 3), (2, 5) and (2, 4) respectively. It should be noted that dummy arrows have zero duration and do not consume any resource.
- The second need for dummy arrows is to show correct logical sequence of activities.
- To develop a better understanding of dummy arrows let us study the following example.
- **Example :** Assume your car develops a puncture while you are travelling. Then the activities to be followed would be as follows

Activity Code	Activity Description
A	Remove tyre
B	Repair Puncture
C	Remove spare tyre
D	Fit spare tyre

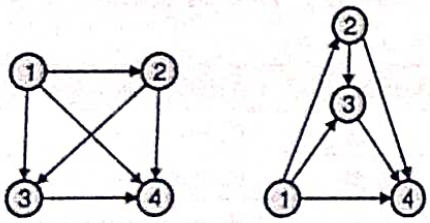
- Now activities B and D must follow activity A and, in addition, activity D must also follow activity C.
- To start with, some wrong arrow diagrams along with mistakes in them are shown for the above. Correct arrow diagram is then drawn by using a dummy arrow.

Sr. No.	Arrow diagram	Comment
(i)		Wrong because activity B in our example need not follow activity C.
(ii)		Through the sequence is correctly shown this arrow diagram is not correct because a single activity A is shown by more than one arrow.
(iii)		Correct the dotted arrow indicates that activity A precedes activity D.

- The first arrow diagram is wrong as activity B need not follow activity C. The second arrow diagram is correct as activity A precedes activity D.

6.17.7 Tips for Drawing Network Diagrams

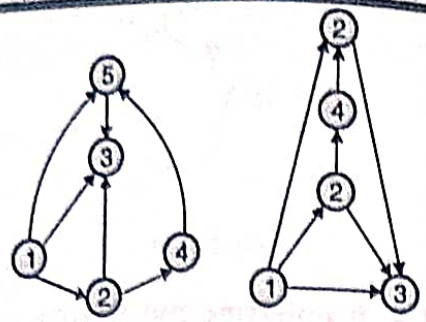
- You may find it difficult to draw network diagrams that are easy to understand initially as the number of individual arrows become fairly large.
- However, with practise you will be able to develop an eye for developing networks. The hints for drawing networks will help you in drawing networks that are easy to understand.
- In the figures below the figure 'b' and not figure 'a' is the preferred way in which network diagrams should be drawn.



(a) (b)

Fig. 6.17.14

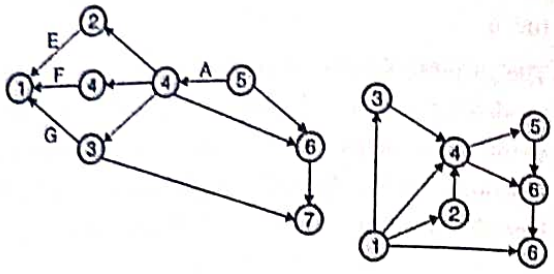
(1) Avoid drawing arrows which cross one another.



(a) (b)

Fig. 6.17.15

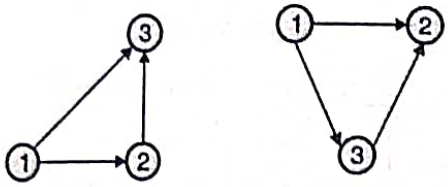
(2) While drawing the arrows should be straight



(a) (b)

Fig. 6.17.16

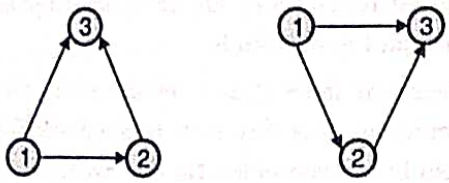
(3) Avoid wide variation in the length of the arrows



(a) (b)

Fig. 6.17.17

(4) The angles between the arrows should be kept as large as possible



(a) (b)

Fig. 6.17.18

(5) A later event should always be drawn on the right side of an earlier event

6.17.8 Drawing Networks – Illustrative Examples

Ex. 6.17.1 : The interdependence of a job consisting of eight activities A to H is given below :

Table P. 6.17.1

Activity	A, B, C	D	E	F	G	H
Predecessor activity	None	B	B	B	A and D	C and E

Draw a neat network and number the events.

☑ **Soln. :** The network is given below :

Events are numbered with the convention that finish event has a higher number than start event for each activity.

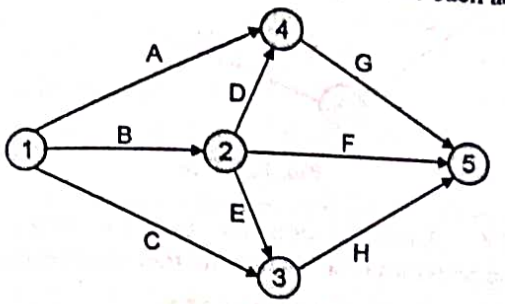


Fig. P. 6.17.1

Ex. 6.17.2 : Interdependence of a project consisting of 11 activities A to K is given below :

Table P. 6.17.2

Activity	Interdependence
A, B, C	Starting activities.
D	Depends on B, controls G and H
E	Depends on C, controls J
F	Depends on A, controls I
G, H	Depends also on C
I	Depends on G
J	Depends on E
K	Depends on I and H.

If J and K are last activities, draw a neat network and number the events.

☑ **Soln. :** The network is given below

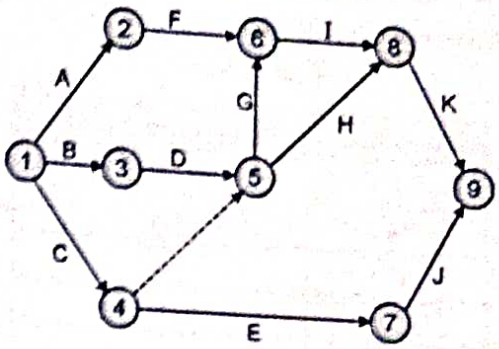


Fig. P. 6.17.2

Events are progressively numbered in the direction of the arrow. One dummy arrow (4 – 5) helps in establishing the logical sequence of activities described in the problem.

Ex. 6.17.3 : A IT project involves the following thirteen activities :

Table P. 6.17.3

Activity	Predecessor Activity
A	-
B	-
C	B
D	A
E	C
F	C
G	F
H	F
I	H
J	I
K	D, E, G, J
L	I
M	K, L

☑ **Soln. :**

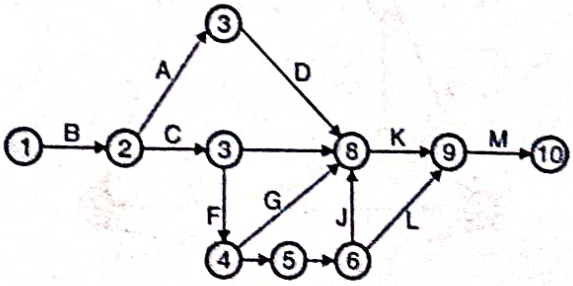


Fig. P. 6.17.3

Ex. 6.17.4 : A small project has ten activities and their interdependence is as follows:

Table P. 6.17.4

Activities	Predecessor Activity
A	-
B	A
C	B
D	B
E	C
F	E
G	D
H	F, G
I	H
J	I

☑ **Soln. :**

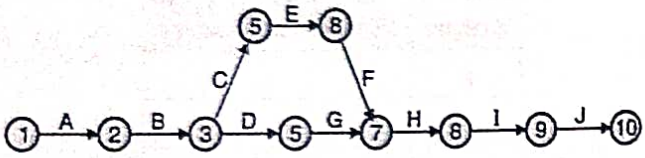


Fig. P. 6.17.4

Ex. 6.17.5 : The interdependencies in a project are as follows: Activities A, B, C and D can start simultaneously right at the beginning. Activity B controls activities E and F. Activities A, C, E and F control activity J. Activities G, H and I cannot start unless activities A and E are completed. Activity G controls activity K. Activity L cannot start unless activities K and H are completed. When activities L, K, J and D are over the project is complete.

☑ **Soln. :**

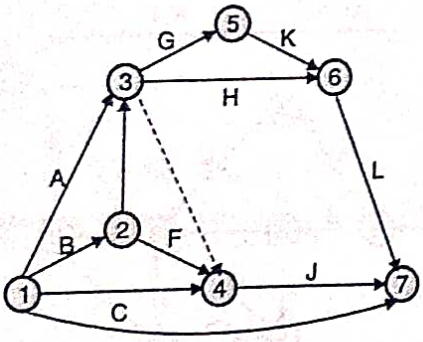


Fig. P. 6.17.5

Ex. 6.17.6 : Draw network diagram by using the following data.

(i) Activities A, B, C are starting activity
(ii) Activity D follows activity C
(iii) Activity A precedes follows activity F
(iv) Activity E follows act B and C
(v) Activity G follows activities E
(vi) Activity G is terminal activity.

☑ **Soln. : Network diagram**

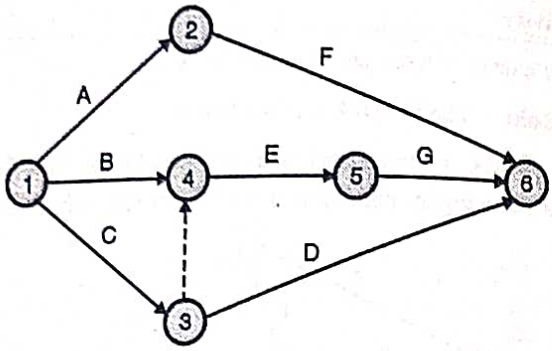


Fig. P. 6.17.6

Ex. 6.17.7 : Draw a network diagram based on the following activities of a Project and their interrelationships.

Table P. 6.17.7

Activity	Predecessor Activity
A	-
B	A
C	B
D	C
E	D
F	E
G	F
H	F
I	F
J	G,H, I
K	H

☑ **Soln. :**

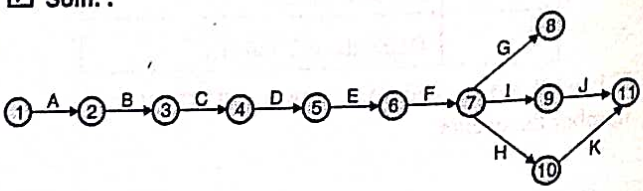


Fig. P. 6.17.7

6.18 NETWORK ANALYSIS

- In the previous section we have understood network as a graphic depiction of activities and events through which the planning, scheduling and controlling of a project becomes much easier. Network analysis plays an important role in project management.
- There are three popular network analysis techniques, namely, Programme Evaluation and Review Technique (PERT), Critical Path Method (CPM) and Precedence Network Analysis or Precedence Diagram Method (PNA or PDA).
- These techniques are used to assist project managers in planning and controlling large scale construction projects like the construction of a residential complex, construction of a dam which would involve the displacement of number of people, installation of a pipeline, construction of a new manufacturing plant, etc.
- The techniques of PERT, CPM and PDA prove extremely valuable in assisting managers in handling projects of such magnitude and complexities, discharging their project management responsibility efficiently and effectively and managing the resources allotted to the project.
- These techniques not only prove useful during the planning stage of the project but also while controlling the project.
- Planning implies developing the overall layout of the project with estimates of the time and resources, detailed scheduling and sequencing of the various activities to be performed. On the other hand, control comes in play during the actual course of the project.
- The detailed utility of the resources enables the project manager to reallocate the resources as per the requirements of the project. These techniques help the project manager to not only determine expected project completion dates but also the expected start and completion dates of the different activities of the project.
- In the previous section we have studied the construction of networks and the principles underlying them. In this section we shall determine which of the activities of a given project are critical and therefore on the critical path. We shall also be determining their scheduling times and the time-cost trade-off. Thus, a project network provides a lot of information about the project.
- Once, the network plan is complete and activity times are known it can be used to schedule various activities, determine the time the project will take to complete and how much cushion is available for performing the activities which will assist in determining which of the activities can be delayed and by how much and which activities cannot be delayed to meet project deadlines. In this section we shall be considering the analysis of the network.

6.18.1 Scheduling Activities

- Once the project network plan is complete and the time taken by each activity determined it is time to know how long the project would take to complete and when the activities should be scheduled. The arrow diagram with the time duration of each activity on them provides answer to these questions.
- The computation of the time duration involves undertaking a forward and backward pass through the network. The forward pass computations provides us with the earliest start and earliest finish times for each activity while the backward pass while the backward pass computations provide the latest start and latest times for each activity.
- The example below will help us demonstrate the calculation of earliest start, earliest finish, latest start and latest finish times of various activities of a project.

Ex. 6.18.1 : From the following activities and their durations calculate the earliest start (ES), earliest finish (EF), latest start (LS) and latest finish (LF) times of each activity.

Activity	Activity Node	Duration
A	1-2	4
B	1-3	14
C	1-4	16
D	2-5	6
E	3-5	12
F	3-6	20
G	3-7	8

Activity	Activity Node	Duration
H	4-6	12
I	5-7	4
J	6-8	10
K	7-8	12

☑ **Soln. :**

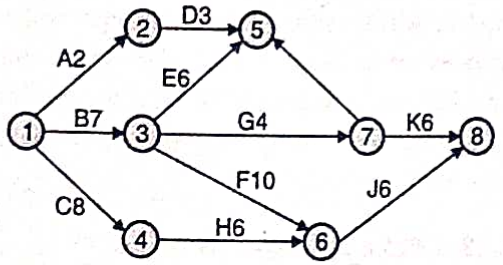
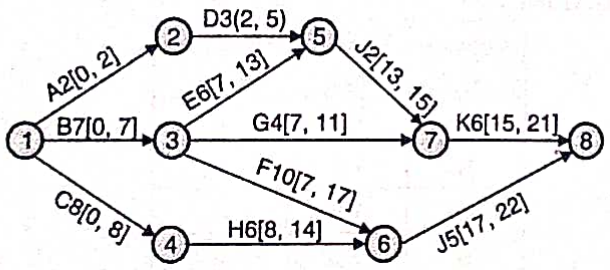


Fig. P. 6.18.1

Determining Forward Pass : Earliest Start and Earliest Finish

- Forward Pass provides us with the Earliest Start (ES) and Earliest Finish (EF) time of each activity. Earliest Start time indicates the earliest time that an activity can be scheduled while Earliest Finish indicates the earliest time that the activity can be finished.
- For this purpose the first activity in any project is assigned a value of 0 which is the first day of the beginning of the project. Now, in our above example activity A, B and C can all start on the first day and hence have been assigned 0 as the earliest time of their start.
- The earliest finish time of each activity can now be obtained by adding the duration of the activity to the earliest start time. So, we get the following :



i. Activity A

Earliest Finish = Earliest Start + Duration of Activity

$= 0 + 2 = 4$

- So activity A's earliest finish can be on the 2nd day. Similarly activity B and C who are also have an earliest start of 0 can have earliest finish of 14 and 16 respectively.
- For the subsequent activities we assume that they too will start as soon as their predecessor activity is completed and hence the Earliest start (ES) of these activities will be the earliest finish (EF) of the preceding activity.

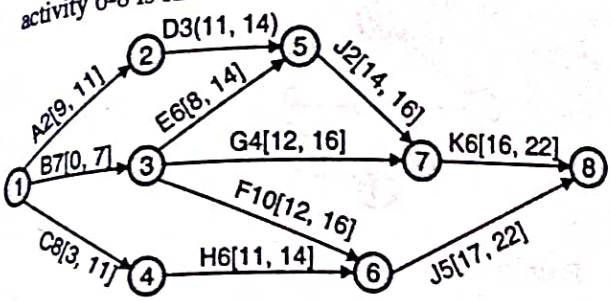
ii. Activity D

- Now, activity D succeeds activity A and hence the earliest finish time of 4 days acts as the earliest start of activity D. Therefore the earliest finish time of activity D is 4 + its duration time of 6 which gives us 10 days.
- The same computation can be used for other activities. However, when a succeeding activity has two or more preceding activities the activity with the maximum earliest finish time should be taken as its earliest start time.
- The earliest start and finish time are depicted in the bracket along with the activity and its duration on the arrow.
- The figure below will give you a clear idea of how the network diagram will look after all the timings have been displayed on it.

Determining Backward Pass : Latest Start and Latest Finish

- The backward time is used to calculate the latest allowable times of starting (LS) and finishing (LF) of each of the activity of a project.
- The computation of the backward pass is very much similar to that of the forward pass. The 'latest allowable time' means the final activity of the project should occur on or before some arbitrary scheduled time or in case that is not allotted then the earliest finish time of the final activity as per the forward pass is taken as the latest finish and the computation is done backward. Continuing with our above example the activities 7-8 and 6-8 are our terminal activities with earliest finish times of 42 and 44 respectively.

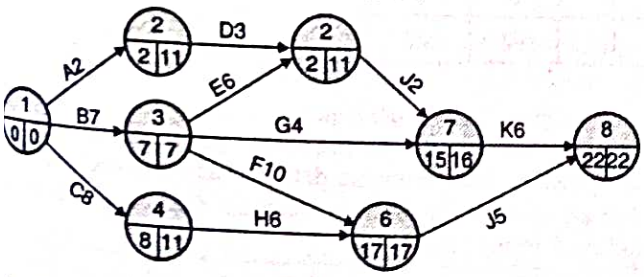
- Hence, we assign a time of 44 to activity 8 which means that both these activities would have their latest finish (LF) as 44. Now, the latest start (LS) for any activity is latest finish time (LF) minus duration. So the latest start for activity 7-8 will be 44-12 = 32 and for activity 6-8 is shall be 44-10 = 34.



- In case an activity has two succeeding activities then the latest allowable finish time for that activity would be the smallest of the latest start time of the succeeding activity.
- The network diagram below depicts latest start and latest finish time of all the activities.

Depicting the earliest and latest timings of events on a network diagram

- In addition to depicting the earliest and latest timings for each activity we shall be needed to depict the earliest and latest times of various events on the network. These times are depicted in a circle along with the event number, its earliest and its latest times.
- In a forward pass the earliest time for the initial event of the project would be 0 and for subsequent events the earliest time would be taken as the latest of the earliest finish (EF) of the activities concluding on that event. As for the terminal event of the project it is assigned latest time equal to its earliest time.



- Once the terminal event has been assigned its latest time the process is rolled back and latest time for events that precede the terminal activity is assigned.
- The latest time of an event is computed by deducting the duration of the activity from the latest time of the previous event. So, in our example the latest time for event 7 would be the latest time of the terminal activity that is 44 – the duration of the activity which is 12 thus giving us its latest time as 32 (44-12).
- However, when more than one activity starts from the node than the lower value of the differences should be taken as the latest time for the event. In our example activity 3 has three activities starting from it now when we compute the difference we find that for event 5 the value is 16 (28-12), for event 5 the value is 14 (34-20) and for event 7 the value is 24 (32-8). As we are to take the lowest value we take 14 as the latest time for event 3. The latest time for other events can be calculated similarly.

6.18.2 Critical Path

GQ. What is a critical path? Why is the longest path the critical path?

Introduction

- In the previous section we have computed the earliest and latest time of all the project activities. Once that is over with, as project manager you would be eager to know the minimum time required for the completion of the entire project.
- For this you would be required the critical path of the project. The critical path denotes the sequence of events that determine the project completion date. It is the longest path from project start to completion.
- Every project has multiple paths from project start to completion but there is only one path that takes longer than any other path in the project network diagram and this is our critical path.
- A path is series of continuous activities thought the network that start from the initial node of the network to its final node. Now, every network has multiple paths that start from the initial node and culminate at the final node. However, all these paths are not critical paths. The critical path is the longest path.

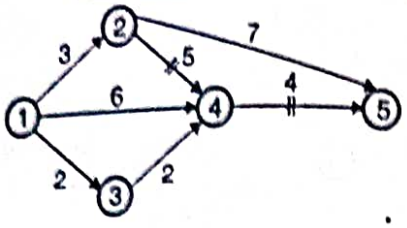
Why is the longest path the critical path?

- This is because a delay in any activity on this path is going to lead to a delay in the completion of the project. However, a project could have more than one paths vying for the title of critical path. The Critical

path is the continuous chain of critical activities in a network diagram.

- It is the longest path starting from start to completion and is shown by a thick line or double lines crossing the activity line in the network diagram.
- In some cases a project may have more than one critical path. This means that there may be two or more than two paths that may be consuming the same time for their completion. In such case all these paths are considered to be critical to the completion of the project and are thus critical paths of the project.
- **Example :** in the table below are given activities of a construction period along with their duration. Draw a network diagram and also identify the critical path.

Activity	Duration
1 - 2	3
1 – 3	2
1 – 4	6
2 – 4	5
2 – 5	7
3 – 4	2
4 – 5	4



- Four different paths can be traced from beginning to end of the network and total time for each path can be calculated as follows.

Sr. no.	Path	Total duration (days)
1	1 – 2 – 5	3 + 7 = 10
2	1 – 2 – 4 – 5	3 + 5 + 4 = 12
3	1 – 4 – 5	6 + 4 = 10
4	1 – 3 – 4 – 5	2 + 2 + 4 = 8

- The path which takes the longest time is called the critical path. Here second path, i.e. 1–2–4–5, is longest and hence the critical path.

Activity	Normal time (days)	Earliest Time		Latest Time		Total Float
		Start	Finish	Start	Finish	
1 – 2	3	0	0	3	3	0
1 – 3	2	0	0	2	6	4
1 – 4	6	0	0	8	8	2
2 – 4	5	3	3	8	8	0
2 – 5	7	3	3	12	12	2
3 – 4	2	2	6	8	8	4
4 – 5	4	8	8	12	12	0

- As the total float of activities 1 – 2, 2 – 4, and 4 -5, is 0 they are the critical activities of the project and the critical path is 1-2-4-5. Also, based on the size of the total float activities 1 -4, and 2 -5 are sub-critical activities and activities 1-3 and 3-4 are non-critical activities.
- However for bigger networks, a more systematic approach is needed. It is possible to indentify the critical path by study of earliest and latest times at which an event can occur.

Ex. 6.18.2 : The following data pertains to a small project. You are to draw the network diagram, find project duration and mark critical path.

Event	A	B	C	D	E	F	G	H
Activity	1-2	1-3	1-4	2-3	2-5	3-5	4-6	5-6
Duration	2	4	3	0	1	6	5	7

▶ **Step 1 :** We will first be drawing the network diagram based on the data in the table. The number next to

activity on the arrow denotes the duration of the activity and the one in the bracket denotes the manpower required to complete the activity. Activity D is a dummy activity and hence has no duration.

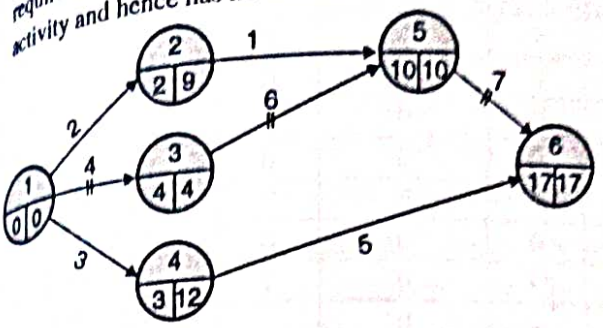


Fig. P. 6.18.2

Step 2 : In the next step we shall calculate the earliest and latest start and finish times of all the activities. This will denote the project duration which in this case is 17 days. The critical path is 1-3 – 5 – 6.

Activity	Normal time (days)	Earliest Time		Latest Time	
		Start	Finish	Start	Finish
1 – 2	2	0	2	7	9
1 – 3	4	0	4	0	4
1 – 4	3	0	3	9	12
2 – 3	0	0	0	0	0
2 – 5	1	2	3	9	10
3 – 5	6	4	10	4	10
4 – 6	5	3	8	12	17
5 – 6	7	10	17	10	17

Step 3 : Conditions of Critical Path

The following conditions are applicable for locating the critical path in a network.

(i) **Condition 1 :** When the earliest start time (ES) of an activity equals the latest finish time (LF) at the head of the arrow and if the ES equals the LF at the tail of the arrow, then the activity is probably a critical activity. E.g. for activity 3 – 5 the ES at the head of the arrow i.e. activity 5 – 6 is 10 which is the same as the LF while the ES at the tail of the arrow is 4 which also coincides with the LF.

(ii) **Condition 2 :** once the first condition has been met then only can we test the activity for the second condition for qualifying as a critical activity. Under the second condition the difference between the ES time at the head of the arrow and the ES time at the tail of the arrow should be equal to the duration of the activity. E.g. for activity 3 – 5 the difference between the head of the arrow and the ES at the tail of the arrow is 6 which is exactly the duration of the activity. Thus activity 3 – 5 is a critical activity. Similarly other activities can be tested to find out whether the lie on the critical path.

- The CPM uses the Network Model in which activities are depicted as nodes on the network and events signify the beginning/ending of activities and are depicted as lines between the nodes.
- The objective of critical path analysis is to estimate the total project duration and to assign starting and finishing times to all activities involved in the project. This helps in checking actual progress against the scheduled duration of the project.

6.18.3 Type of Floats

GQ. What is a float ? State the types of float.

- We are aware that the activities on the critical path cannot be delayed but can the other activities that are not on the critical path be delayed.
- Yes, the activities not on the critical path can be delayed without any impact on the project or succeeding activity, although there is a limit to the amount of time they can be delayed. This limit is the called the float. There are four types of floats:

I. Total Float

- The total float of an activity represents the time by which the activity can be delayed without delaying the completion of the project.
- Thus, it refers to the amount of free time available with the activity that can be used for altering the early start of the activity. Total float is the difference between the total time available for the performance of the activity and the time required for its performance.

- **Example:** Type of Floats and their computations:

Activity	Normal time (days)	Earliest Time		Latest Time		Total Float	Interfering Float	Free Float	Independent Float
		Start	Finish	Start	Finish				
1 – 2	2	0	2	7	9	7	7	0	0
1 – 3	4	0	4	0	4	0	0	0	0
1 – 4	3	0	3	9	12	9	9	0	0
2 – 3	0	0	0	0	0	0	0	0	0
2 – 5	1	2	3	9	10	7	0	7	0
3 – 5	6	4	10	4	10	0	0	0	0
4 – 6	5	3	8	12	17	9	0	2	0
5 – 6	7	10	17	10	17	-	0	-	0

Thus;

Total Float = Latest Start Time – Earliest Start Time

Or

= Latest Finish Time – Earliest Finish Time

Or

= Latest Finish Time – Earliest Start Time – Duration

- ∴ For activity 1 – 2 the total float is 7 which can be either (7-0) or (9-2) or (9-0-2). On the other hand the total float for activity 1 – 3 is 0 (0-0) or (4-4) or (4-0-4) as this activity lies on the critical path.
- The activity can be delayed by the amount of total float available with it. If it is delayed by more than the total float it would delay the completion of the project. However, total float should never be studied in isolation.

II. Interfering Float

- There are chances that the utilization of the total float of an activity may affect the float times of other activities in the network. Thus, that part of the total float that causes a reduction in the float of the successor activity is called as interfering float.
- Suppose activity 1- 2 in the above table is delayed by 3 days and begins at t = 3 instead of t = 0 and it would end at t = 5. This delay will affect its succeeding activity which is 2 – 5. Due to the delay in preceding activity 1 – 2 the float of activity 2 – 5 is reduced to 4 (9-5) from 7 (9-2). Thus the utilization of float of 1 – 2 is seen to affect the float of its successor activity 2 - 5.
- Thus, interfering float is the difference between the latest finish time of the activity under consideration and the earliest starting time of the succeeding activity. Interfering float is that portion of the total float that cannot be consumed without affecting the float of the succeeding activity.

Thus :

Interfering Float = Latest Finish of Activity

– Earliest Finish of the succeeding activity

III. Free Float

- As the name indicates free float is that part of the total float that can be used without affecting the float of the succeeding activities. Thus, it is that quantum of the total float which can be consumed without affecting the early start of the succeeding activity. E.g. let us consider activity 2 – 5 which has a total float of 7 days.
- Suppose we delay this activity by 5 days so that it starts at t = 2 + 5 = 7 and completes at t = 10. The succeeding activity 5 – 6 starts only at t = 10. Thus the delay in 2 – 5 does not cause a delay in the start of the successor activity.

Thus:

Free Float = earliest start time of succeeding activity

– earliest finish time of this activity

So the free float for activity 2 – 5 is

= 10 – 3 = 7. So the free float for the activity is 7.

- Free float indicates the amount of time an activity can be delayed beyond its earliest starting time without affecting the earliest start of the succeeding activity and the total float of the other succeeding activities.

iv. Independent Float

- The independent float time of an activity is that amount of float time that can be used without affecting the preceding or the succeeding activity.
- Thus, independent float represents the amount of time available with an activity when its preceding activity are completed at their latest finish time and the succeeding activities begin at their earliest time.
- This invariably means that the activity under consideration is left with minimum time for its completion. Any time available with the activity over the minimum time allotted for its completion is called as independent float.

Thus:

Independent Float = Earliest start time for succeeding activity – latest finish time for preceding activity

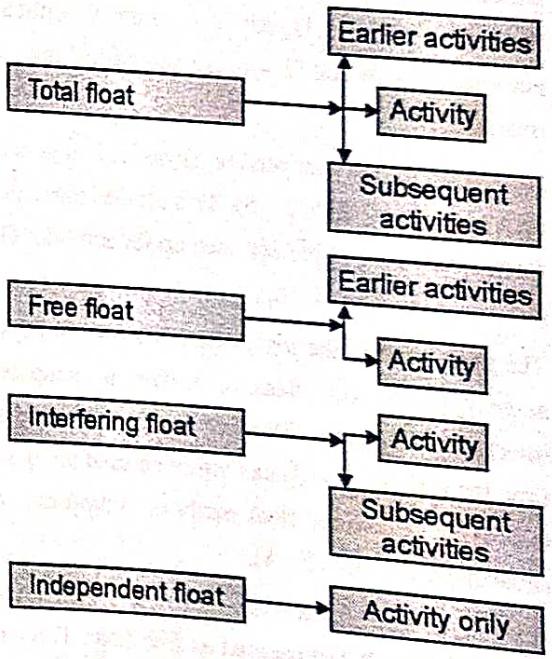
– duration of activity under consideration.

Calculating the independent float for activity 2 – 5

- The preceding activity is 1 – 2 with latest finish time of 9 days, the succeeding activity 5 - 6 has earliest start time of 10 days and the duration of activity 2 – 5 is 1 day. Thus the independent float is = 10-9-1 = 0.
- A negative independent float is taken as 0 independent float.

Interpretations

- Total float is a measure of criticality for an activity. The necessary and sufficient condition for an activity to be critical is that its total float is zero.
- Based on magnitude of total float we can categorise the activities of a project into critical, subcritical and noncritical ones.
- Any delay in executing any of the critical activities will result in delaying the overall project by the corresponding extent. Float of an activity can be used to balance the requirement of resources and / or to reduce the cost or duration of the project. Total float can be used on the activity or can be passed on to earlier and / or to subsequent activities.
- Free float must be absorbed on activities in the chain terminated by the activity under consideration. It cannot be passed on to the subsequent activities. After use in earlier activities whatever float is left as balance can be used freely for the particular activity. Hence the name free float.
- Interfering float can be used for the activity or it is possible to use it up (partly or completely) on the subsequent activities. It cannot be used for earlier activities. In other words, interfering float is that portion of total float of an activity which causes losses in the normal float of subsequent activities.
- This float, therefore, cannot be consumed without affecting the float of subsequent activities i.e. it interferes with float of subsequent activities; hence the name. Finally independent float is the spare time which must be absorbed solely on the activity under consideration.
- The above descriptions on floats can be easily visualised as follows :



Example 1

Calculation of various floats of the network given in figure is brought out in table below.

Activity	Duration (d)	Start event		Finish event		Floats			
		Earliest time (e)	Latest time (l)	Earliest time (E)	Latest time (L)	Total L–e–d	Free E–e–d	Interfering L–E	Independent E–l–d
1–2	3	0	0	3	3	0	0	0	0
1–3	2	0	0	2	6	4	0	4	0
1–4	6	0	0	8	8	2	2	0	2
2–4	5	3	3	8	8	0	0	0	0
2–5	7	3	3	12	12	2	2	0	2
3–4	2	2	6	8	8	4	4	0	0
4–5	4	8	8	12	12	0	0	0	0

Interpretation of various floats computed above runs as follows

- Total float is zero for activities (1 – 2), (2 – 4) and (4 – 5). Accordingly these are critical activities and the critical path is 1 – 2 – 4 – 5.
- Based on the relative magnitude of total float we notice that activities (1 – 4) and (2 – 5) are subcritical and activities (1 – 3) and (3 – 4) are non critical ones in our example.
- Next we interpret independent floats of 2 days each for activities (1 – 4) and (2 – 5). This implies that 2 days of float should be completely used up for activities (1 – 4) and (2 – 5) independently.
- The spare time in the job chain 1 – 3 – 4 is 4 days. For activity (1–3), total float of 4 days is composed of interfering float only. This implies that 4 days of spare time for activity (1 – 3) can either be used up by it or it is possible to use the float partly or completely on the subsequent activity (3 – 4).
- Analysis of float for activity (3 – 4) shows that its total float of 4 days is represented as free float. This clearly indicates that it must be either used up on activity (3 – 4) or the previous activity (1 – 3). However, it cannot be passed on to subsequent activities.

Ex. 6.18.3 : A project consists of 12 activities. The duration of the activities and their relationship is as under :

Activity	Predecessor	Duration (Days)
A	-	7
B	-	5
C	A	10
D	A	5
E	B	8
F	B	6
G	C	5
H	C	4
I	D and E	10
J	F	5
K	G	8
L	H	9

Draw a CPM network and calculate project duration and highlight the critical path. Calculate EST, EFT, LST, LFT, TF, FF, INDF, INTF for all the activities.

☑ **Soln. :**

- The network is drawn as per the above activity relationship and using forward pass and backward pass, T_E, T_L, EST, EFT, LST and LFT are calculated which we have seen in section 2.16 of this book.
- Then all the floats TF, EF, INDF, INTF are calculated as per the section 2.20. The result is tabulated as below :

twork diagram

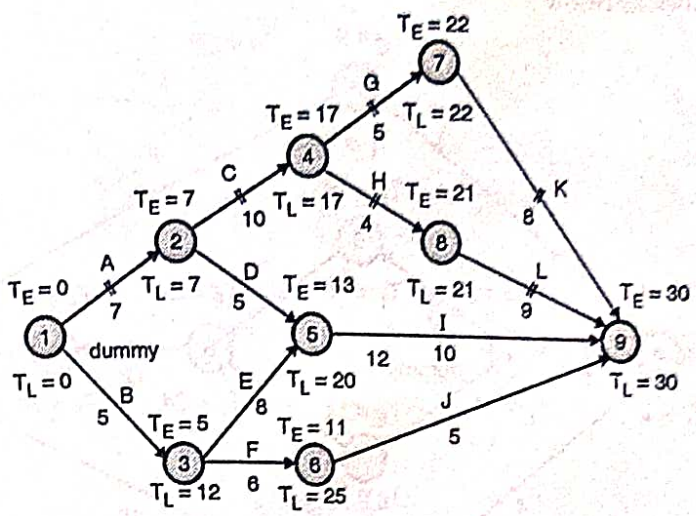


Fig. P. 6.18.3

ritical path

A – C – G – K both the critical path 2. A – C – H – L

Project duration = 30 days

C = Critical activity

NC = Non critical activity

Table P. 6.18.3

Activity	EST	EFT	LST	LFT	TF	FF	INDF	INTF	Remark
A	0	7	0	7	0	0	0	0	C
B	0	5	7	12	7	0	0	7	NC
C	7	17	7	17	0	0	0	0	C
D	7	12	8	20	8	1	1	7	NC
E	5	13	7	20	7	0	–7	7	NC
F	5	11	19	25	14	0	–7	14	NC
G	17	22	17	22	0	0	0	0	C
H	17	21	17	21	0	0	0	0	C
I	13	23	20	30	7	7	0	0	NC
J	11	16	25	30	14	14	0	0	NC
K	22	30	22	30	0	0	0	0	C
L	21	30	21	30	0	0	0	0	C

Ex. 6.18.4 : Listed below are the activities of a project along their durations.

Activity	1-2	2-3	2-4	2-5	3-10	4-6	4-7	5-10	6-8	7-8	8-9	9-10
Duration (days)	4	5	7	4	15	7	Dummy	10	6	7	12	10

(i) Draw AOA network and calculate the total project duration. Highlight the critical path.

(ii) Calculate EST, EFT, LST, LFT, Total float, Free- float and Independent float. Write in tabular form with sample calculation.

☑ **Soln. :** AOA network is drawn as below :

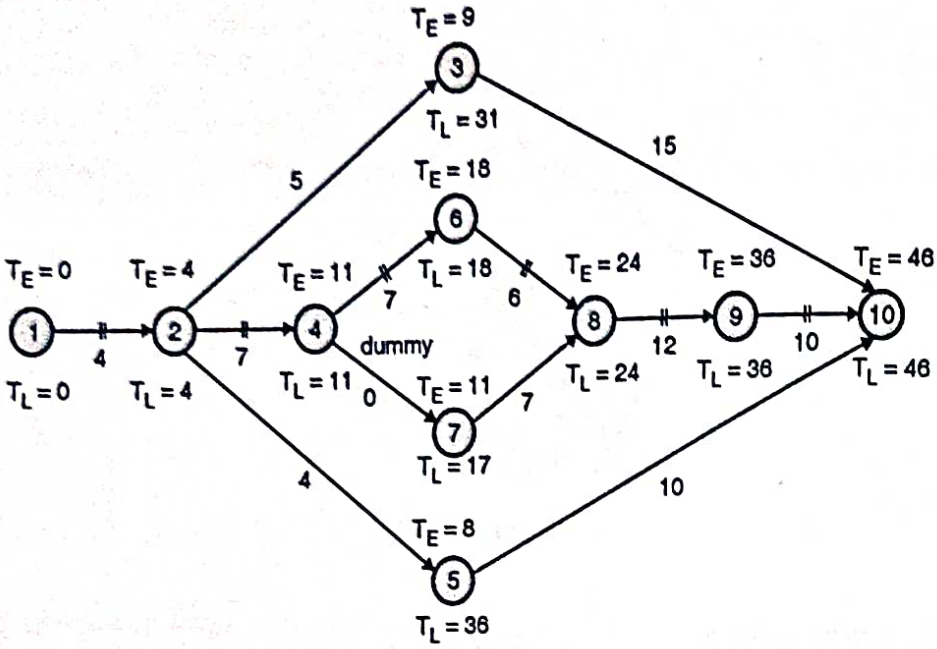


Fig. P. 6.18.4

Project duration = 46 days

Critical path = 1 – 2 – 4 – 6 – 8 – 9 – 10

Activity	EST	EFT	LST	LFT	TF	FF	INDF	Remark
1 – 2	0	4	0	4	0	0	0	C
2 – 3	4	9	26	31	22	0	0	NC
2 – 4	4	11	4	11	0	0	0	C
2 – 5	4	8	32	36	28	0	0	NC
3 – 10	9	24	31	46	22	22	0	NC
4 – 6	11	18	11	18	0	0	0	C
4 – 7	11	11	17	17	6	0	0	NC dummy
5 – 10	8	18	36	46	28	28	0	NC
6 – 8	18	24	18	24	0	0	0	C
7 – 8	11	18	17	24	6	6	0	NC
8 – 9	24	36	24	36	0	0	0	C
9 – 10	36	46	36	36	0	0	0	C

Ex. 6.18.5 : For the work consisting following six activities as shown in table below, work out critical path and days for completion of the work.

Activity (i – j)	A	B	C	D	E	F
Activities immediately preceding	–	–	A	A, B	B, C, D	A, D
Activities immediately succeeding	C, D, F	D, E	E	E, F	–	–
Duration (Days)	5	3	4	3	2	4

Soln. :

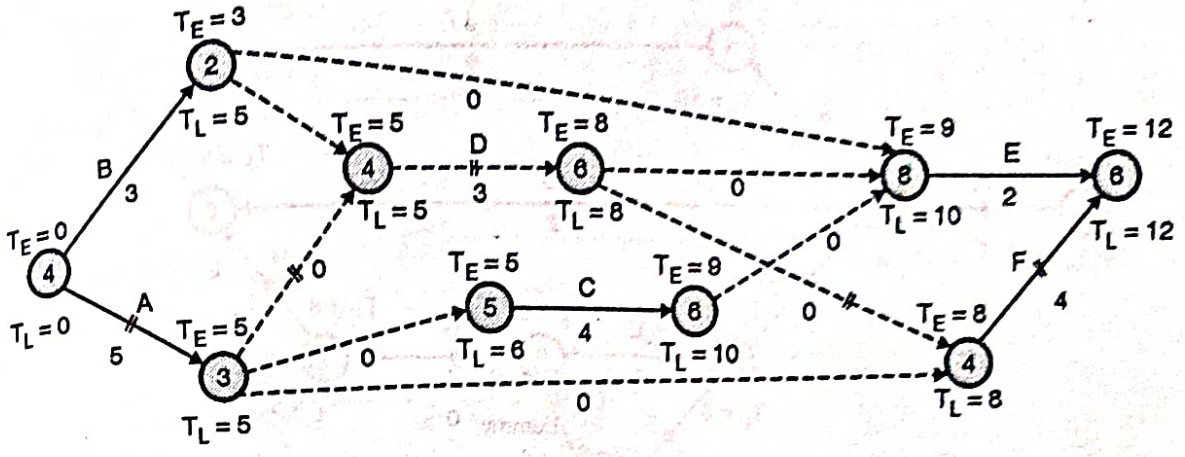


Fig. P. 6.18.5

Here in table dummy activities which are shown by dotted lines in the network are not shown

Total project duration = 12 days

Critical path = 1 – 3 – 4 – 6 – 9 – 10

Activity	EST	EFT	LST	LFT	TF	FF	INDF	Remark
A	0	5	0	5	0	0	0	C
B	0	3	2	5	2	0	0	NC
C	5	9	6	10	1	0	– 1	NC
D	5	8	5	8	0	0	0	C
E	9	12	10	12	1	1	0	NC
F	8	12	8	12	0	0	0	C

Ex. 6.18.6 : Listed below are the activities of a project along their durations.

Activity (i – j)	A	B	C	D	E	F	G	H	I	J
immediately preceding	None	A	A	A	D	D	E, F	G	C, H	B
Duration (Months)	1	4	2	2	1	3	2	1	3	2

(i) Draw AOA network and calculate the total project duration. Highlight the critical path.

(ii) Calculate EST, EFT, LST, LFT find out TF, FF, IF. Write in tabular form with sample calculation.

☑ **Soln. :**

Listed below are the activities of a project along their durations

Activity (i – j)	A	B	C	D	E	F	G	H	I	J
Immediate predecessor	–	A	A	A	D	D	E, F	G	C, H	B
Duration	1	4	2	2	1	3	2	1	3	2

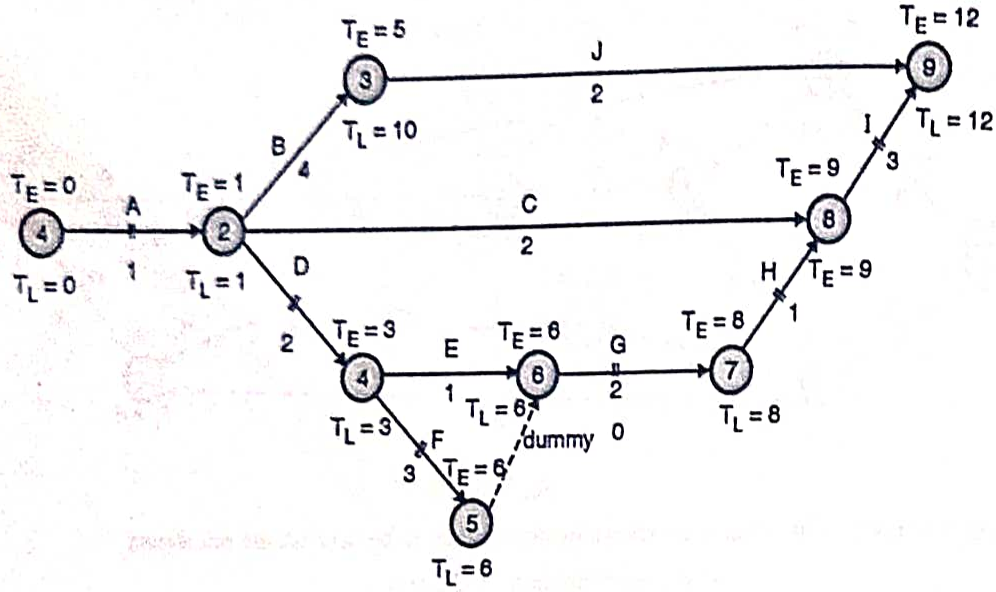


Fig. P. 6.18.6

Total project duration =12 months

Critical path = 1 – 2 – 4 – 5 – 6 – 7 – 8 – 9

Activity	EST	EFT	LST	LFT	TF	EF	INDF	Remark
A	0	1	0	1	0	0	0	C
B	1	5	6	10	5	0	0	NC
C	1	3	7	9	6	6	6	NC
D	1	3	1	3	0	0	0	C
E	3	4	5	6	2	2	2	NC
F	3	6	3	6	0	0	0	C
G	6	8	6	8	0	0	0	C
H	8	9	8	9	0	0	0	C
I	9	12	9	12	0	0	0	C
J	5	7	10	12	5	5	0	NC

NOTES

Data pertains to small project, draw the network diagram. Find project duration and mark critical path.

Activity	1 – 2	1 – 3	1 – 4	2 – 3	2 – 5	3 – 5	4 – 6	5 – 6
Duration	2	4	3	0	1	6	5	7

letwork diagram

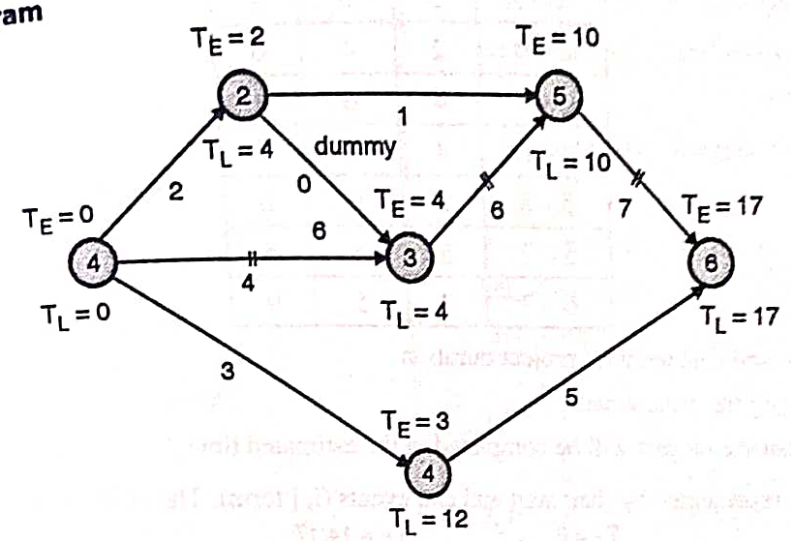


Fig. P. 6.18.7

l project duration = 17 days Critical path = 1 – 3 – 5 – 6

: Calculate EST, EFT, LST, LFT, IDF and IF for following activities. Tabulate the result with sample n.

Activity	1-2	1-3	2-4	3-4	4-5
Duration in 'Days'	2	3	4	5	6

.:

Activity	EST	EFT	LST	LFT	TF	EF	INDF	INTF
1 – 2	0	2	2	4	2	0	0	2
1 – 3	0	3	0	3	0	0	0	0
2 – 4	2	6	4	8	2	2	0	0
3 – 4	3	8	3	8	0	0	0	0
4 – 5	8	14	8	14	0	0	0	0

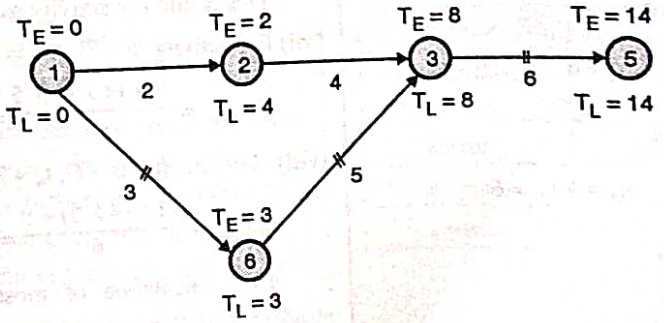


Fig. P. 6.18.8

Ex. 6.18.9 : For the given project, following are the activities as below :

Activity	Duration in weeks		
i – j	t_o	t_m	t_p
1 – 2	1	2	3
2 – 3	3	6	9
2 – 4	2	4	6
3 – 5	4	6	9
4 – 6	4	6	8
5 – 6	0	0	0
5 – 7	3	4	5
6 – 7	2	5	9

(a) Draw the project network and find the total project duration

(b) Calculate the variance along the critical path.

(c) What is the probability that the project will be completed in the estimated time ?

☑ **Soln. :** Here activities are represented by their start and end events (i, j form). The network is drawn as below

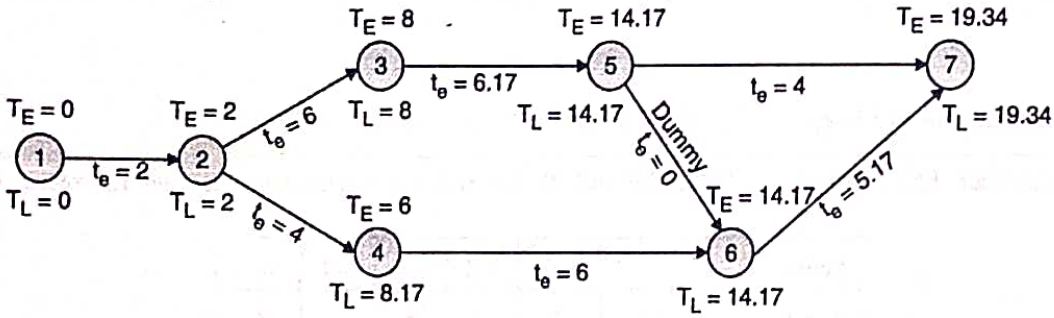


Fig. P. 6.18.9

Now, the most expected time for each activity is calculated using the formula below and written in the network diagram.

$$t_e = \frac{t_o + 4\,t_m + t_p}{6}$$

(i) For activity '1 – 2',

$t_o = 1 \quad t_m = 2 \quad t_p = 3$

$$\therefore\ t_e = \frac{1 + (4 \times 2) + 3}{6} = 2$$

(ii) For activity '2 – 3', $t_o = 3 \quad t_m = 6 \quad t_p = 9$

$$\therefore\ t_e = \frac{3 + (4 \times 6) + 9}{6} = 6$$

(iii) For activity '2 – 4', $t_o = 2 \quad t_m = 4 \quad t_p = 6$

$$\therefore\ t_e = \frac{2 + (4 \times 4) + 6}{6} = 4$$

(iv) For activity '3 – 5', $t_o = 4 \quad t_m = 6 \quad t_p = 9$

$$\therefore\ t_e = \frac{4 + (4 \times 6) + 9}{6} = 6.17$$

(v) For activity '4 – 6', $t_o = 4 \quad t_m = 6 \quad t_p = 8$

$$\therefore\ t_e = \frac{4 + (4 \times 6) + 8}{6} = 6$$

(vi) For activity '5 – 6', $t_o = 0 \quad t_m = 0 \quad t_p = 0$

$\therefore\ t_e = 0$

This is dummy activity as its duration is zero.

(vii) For activity '5 – 7', $t_o = 3 \quad t_m = 4 \quad t_p = 5$

$$\therefore\ t_e = \frac{3 + (4 \times 4) + 5}{6} = 4$$

(viii) For activity '6 – 7', $t_o = 2 \quad t_m = 5 \quad t_p = 9$

$$\therefore\ t_e = \frac{2 + (4 \times 5) + 9}{6} = 5.17$$

After calculation of most expected time (t_e) now, standard deviation (σ) and variance (V) for each activity is calculated and is tabulated in the table form.

(i) For activity '(1 – 2)' $t_p = 3$ $t_o = 1$

Standard deviation (σ) $= \frac{t_p - t_o}{6} = \frac{3-1}{6} = 0.33$

Variance (V) $= \left(\frac{t_p - t_o}{6}\right)^2 = \left(\frac{3-1}{6}\right)^2 = 0.11$

(ii) For activity '2 – 3', $t_o = 3$ $t_p = 9$

Standard deviation (σ) $= \frac{9-3}{6} = 1$

Variance (V) $= \left(\frac{9-3}{6}\right)^2 = 1$

(iii) For activity '2 – 4' $t_o = 2$ $t_p = 6$

∴ Standard deviation (σ) $= \frac{6-2}{6} = 0.67$

Variance (V) $= \left(\frac{6-2}{6}\right)^2 = 0.44$

(iv) For activity 3 – 5, $t_o = 4$ $t_p = 9$

∴ Standard deviation σ $= \frac{9-4}{6} = 0.83$

∴ Variance (V) $= \left(\frac{9-4}{6}\right)^2 = 0.69$

(v) For activity 4 – 6, $t_o = 4$ $t_p = 8$

∴ Standard deviation (σ) $= \frac{8-4}{6} = 0.67$

Variance (V) $= \left(\frac{8-4}{6}\right)^2 = 0.30$

(vi) For activity 5 – 6, $t_o = 0$ $t_p = 0$

∴ Standard deviation = 0, Variance = 0

(vii) For activity 5 – 7, $t_o = 3$ $t_p = 5$

∴ Standard deviation (σ) $= \frac{5-3}{6} = 0.33$

Variance (V) $= \left(\frac{5-3}{6}\right)^2 = 0.11$

(viii) For activity 6 – 7, $t_o = 2$ $t_p = 9$

∴ Standard deviation (σ) $= \frac{9-2}{6} = 1.17$

Variance (V) $= \left(\frac{9-2}{6}\right)^2 = 1.36$

After calculation of standard deviation (σ) and variance (V), Now the event occurance times for each event i.e. T_E and T_L are calculated by forward pass and backward pass as we have seen in CPM network. For this calculation duration of the activity is taken in terms of most expected time (t_e).

(a) From the network diagram,

Critical path = 1 – 2 – 3 – 5 – 6 – 7

Normal project duration = 19.34 weeks.

(b) Variance along the critical path is calculated by adding variance of all critical activities.

∴ Variance along critical path = 0.11 + 1 + 0.83 + 0 + 1.36

= 3.3

Standard deviation along critical path $= \sqrt{\text{Variance}}$

$= \sqrt{3.3} = 1.82$

(c) Probability of completion of project in estimated time

Here, Estimated time T_E = 19.34 weeks

Scheduled time (T_s) = 19.34 weeks

$\therefore Z = \frac{T_s - T_E}{\sigma} = \frac{19.34 - 19.34}{1.82} = 0$

∴ Probability is calculated from the normal distribution chart, for the corresponding Z value.

For Z = 0 probability = 50 %

Ex. 6.18.10 : The data pertains to small project are as follows :

(i) Construct the network diagram.

(ii) Find expected duration and variance,

(iii) What is the probability of completing the project on or before 20 weeks? Take probability 93.3 for Z = 1.5 and probability 94./5 for Z = 1.6, (iv) If the probability of completing project is 0.8, find the expected project completion time. Take Z = – 2.4 for probability 0.8.

Activity	A	B	C	D	E	F	G	H
Immed. Predecessors	None	None	A	B	B	C, D	C, D, E	F
a, (weeks)	1	2	6	1	1	1	1	1
m, (weeks)	2	2	7	2	4	5	2	2
b, (weeks)	3	8	8	3	7	9	3	9

☑ **Soln. :**

Activity	A	B	C	D	E	F	G	H
Immediate predecessor (weeks)	–	–	A	B	B	C, D	C, D, E	F
a (weeks)	1	2	6	1	1	1	1	1
m, (weeks)	2	2	7	2	4	5	2	2
b, (weeks)	3	8	8	3	7	9	3	9

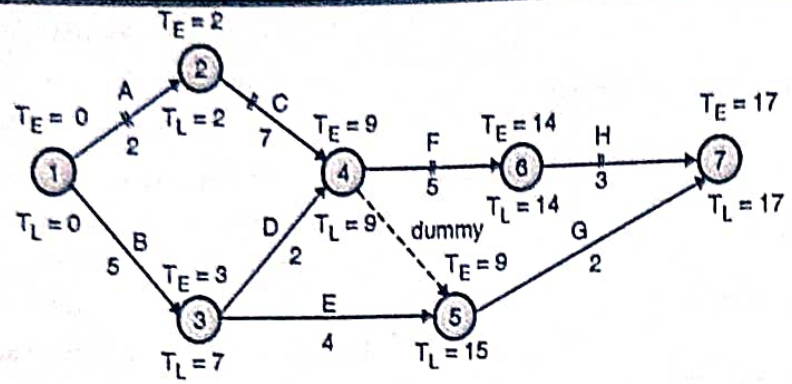


Fig. P. 6.18.10

Activity	a	m	b	t_e	σ	V	Remark
A	1	2	3	2	0.33	0.11	C
B	2	2	8	3	1	1	NC
C	6	7	8	7	0.33	0.11	C
D	1	2	3	2	0.33	0.11	NC
E	1	4	7	4	1	1	NC
F	1	5	9	5	1.33	1.77	C
G	1	2	3	2	0.33	0.11	NC
H	1	2	9	3	1.33	1.77	C

(ii) Expected duration = 17 weeks

Variance $= 0.11 + 0.11 + 1.77 + 1.77 = 3.76$

Standard deviation (σ) $= \sqrt{3.76} = 1.939$

(iii) The scheduled duration of completion for probability 93.3 and z = 1.5 is,

$$z = \frac{T_S - T_E}{\sigma}$$

$$1.5 = \frac{T_S - 17}{1.939}$$

$\therefore T_S = 19.908$ weeks

The scheduled duration of completion for probability 94.5 for z = 1.6

$$z = \frac{T_S - T_E}{\sigma}$$

$$1.6 = \frac{T_S - 17}{1.939}$$

$\therefore T_S = 20.10$ weeks

(iv) If the probability of completing project is 0.8, find, the expected project completion time.

∴ From the normal distribution chart for probability of 0.8

$$z = -2.4 = \frac{T_S - T_E}{\sigma}$$

$$-2.4 = \frac{T_S - 17}{1.939} \qquad \therefore T_S = 12.35 \text{ weeks}$$

Ex. 6.18.11 : PERT calculations yield a project length of 50 weeks with a variance of 16. Within how many weeks would you expect the project to be completed with a probability, of

(i) 95% (ii) 75% (iii) 40%

Given z values for 95% = + 1.65, 75% = + 0.69, 40% = – 0.25

☑ **Soln. :** Expected project duration = 50 weeks

$V = 16 \qquad \therefore \sigma = \sqrt{16} = 4$

(i) For 95 % probability z = + 1.65

$$\therefore z = \frac{T_S - T_E}{\sigma}$$

$$1.65 = \frac{T_S - 50}{4} \qquad \therefore T_s = 56.6 \text{ weeks}$$

(ii) For probability of 75 % and z = + 0.69

$$\therefore 0.69 = \frac{T_S - 50}{4} \qquad \therefore T_s = \mathbf{52.76} \text{ weeks}$$

(iii) For probability of 40 % and z = 0.25

$$-0.25 = \frac{T_S - 50}{4} \qquad \therefore T_s = \mathbf{49} \text{ weeks}$$

Ex. 6.18.12 : Construct network diagram and find out expected project duration. What will be the chances of completing a work in 26 days ? The activities of production schedule are as follows.

Activity	1-2	1-3	2-4	3-4	3-5	4-5	4-6	5-7	6-7
T_0	1	1	4	5	2	6	2	3	1
T_{in}	2	4	6	9	4	7	5	5	2
T_p	3	7	8	13	12	8	114	13	15

Soln. :

Activity	t_0	t_m	t_p	t_e	σ	V	Remark
1 – 2	1	2	3	2	0.33	0.11	NC
1 – 3	1	4	7	4	1	1	C
2 – 4	4	6	8	6	0.66	0.44	NC
3 – 4	5	9	13	9	1.33	1.77	C
3 – 5	2	4	12	5	1.67	2.77	NC
4 – 5	6	7	8	7	0.33	0.11	C
4 – 6	2	5	14	6	2	4	NC
5 – 7	3	5	13	6	1.67	2.77	C
6 – 7	1	2	15	4	2.33	5.44	NC

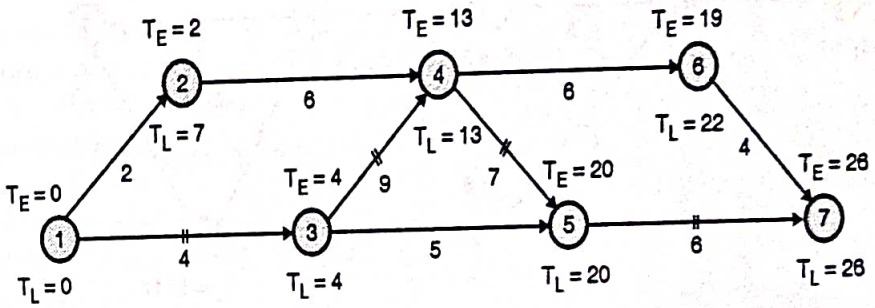


Fig. P. 6.18.12

Variance along critical path $= 1 + 1.77 + 0.11 + 2.77 = 5.65$

Standard deviation $(\sigma) = \sqrt{5.65} = 2.376$

Expected project duration $(T_E) = 26$ days

∴ Probability for completing project in 26 days

$$\therefore z = \frac{T_S - T_E}{\sigma} \quad \text{here } T_S = 26 \quad T_E = 26$$

$$\therefore z = 0$$

For $z = 0$ probability is **50%**

Ex. 6.18.13 : The following tables gives the time estimates of the various activity of a project.

Activity	1-2	2-3	2-4	3-5	4-6	5-6	5-7	6-7
t_o	1	3	2	4	4	0	3	2
t_m	2	6	4	6	6	0	4	5
t_p	3	9	6	9	8	0	5	9

(i) Draw project network and find out total duration.

(ii) Calculate variance along critical path.

Soln. :

Activity	t_o	t_m	t_p	t_e	σ	V	Remark
1 – 2	1	2	3	2	0.33	0.11	C
2 – 3	3	6	9	6	1	1	C
2 – 4	2	4	6	4	0.66	0.44	NC
3 – 5	4	6	9	6.17	0.83	0.69	C
4 – 6	4	6	8	6	0.67	0.44	NC
5 – 6	0	0	0	0	0	0	C
5 – 7	3	4	5	4	0.33	0.11	NC
6 – 7	2	5	9	5.16	1.16	1.36	C

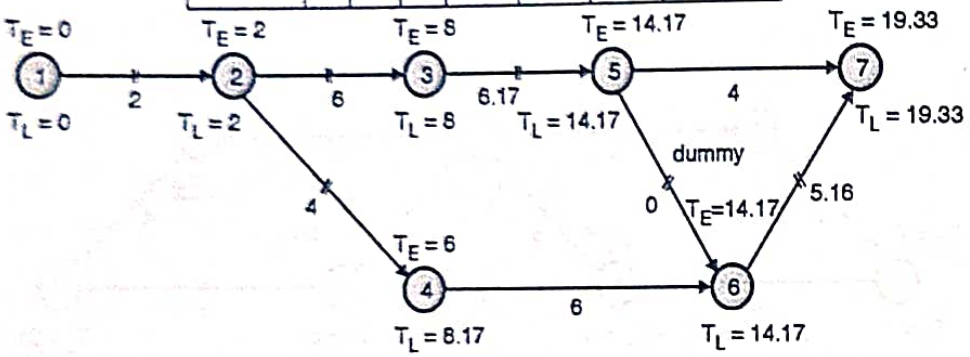


Fig. P. 6.18.13

Total project duration = 19.33 days

Critical path = 1 – 2 – 3 – 5 – 6 – 7

Variance along critical path = 0.11 + 1 + 0.69 + 0 + 1.36

= **3.16**

Syllabus Topic : CPM and PERT

6.19 INTRODUCTION TO CRITICAL PATH METHOD (CPM) AND PROGRAM EVALUATION REVIEW TECHNIQUE (PERT)

GQ. Explain the three time estimates required for each activity under PERT.

- The three most commonly used network techniques are Critical Path Method (CPM), Project Evaluation and Review Technique (PERT) and Precedence Diagramming Method (PDM) also called as Precedence Network Analysis (PNA). These network techniques generate time oriented diagram having activities organized into a logical order. The common features of all three techniques are that :

(i) They make use of the network model for depicting the time plan of the project.

(ii) Apply critical path concept for determining project duration and identifying critical activities

(iii) Employ network analysis techniques for controlling project time objectives.

However, each of these techniques has a distinct model of application.

- CPM (Critical Path Method) and PERT (Programme Evaluation and Review Technique) are network techniques used for planning, scheduling and executing large projects which require coordination and execution of a variety of complex and large number of activities.
- These activities have to be completed within a specified time, cost and meeting the performance standards. CPM and PERT share in common the determination of sequence of activities, also called critical path and are based on the network representation of activities and their scheduling that determines the most critical activities to be controlled so as to meet the completion date of the project.

(i) CPM is best suited for developing the sub-networks of sub-groups and tasks having activities with deterministic single time duration. The sub-networks can then be systematically integrated using PNA technique.

(ii) PERT is useful for project involving uncertainties. In such cases, probabilistic approach of three times is used for assessing the activity duration.

(iii) PDM/PNA is the most commonly used method for time planning.

(iv) GERT is rarely used in project management, hence is not included in this chapter.

CPM

1. This technique was developed in connection with a construction and maintenance project in which duration of each activity was known with certainty.
2. It is suitable for establishing a trade-off for optimum balancing between schedule time and cost of the project.
3. It is used for completion of projects involving activities of repetitive nature.

PERT

1. It assumes a probability distribution for the duration of each activity. Thus completion time estimates for all of the activities are needed.
2. To perform PERT analysis on a project, the emphasis is given on the completion of a task rather than the activities required to be performed to reach to a particular event or task. Thus, it is also called an event -oriented technique.
3. It is used for one time projects involving activities of non-repetitive nature (i.e. activities which may never have been performed before) in which time estimates are uncertain, such as redesigning an assembly line or installing a new information system.
4. It helps in identifying critical areas in a project so that necessary adjustment can be made to meet the scheduled completion date of the project.

6.19.1 Significance of Using CPM/PERT

1. A network diagram helps translation of highly complex projects into a set of simple and logical arranged activities and therefore,

   Help in the clarity of thoughts and actions.

   Help in clear and unambiguous communication developing from top to bottom and vice versa among the people responsible for executing the project.
2. Detailed analysis of network help project in charge to peep in future and anticipate bottleneck areas.
3. Isolation activities which control the project completion and therefore, results in expeditious completion of the project.
4. Helps in the division of the responsibilities and therefore, enhance effective coordination among different departments/agencies involved.
5. Helps in timely allocation of resources to various activities to achieve optimal utilization of resources.

6.19.2 Basic Difference between CPM and PERT

GQ. Differentiate between CPM and PERT.

- Although there are no essential differences between CPM and PERT as both of them share in common the determination of a critical path and are based on the network representation of activities and their scheduling that determines the most critical activities to be controlled so as to meet the completion date of the project.

- However, the critical path method reduces the examination of a large project to three stages :
  (i) Breaking down the project into a set of individual activities and arranging them into a logical sequence.
  (ii) Estimating the duration and resource requirements of each activity, drawing up a schedule and finding which activity controls the completion of the project.
  (iii) Rescheduling money and other resources to improve the schedule.
- There are three ways in which CPM type of network differs from the PERT type :
  (i) CPM is built from activities instead of events
  (ii) Time is related to costs
  (iii) No allowance is made for uncertainty in the estimates of their durations

Factors	CPM	PERT
Application	Deterministic Projects	Projects involving uncertainties.
Model emphasis	Activity Oriented	Event Oriented
Activity duration estimation	Single time	Three times
Time-cost trade-off	Feasible	Not Feasible
Resources optimization	Feasible	Not Feasible
Technique Complexity	Simple	Difficult

Syllabus Topic : Precedence Network Analysis (AON)

6.20 PRECEDENCE NETWORK ANALYSIS (AON)

GQ. Briefly describe Precedence Diagram Method.

- The primary drawback of PERT and CPM was that they were essentially limited to "finish-start" relationships i.e. activity 'B' cannot start until activity 'A' is completed.
- Precedence Network Analysis was developed to overcome the drawbacks of PERT/CPM techniques and its function is to permit a more accurate depiction of relationships among various activities.
- Precedence Network Analysis or Precedence Diagram Method (PDM) is a visual representation technique that depicts the activities involved in a project. PDM is a method of constructing a project schedule network diagram that uses boxes/nodes to represent activities and dependencies.
- Since, the activities are represented by the node, Precedence Diagrams are also called activity-on-node (AON) diagram.

6.20.1 Need for Precedence Diagramming

- Let us understand the need for precedence diagramming through a small activity which requires us to draw network diagram for a pipeline project. Let us assume, for the sake of simplicity, that the project has only three activities, namely excavating, laying of pipes and back filling.
- Now, if we were to draw these three activities in a network we would first have excavation, followed by laying of pipes and finally back filling.
- So basically what we would be doing is start laying of pipes after completing excavation and once laying of pipes is over we would start back filling. But, is this really the way it would be done?
- No, surely not. Assuming that pipeline is to be laid for a few hundred kilometres we are surely not going to wait till all of it is excavated and then start laying of pipes.
- We will be dividing the work into sections. Once, excavation in one section is over we would start with laying of pipes and start excavating in another section. Similarly once laying of pipes is over first section back filling can start there while laying of pipes starts in the next section.
- So, now there is going to be excavation in progress in one section while there is going to be pipe laying on in another and back filling on in the third. Now, the question here is how will you be able represent this on a diagram which has only finish-start relationship.
- Assume that you divide the entire work into sections to depict the activities in the diagram. We would be excavating section 1 and then after finishing that we would be moving to the section 2 and so on.

- Likewise, after excavating in section 1 we would be laying pipes and after finishing the pipe laying there we would move to the section 2 while back filling would start in section1. This means that for merely two sections we would be having 6 activities and for a project running into hundreds of kilometres we would be having hundreds of activities.
- Therefore, for such projects our regular activity-on-arrow and activity-on-node kind of representation will surely not prove useful. This is where the precedence diagram method comes in handy.

6.20.2 History of Precedence Diagramming Method

- The Precedence Diagramming Method or Precedence Network Analysis was developed by Zachary Construction and IBM in the year 1960. This method attempts to get advantage of AOA's event with an AON type of representation.
- Although, this method is the default representation in all popular scheduling software it is not fully accepted by all professional schedulers as the software algorithms are not standardised.

6.20.3 Notations used in Precedence Diagramming Method

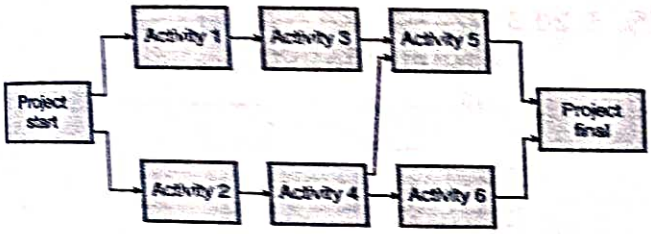


Fig. 6.20.1

- The activity-on-node network diagram is the most common form of a preceding diagram. Each box in a precedence diagram represents a project activity or task. The project activity flows from left to right through the diagram. The arrows denote the interdependencies between the various activities.
- A standard activity box in precedence diagramming method contains the following information :

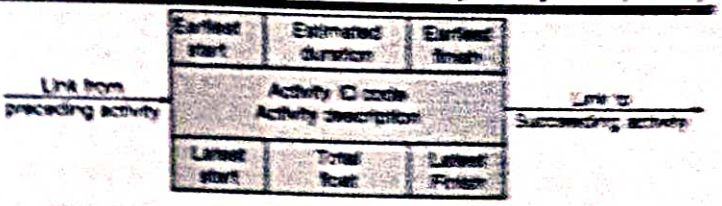


Fig. 6.20.2

6.20.4 Types of Logical Relationships or Dependencies in Precedence Network Analysis

- There are four types of logical relationships or dependencies in Precedence Diagram Method. The four types of relationships exist between two activities. Now, so far we have been only using the finish-start relationship wherein the preceding activity has to finish before the succeeding activity can begin.
- In addition to finish-to-start (FS) relationship, Precedence Network Analysis uses three other types of relationship which are, finish-to-finish (FF), start-to-start (SS) and start-to-finish (SF).
- Let us for the sake of understanding the concept name the activities as A and B. Activity A is a predecessor activity that logically comes before a dependent activity in a schedule. Activity B is a successor activity a dependent activity that logically comes after another activity in a schedule.
- The predecessor activity 'A' is not bound by any condition, it is independent, the successor activity 'B' is the one which is dependent and is watching the predecessor activity 'A' to take trigger to start or finish.
- In addition to this, each relationship can be qualified with lead or lag. Example : If this is a finish-to-start relationship, then we can say it is a finish-to-start relationship with a two day lag.
- This means that the next activity will do finish-to-start with a two day lag. Similarly, we can have a start-to-start with a two day lag which means that the next activity can start two days after the first activity starts. This lag is represented in the relationship itself.

Fig. 6.20.3

I. **Finish-to-Start (FS)** : In this form of relationship the successor activity cannot start until the predecessor activity has finished. Thus, in form the independent activity is not impacted with this relationship constraint. The relationship constraint only impacts the dependent i.e. Activity 'B'. Example: Construction activity cannot commence without approval of design.

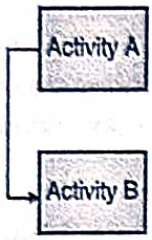


Fig. 6.20.4

II. **Start-to-Start (SS)** : In this form of relationship the successor activity cannot start until the predecessor activity has started. So, if activity 'B' is bound in this relation with activity 'A' then it means that it cannot start until activity A has started. Example: the laying of pipes cannot start no earlier than excavation. Also, there may be a lag of few days in which the laying of pipes can start a day after the excavation begins.

Fig. 6.20.5

III. **Finish-to-Finish (FF)** : In this form of relationship the successor activity cannot finish until the predecessor activity has finished. So, if the activity 'B' is bound in this relation with activity 'A' then it cannot finish until activity 'A' is finished. Activity 'B' needs to finish its own deliverables and continue its activity till the time activity 'A' is not done. Example: Backfilling cannot be finished until laying of pipes is finished. The lights in the room cannot be shut till the students are reading. The commentary of a cricket match cannot be finished till the match is being played.

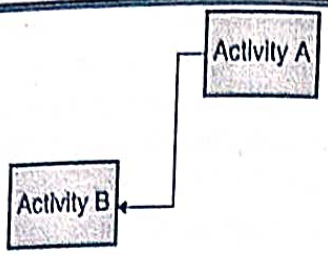


Fig. 6.20.6

IV. **Start-to-Finish (SF)** : In this form of relationship the successor activity cannot finish until the predecessor activity has started. So, if the activity 'B' is bound in this relation with activity 'A' then it cannot finish until activity 'A' has started. Activity 'B' needs to finish its own deliverables and continue its activity till the time independent activity 'A' starts. Sounds confusing doesn't it? In normal cases predecessor activity gets performed before the successor activity but in this case the successor activity is being performed first. However, even though the successor activity is happening first, the predecessor is not at all restricted by successor activity, it is independent, but the successor activity 'B' cannot finish till the predecessor activity 'A' starts. Example : let us assume you have two shifts on the construction site. Now, the first shift can end only when the people of the second shift have come in. So, it is only when the second shift has started can the first shift end.

6.20.5 Precedence Network Representations

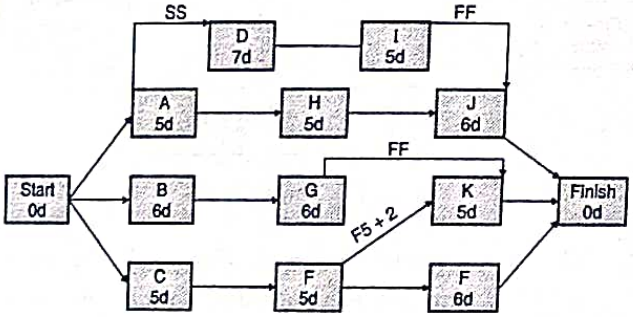


Fig. 6.20.7

- The Fig. 6.20.7 precedence diagram looks similar to an activity-on-node diagram with the arrows representing the relationships.
- Unless specifically stated all the relationships are Finish-to-Start relationships. So, the default relationship is finish-to-start. Activity 'E' to 'K' in the

above diagram is finish-to-start with a lag of two days. The plus sign represents a lag i.e. a delay. So, activity 'K' shall start 2 days after finish of activity 'E'.

Let us study Precedence Network Representation through the example with which we had started the section. We shall be drawing a PDM to represent the following activities.

Activity 1 : Excavation – duration 10 days

Activity 2 : Laying of Pipes – duration 15 days

Activity 3 : Back Filling – duration 5 days

So, we have three activities which seem to be sharing a start-to-start relationship.

Alternative I

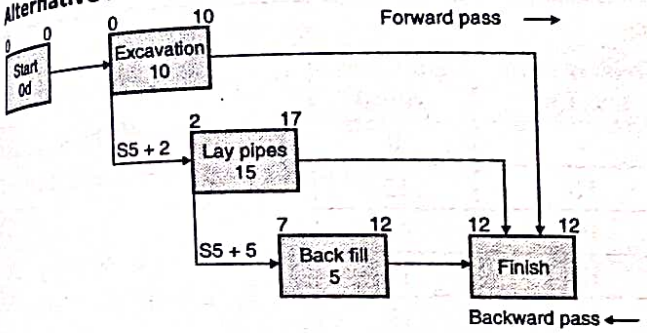


Fig. 6.20.8

Now, if we analyse the PDM we will notice that from excavation to laying of pipes the number of days to finish the activities are 17 and then for backfilling it goes to 12. This cannot be correct and hence our start-to-start relationship cannot be used. We will have to use finish-to-finish relationship.

Alternative II

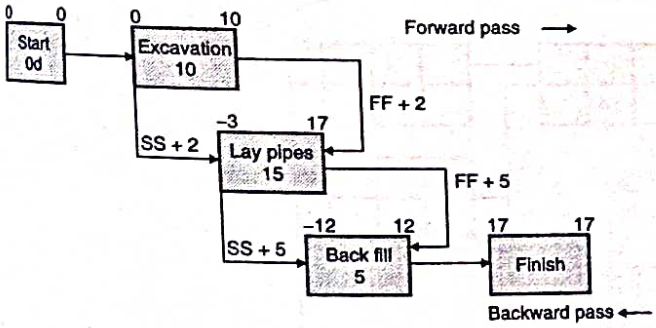


Fig. 6.20.9

The second alternative seems fairly straight forward. You can start laying of pipes two days after excavation and back filling 5 days after starting of laying of pipes taking us to the finish of activities. However, this will not solve the problem as we do not know when the laying of pipe starts. You know that the laying of pipe can start 2 days after starting of excavation but you do not know when the laying will actually start.

Alternative III

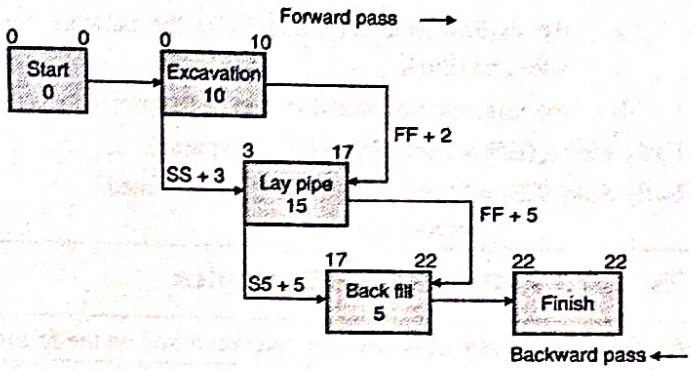


Fig. 6.20.10

The alternative is more like real life projects. We are constraining the laying of pipe activity with the finish-to-finish relationship. So, PDM requires multiple relationships between activities and it can be challenging to represent the relationship.

6.20.6 Network Calculations

Important Terminology

- **Forward Pass** - A calculation starting with the first activity and culminating with the last activity performed to find the early dates and the duration of a specific project.
- **Backward Pass** - A scheduling calculation done to determine the activity late dates. This calculation begins with the last activity and project duration and culminates with the first activity.
- **Float** - The number of days that an event or lag can be delayed or extended without impacting the completion of the project.
- **Early Start (ES)** - The earliest that an activity can start.
- **Early Finish (EF)** - The earliest that an activity can possibly finish.
- **Late start (LS)** - The lates[illegible] activity can start and not impact project comp[illegible]

- **Late Finish (LF)** - The latest that an activity can be completed without impacting the project completion.

Forward and Backward Pass

1. Forward Pass

Forward pass is performed on a network schedule to calculate;

a. the earliest time each activity in the network can start and finish.

b. the minimum overall duration of the project

Early Finish (EF) = Early Start (ES) + Duration

Early Start (ES) = Max Early Finish (EF) of related predecessor activity.

2. Backward Pass

The purpose of performing a backward pass on a network is to calculate the latest time that each activity in the network can start and finish and still maintain the minimum overall duration of the project as calculated by the forward pass.

Late Start (LS) = Late Finish – Duration

Late Finish = Minimum Late Start (LS) of related successor activity.

3. Total Float

Total Float (TF) = Late Start (LS) – Early Start (ES)

= Late Finish (LF) – Early Finish (EF)

6.20.7 Illustrative Examples

Ex. 6.20.1 : Draw a precedence diagram based on the following :

Activity Code	Immediate Predecessor	Relationship and Lag	Duration (days)
A	-	-	8
B	A	FS	12
C	A	FS	16
D	B	FS	14
E`	C	FS	13
F	C, G	SS+10, FS+6	19
G	D	FS	8
H	D, E	FS	14
I	H	FS	12

Also,

(i) Calculate the forward and backward passes

(ii) Identify the Critical Path (iii) Calculate the float

Soln. :

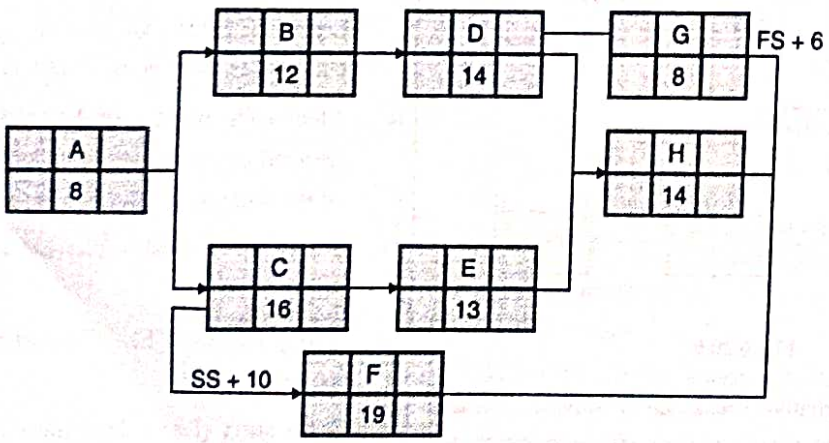


Fig. P. 6.20.1

I. Forward Pass

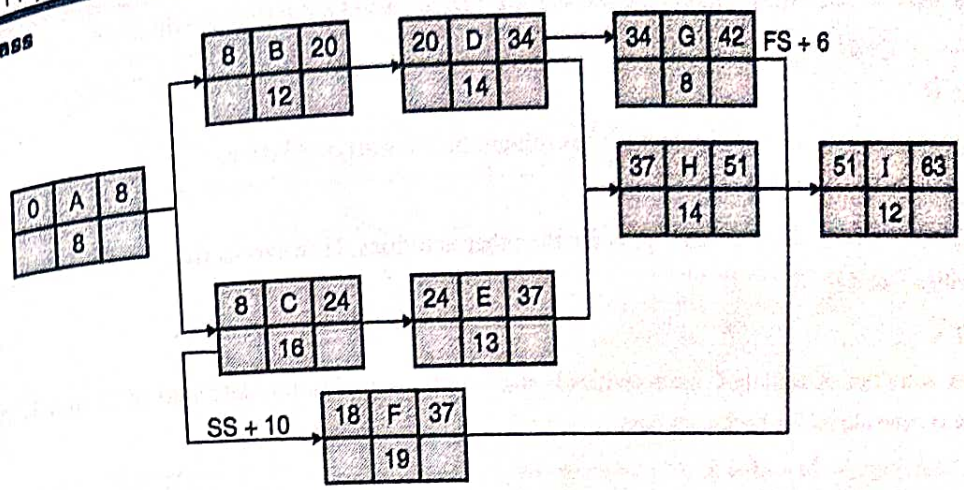


Fig. P. 6.20.1(a)

Forward Pass Calculations

i. **Activity A :** this is the starting activity hence Early Start (ES) = 0.

∴ Early Finish (EF) = Early Start + duration = 0+ 8 =8

ii. **Activity B :** here Early Start (ES)= 8 which is the Early Finish of Activity A.

∴ Early Finish(EF) = Early Start (ES) + Duration = 8 + 12

= 20

iii. **Activity C :** here Early Start (ES) = 20 which the Early Finish of Activity B.

∴Early Finish (EF) = Early Start (ES) + Duration

= 20 + 14 = 34

Similarly the Forward Pass for other Activities can be calculated and have been depicted in the Fig. P. 6.20.1(a). Also, Activity F has two predecessor activities, namely activity C and G, while calculating forward pass of activity F we will have to consider higher duration of predecessor activity.

II. Backward Pass

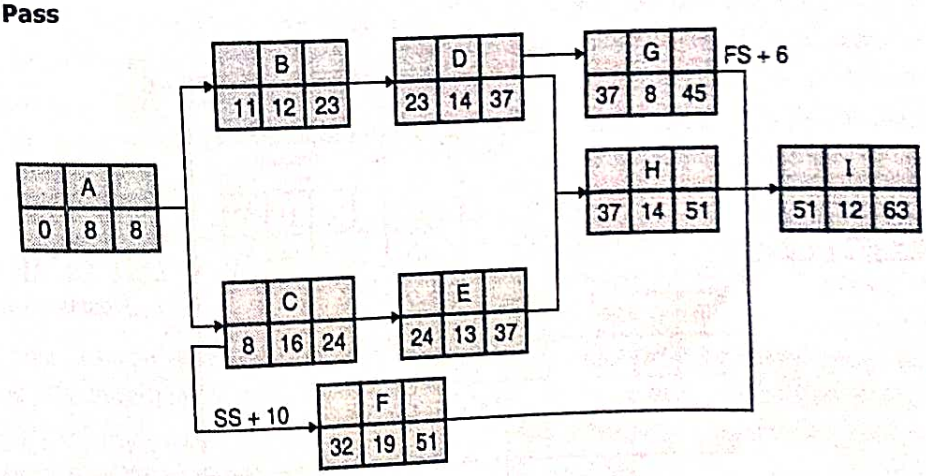


Fig. 6.20.1(b)

Backward Pass Calculations

i. **Activity H**

The minimum late start of activity I which is the related successor is 51 days.

- **Late Finish (LF)** - The latest that an activity can be completed without impacting the project completion.

Forward and Backward Pass

1. Forward Pass

Forward pass is performed on a network schedule to calculate;

a. the earliest time each activity in the network can start and finish.

b. the minimum overall duration of the project

Early Finish (EF) = Early Start (ES) + Duration

Early Start (ES) = Max Early Finish (EF) of related predecessor activity.

2. Backward Pass

The purpose of performing a backward pass on a network is to calculate the latest time that each activity in the network can start and finish and still maintain the minimum overall duration of the project as calculated by the forward pass.

Late Start (LS) = Late Finish – Duration

Late Finish = Minimum Late Start (LS) of related successor activity.

3. Total Float

Total Float (TF) = Late Start (LS) – Early Start (ES)

= Late Finish (LF) – Early Finish (EF)

6.20.7 Illustrative Examples

Ex. 6.20.1 : Draw a precedence diagram based on the following :

Activity Code	Immediate Predecessor	Relationship and Lag	Duration (days)
A	-	-	8
B	A	FS	12
C	A	FS	16
D	B	FS	14
E`	C	FS	13
F	C, G	SS+10, FS+6	19
G	D	FS	8
H	D, E	FS	14
I	H	FS	12

Also,

(i) Calculate the forward and backward passes

(ii) Identify the Critical Path (iii) Calculate the float

Soln. :

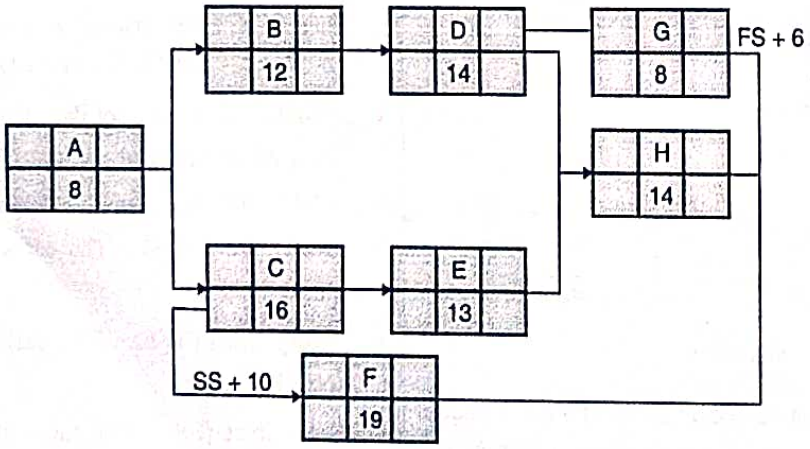


Fig. P. 6.20.1

I. Forward Pass

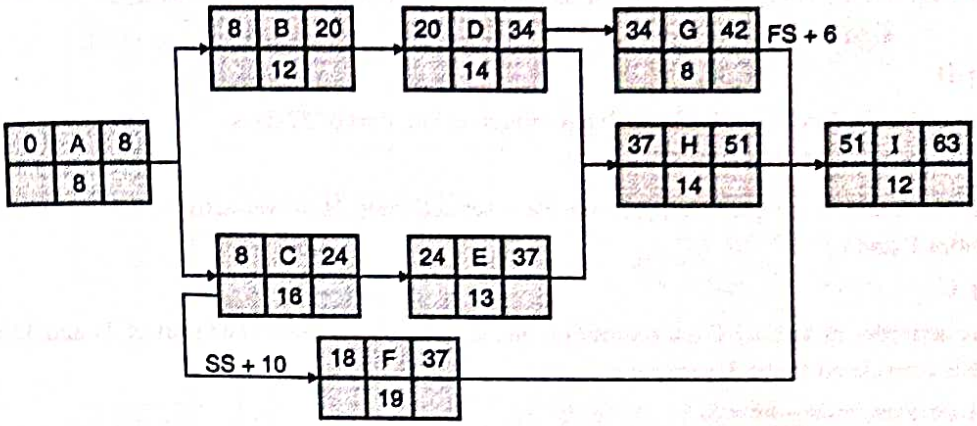


Fig. P. 6.20.1(a)

Forward Pass Calculations

i. **Activity A :** this is the starting activity hence Early Start (ES) = 0.

∴ Early Finish (EF) = Early Start + duration = 0+ 8 =8

ii. **Activity B :** here Early Start (ES)= 8 which is the Early Finish of Activity A.

∴ Early Finish(EF) = Early Start (ES) + Duration = 8 + 12

= 20

iii. **Activity C :** here Early Start (ES) = 20 which the Early Finish of Activity B.

∴Early Finish (EF) = Early Start (ES) + Duration

= 20 + 14 = 34

Similarly the Forward Pass for other Activities can be calculated and have been depicted in the Fig. P. 6.20.1(a). Also, Activity F has two predecessor activities, namely activity C and G, while calculating forward pass of activity F we will have to consider higher duration of predecessor activity.

II. Backward Pass

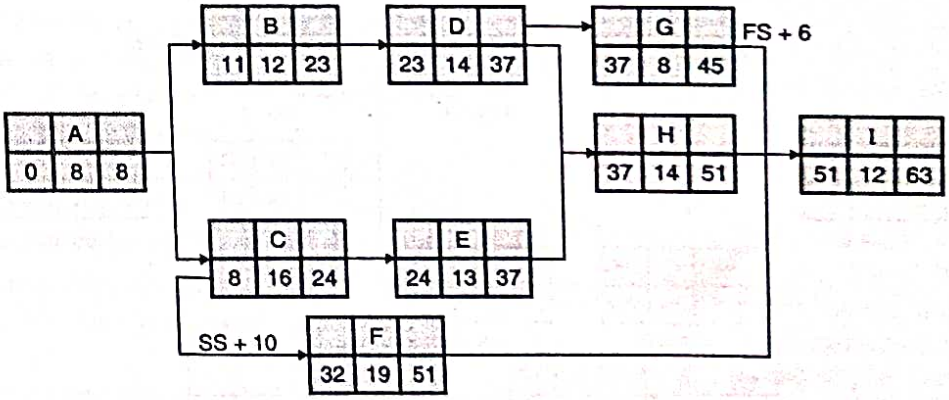


Fig. 6.20.1(b)

Backward Pass Calculations

i. **Activity H**

The minimum late start of activity I which is the related successor is 51 days.

∴ Late Start = Late Finish (minimum late start of related successor activity) – duration

= 51 – 14 = 37

ii. Activity D

The successor activity to activity D is H which has minimum late start of 37 days.

∴ **Late Start = 37 – 14 = 23**

We can similarly calculate the backward pass for the other activities. However activity C has two successor activities, namely activities E and F.

iii. Activity C

The successor activities of activity C are activities E and who have a minimum late start of 24 and 32 respectively. The smaller value is considered for backward pass.

∴Late Start = 24 – 16 = 8.

III. Critical Path

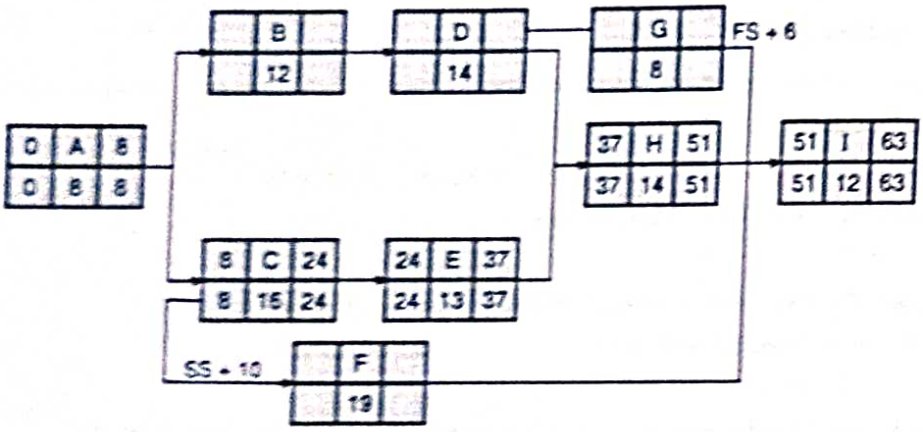


Fig. P. 6.20.1(c)

Critical Path = A-C-E-H-I and the project duration is 63 days.

IV. Float

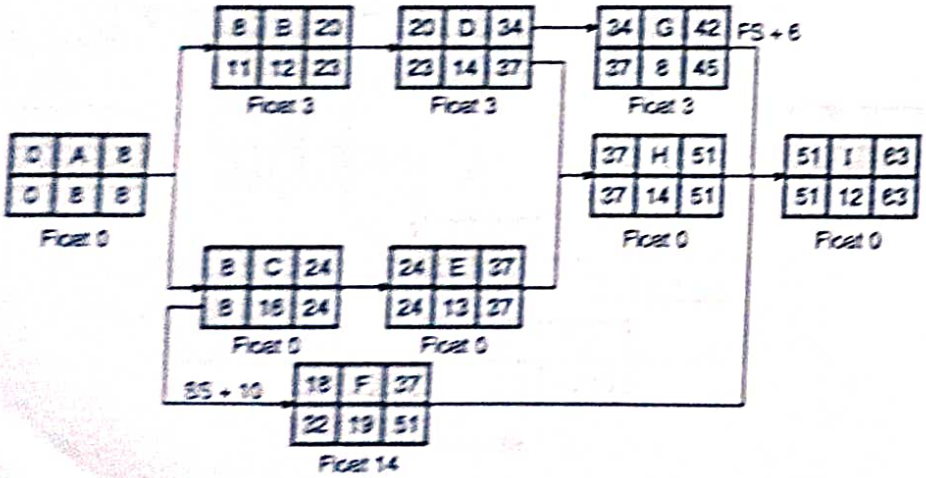


Fig. P. 6.20.1(d)

Draw a precedence diagram based on the following activities :

Activity	Immediate Predecessor	Relationship and Lag	Duration
A	-	-	8
B	A	SS + 1, FF + 1	8
C	B	FS	9
D	-	-	7
E	B, D	SS + 3, FF + 4	9

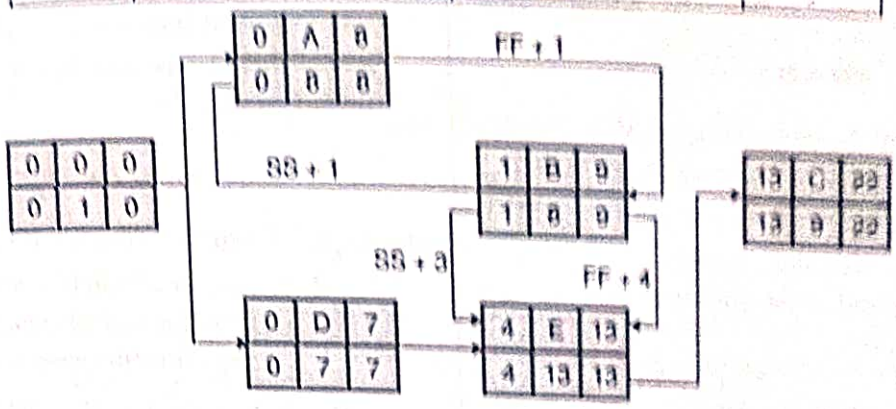


Fig. P. 6.20.2

PROGRAM EVALUATION REVIEW TECHNIQUE (PERT)

"It is an event oriented technique." Explain.

was developed in the context where many ies associated with the project had never been ted previously (e.g. R&D projects) PERT was ped to handle projects where time duration for ctivity is no longer just a single time estimate, decision-makers best guess) but is a random le which is characterized by some probability bution - usually a β (beta)-distribution.

timate the parameters of the β -distribution (the and variance), the PERT model requires three estimates for each activity.

these times a single value is estimated for future ideration. The three-time estimates required are as r.

Optimistic time (to or a) : The shortest possible time (duration) in which an activity, can be performed assuming that everything goes well.

Pessimistic time (tp or b) : The longest possible time required to perform an activity under extremely bad conditions. However, such conditions do not include acts of God like earthquakes, flood, etc.

(iii) Most likely time (t_m or m) : The time that would occur most often to complete an activity, if the activity was repeated under exactly same conditions many times. Obviously, it is the completion time that would occur most frequently (i.e. model value).

- The β-distribution is not necessarily symmetric, the degree of skewness depends on the location of t_m to t_o and t_p. Thus the range specified by the optimistic time (t_o) and pessimistic time (t_p) estimates is assumed to enclose every possible estimate of the duration of the activity.
- The most likely time (t_m) estimate may not coincide with the midpoint $(t_o + t_p)/2$ and may occur to its left or to its right as shown in Fig. 6.21.1. Because of these properties, it is justified to assume that the duration of each activity may follow Beta (β)-distribution with its unimodal point occurring at t_m and its end points at t_o and t_p.

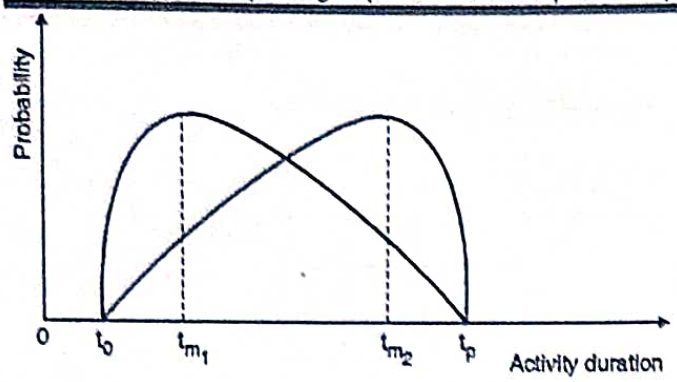


Fig. 6.21.1

- In Beta-distribution the midpoint $(t_o + t_p)/2$ is assumed to weigh half as much as the most likely point (t_m). Thus the expected or mean (t_e or μ) time of an activity, that is also the weighted average of three time estimates, is computed as the arithmetic mean of $(t_o + t_p)/2$ and $2\,t_m$. That is

$$\text{Expected time of an activity } (t_e) = \frac{(t_o + t_p)/2 + 2\,t_m}{3}$$

$$= \frac{t_o + 4\,t_m + t_p}{6}$$

- With uncertain activity time, variance can be used to describe the dispersion (variation) in the activity time values.
- The calculations are based on an analogy to the normal distribution where 99 per cent of the area under normal curve is within $\pm 3\sigma$ from the mean or fall within the range approximately 6 standard deviation in length, therefore the interval (t_o, t_p) or range $(t_p - t_o)$ is assumed to enclose about 6 standard deviations of a symmetric distribution.
- Thus, if σ_i denotes the standard deviation of the duration of activity i, then

$$6\sigma_i \quad t_p - t_o \qquad \text{or} \sigma_I = \frac{t_p - t_o}{6}$$

$$\text{Variance of activity time,} \quad \sigma_i^2 = \left[\frac{1}{6}(t_p - t_o)\right]^2$$

- If we assume that the durations of the activities are independent random variables, then the variance of the total critical path's duration is the sum of the variances on the critical path. Suppose σ_c is the standard deviation of the critical path. Then

$$\sigma_c^2 = \sum \sigma_i^2 \text{ and } \sigma_c = \sqrt{\sum \sigma_i^2}$$

Estimation of Project Completion Time

- Since we expect variation in the activity duration, therefore the chance of completing the project in a desired time and the duration necessary for obtaining any desired probability of actually meeting the scheduled time can be calculated as discussed below :
- The probability distribution of times for completing an event can be approximated by the normal distribution due to central limit theorem. Thus, the probability of completing the project in the schedule (or desired) time, T_s is given by

$$\text{Prob}\left(Z = \frac{T_s - T_e}{\sigma_p}\right)$$

where, T_e = expected completion time of the project

Z = number of standard deviations the scheduled times (or desired completion times) lies away from the mean or expected date.

$\sigma_i^2 = \sigma_1^2 + \sigma_2^2 + \ldots \sigma_n^2$ is the sum of variances of critical activities.

The desired completion time of the project can be calculated as:

$$T_s = Z\sigma + T_e$$

- Where, value of Z corresponds to the probability of project completion time. The computation of T_s enables a decision-maker to make certain commitments knowing the degree of risk assumed.
- The expected completion time of the project is obtained by adding the expected time of each activity lying on the critical path. Since it is assumed that the two activities are independent, therefore variance of the critical path can be known by adding variance of critical activities. We will understand PERT with the help of following example.

Ex. 6.21.1 : A small project is composed of 7 activities whose time estimates are listed in the table below. Activities are identified by their beginning (i) and ending (j) node numbers.

Activity	Estimated Duration (weeks)		
(i – j)	Optimistic	Most Likely	Pessimistic
1 – 2	1	1	7
1 – 3	1	4	7
1 – 4	2	2	8

Activity	Estimated Duration (weeks)		
2 – 5	1	1	1
3 – 5	2	5	14
4 – 6	2	5	8
5 – 6	3	6	15

(a) Draw the network diagram of activities in the project.

(b) Find the expected duration and variance for each activity. What is the expected project length ?

(c) Calculate the variance and standard deviation of the project length. What is probability that the project will be completed ?

(i) at least 4 weeks later than expected time.

(ii) no more than 4 weeks later than expected time.

(d) If the project due date is 19 weeks, what is the probability of not meeting the due date.

Given:	Z	0.50	0.67	1.00	1.33	2.00
	Prob	0.3085	0.2514	0.1587	0.0918	0.0228

☑ **Soln. :**

The network diagram of activities in the project is shown in Fig. P. 6.21.1. The earliest and latest expected time for each event is calculated by considering the expected time of each activity as shown in Table P. 6.21.1.

Table P. 6.21.1

Activity	t_o	t_m	t_p	$t_e = \frac{(t_o + 4t_m + t_p)}{6}$	$\sigma^2 = \left[\frac{(t_p - t_o)}{6}\right]^2$
1 – 2	1	1	7	2	1
1 – 3	1	4	7	4	1
1 – 4	2	2	8	3	1
2 – 5	1	1	1	1	0
3 – 5	2	5	14	6	4
4 – 6	2	5	8	5	1
5 – 6	3	6	15	7	4

The E-values and L – values bases on expected time (t_e) of each activity are shown in Fig. P. 6.21.1.

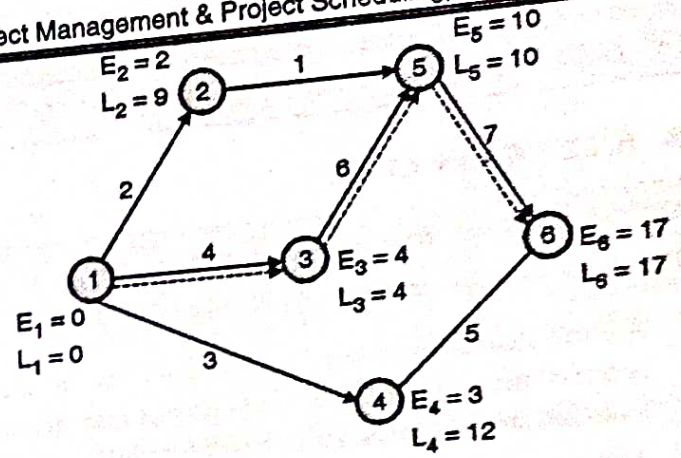


Fig. P. 6.21.1 : Network diagram

(a) Critical path is : 1 – 3 – 5 – 6.

(b) The expected duration and variance for each activity is shown in Table P. 6.21.1. The expected project length is the sum of the duration of each critical activity. That is,

Expected project length = 1 – 3 – 5 – 6 = 4 + 6 + 7 = 17 weeks.

(c) Variance of the project length is the sum of the variances of each critical activity, that is

Variance of project length = 1 – 3 – 5 – 6 = 1 + 4 + 4 = 9 weeks.

Therefore, Standard deviation, $\sigma_p = \sqrt{9} = 3$

(i) Probability that the project will be completed at least 4 weeks earlier (i.e. 13 weeks) than expected time of 17 weeks is given by

$$\text{Prob.}\left\{Z \le \frac{T_s - T_e}{\sigma_p} = \frac{13-17}{3}\right\}$$

$$= \text{Prob.}\{Z \le -1.33\}$$

$$= 0.5 - 0.4082 = 0.0918$$

Thus the probability of completing the project in less than 13 days is 9.18 per cent.

(ii) Probability that the project will be completed in 4 weeks later (i.e. 21 weeks) than expected time of 17 weeks in given by

$$P\left\{Z \ge \frac{21-17}{3}\right\} = P\{Z \ge 1.33\}$$

$$= 0.5 + 0.4082 = 0.9082.$$

6.22 NETWORK PLANNING MODELS

6.22.1 Gantt Chart

GQ. What is Gantt chart? How is it created?

GQ. State the advantages and limitations of Gantt chart.

- A Gantt chart or more popularly known as the bar chart is one of the most commonly used project management tools and are used in tracking project schedules.
- Although, the Gantt charts were originally devised by Karol Adamiecki in the 1890's, they were later on modified by Henri Gantt, an American engineer and management consultant, and hence they are widely known in western countries as Gantt charts.
- Gantt chart provide a standard format for displaying project schedule information by listing project activities and their corresponding start and finish dates in a calendar format. These charts are also referred to as bar charts as activity start and finish dates are shown by horizontal bars.
- The primary reason for the popularity of the Gantt charts is because they are easy to plot and simple to understand. The Gantt adopts a very simple process of showing activities (tasks or events) displayed against time.
- A Gantt chart is a graphical representation of a project plan on a time scale. The X axis of the chart represents time i.e. project duration while the Y axis represents activities in the form of rectangular bars.
- The position and length of the bar reflects the start date, duration of the activity and the end date of the activity. Each horizontal bar represents one specific activity to be performed and the beginning and end of each bar represents the commencement and completion of the activity.
- The Gantt chart enables the user to :
  - Identify the various activities in the project
  - Identify the beginning and end of each activity
  - Identify the duration of each activity
  - Identify the overlapping of activities
  - Identify the start and end date of activities
- Therefore, in simple terms a Gantt chart enables the user to understand what has to be done i.e. the activities to be performed and when it is to be done i.e. the schedule of the activities.

Creating Gantt chart

- Gantt charts are widely used in business to describe and monitor all kinds of projects according to the rules of project management.
- The basic requirement of a Gantt chart is a detailed project plan. A project plan is a series of interdependent tasks that need to be performed in a particular order, for example you create a WBS before developing a project scope statement.
- Project plans have a specific start date, corresponding to the start of the first task, and a specific end date, corresponding to the end of the last task. The project plan can be created by using the Work Breakdown Structure, which splits tasks into sub tasks and creates a task hierarchy.
- Once the project plan is complete the project manager can create the Gantt chart. The X axis will have specific units of time designated such as days, weeks, etc. Activities on Y Axis are listed in order of the earliest start on the project.
- Activities are first listed out vertically in a logical sequence of execution. Tentative duration of each activity is estimated based on the availability of resources.
- Activities on the Gantt chart should coincide with the activities on the WBS which in turn should coincide with the activity list and milestone list.
- The scheduling starts from the project start date and tasks start as soon as possible which means that the project finishes at the earliest possible date. However, while scheduling the project manager has to keep in mind the number of working days in the week and the number of working hours in the day.
- The tasks are named and duration and resources are assigned to each task. The project manager will need to specify the dependencies between the project tasks.
- Any change in the timing of a task affects all the tasks that depend on it. If a task runs ahead of schedule, the Gantt should recalculate the dates of all the tasks that depend on it in order to take advantage of the time gained.

- Conversely, if a task is delayed, all the tasks that depend on it are rescheduled, which may or may not impact the overall end date of the project.
- The Gantt chart for IT project management contains milestones, summary tasks, individual task duration and arrows showing task dependencies.

Example

Activity No	Activity Description	Duration in Days	Remarks
1	Task1	5 days	
2	Task2	2 days	
3	Task3	6 days	Starts with activity 2
4	Task4	4 days	Starts 2 days after activity 3 or after completion of activity 2
5	Task5	4 days	Starts after completion of activity 4
6	Task6	12 days	Starts after completion of activity 5
7	Task7	3 days	Starts after completion of activity 6
8	Task8	2 days	Starts after completion of activity 5
9	Task9	5 days	Starts after completion of activity 6
10	Task10	12 days	Starts 3 days after starting of activity 9
11	Task11	4 days	Starts with activity 10
12	Task12	12 days	Starts 3 days after starting of activity 11
13	Task13	6 days	Starts 1 days after starting of activity 12
14	Task14	5 days	Starts after completion of activity 13
15	Task15	4 days	Starts after completion of activity 14

- Total duration without considering overlapping of activities = 86 days
- Actual duration after considering the overlapping of activities = 86 – 37 = 49 days

Using Gantt Charts to compare actual dates with planned dates

- The Gantt chart can be used to evaluate progress on a project by showing actual project information. The chart can be designed to compare the planned and actual project schedule information.
- The planned schedule dates for activities are called the baseline dates and the planned schedule is called baseline schedule.
- The Gantt chart includes columns depicting the actual and finish dates for each activity and columns depicting the baseline start and finish for each activity. A one on one comparison enables project stakeholders to compare the actual progress with the planned.

☞ Advantages of Gantt chart

- Gantt chart provides vital information in simple graphical and tabular format
- Gantt chart is easy to understand by everyone in all levels of hierarchy starting from top management to workers working on the project.
- Gantt chart is a tool in which project planning, scheduling and controlling all these activities can be performed using a single chart.
- Using Gantt chart manager can easily find out the work load on machines and current status of the work progress.
- Gantt chart is flexible to apply with some common changes.
- Project monitoring and evaluating the project is easy by using Gantt chart.

☞ Limitations of Gantt chart

- Gantt chart does not provide the details regarding the operations progress.

- Interrelationship between various operations is not clear in Gantt chart
- Gantt chart is static in nature and for changes in schedule and plans, the entire old Gantt chart need to be replaced by a new one.
- Gantt chart does not provide information related to manpower, machinery performance and breakdowns.
- CPM (Critical Path Method) and PERT (Program Evaluation Review Technique) are project management techniques, which have been created to overcome the limitations of Gantt.

6.22.2 Milestone Chart

GQ. Write short note on : Milestone Chart.

- Milestones, as we have seen, are important part of the project schedule especially for large projects. Project managers like to focus on meeting milestones so as to emphasize the project accomplishment.
- Milestone charts are similar to Gantt Charts only that they highlight the scheduled completion of major deliverables and key external interfaces.
- The Gantt chart can also be made to incorporate the milestones within the project. Milestones are marked by the completion of various project activities/tasks and hence are created by entering them as tasks with zero duration.

6.23 TRACKING THE SCHEDULE

- If it has been properly developed, the project schedule becomes a road map that defines the tasks and milestones to be tracked and controlled as the project proceeds. Tracking can be accomplished in a number of different ways :
  - Conducting periodic project status meetings in which each team member reports progress and problems
  - Evaluating the results of all reviews conducted throughout the software engineering process
  - Determining whether formal project milestones have been accomplished by the scheduled date
  - Comparing the actual start date to the planned start date for each project task listed in the resource table
  - Meeting informally with practitioners to obtain their subjective assessment of progress to date and problems on the horizon
  - Using earned value analysis to assess progress quantitatively In reality, all of these tracking techniques are used by experienced project managers.
- Control is employed by a software project manager to administer project resources, cope with problems, and direct project staff.
- If things are going well (i.e., the project is on schedule and within budget, reviews indicate that real progress is being made and milestones are being reached), control is light. But when problems occur, you must exercise control to reconcile them as quickly as possible.
- After a problem has been diagnosed, additional resources may be focused on the problem area: staff may be redeployed or the project schedule can be redefined.
- When faced with severe deadline pressure, experienced project managers sometimes use a project scheduling and control technique called *time-boxing*.
- The time-boxing strategy recognizes that the complete product may not be deliverable by the predefined deadline. Therefore, an incremental software paradigm is chosen, and a schedule is derived for each incremental delivery.
- The tasks associated with each increment are then time-boxed. This means that the schedule for each task is adjusted by working backward from the delivery date for the increment.
- A "box" is put around each task. When a task hits the boundary of its time box (plus or minus 10 percent), work stops and the next task begins. The initial reaction to the time-boxing approach is often negative: "If the work isn't finished, how can we proceed?" The answer lies in the way work is accomplished. By the time the time-box boundary is encountered, it is likely that 90 percent of the task has been completed.[9] The remaining 10 percent, although important, can (1) be delayed until the next increment or (2) be completed later if required. Rather than becoming "stuck" on a task, the project proceeds toward the delivery date.

6.24 EARNED VALUE ANALYSIS

- 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.
- Stated even more simply, earned value is a measure of progress. It enables you to assess the "percent of completeness" of a project using quantitative analysis rather than rely on a gut feeling
- 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, $BCWS_i$ 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. 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. "the distinction between the BCWS and the BCWP is that the former represents the budget of the activities that were planned to be completed and the latter represents the budget of the activities that actually were completed." Given values for BCWS, BAC, and BCWP, important progress indicators can be computed:

   Schedule performance index, SPI

   SPI = BCWP / BCWS

   Schedule variance, SV = BCWP - BCWS

- SPI is an indication of the efficiency with which the project is utilizing scheduled resources. An SPI value close to 1.0 indicates efficient execution of the project schedule. SV is simply an absolute indication of variance from the planned schedule.

  Percent scheduled for completion = BCWS/BAC

  provides an indication of the percentage of work that should have been completed by time *t*.

  Percent complete = BCWP/BAC

  provides a quantitative indication of the percent of completeness of the project at a given point in time *t*.

It is also possible to compute the *actual cost of work performed* (ACWP). The value for ACWP is the sum of the effort actually expended on work tasks that have been completed by a point in time on the project schedule. It is then possible to compute

Cost performance index, CPI=BCWP/ACWP

Cost variance, CV = BCWP-ACWP

A CPI value close to 1.0 provides a strong indication that the project is within its defined budget. CV is an absolute indication of cost savings (against planned costs) or shortfall at a particular stage of a project.

Like over-the-horizon radar, earned value analysis illuminates scheduling difficulties before they might otherwise be apparent. This enables you to take corrective action before a project crisis develops.

Chapter Ends...