SOFTWARE ENGINEERING

DEPT. OF COMPUTER SCIENCE AND ENGINEERING

PREPARED BY:-PRIYANKA PANDA

5TH SEMESTER

[8 marks long question]

1. Explain the classical life cycle model?[2018][2017][2016]

Ans:-Classical life cycle model:

In the Classical life cycle model (also known as the waterfall model), the development of software proceeds linearly and sequentially from requirement analysis to design, coding, testing, integration, implementation, and maintenance. Thus, this model is also known as the linear sequential model.

This model is simple to understand and represents processes which are easy to manage and measure. The waterfall model comprises different phases and each phase has its distinct goal. After the completion of one phase, the development of software moves to the next phase. Each phase modifies the intermediate product to develop a new product as an output. The new product becomes the input of the next process. Table lists the inputs and outputs of each phase of waterfall model.This model consist basically four phases:

(A)Analysis Phase

(b)Design Phase

(c)Coding Phase

(d)Testing phase

(A)Requirements analysis: This phase focuses on the requirements of the software to be developed. It determines the processes that are to be incorporated during the development of the software. To specify the requirements, users' specifications should be clearly understood and their requirements be analyzed. This phase involves interaction between the users and the software engineers and produces a document known as Software Requirements Specification(SRS).

(B)Design: This phase determines the detailed process of developing the software after the requirements have been analyzed. It utilizes software requirements defined by the user and translates them into software representation. In this phase, the emphasis is on finding solutions to the problems defined in the requirements analysis phase. The software engineer is mainly concerned with the data structure, algorithmic detail and interface representations.

(C)Coding: This phase emphasizes translation of design into a programming language using the coding style and guidelines. The programs created should be

easy to read and understand. All the programs written are documented according to the specification.

(D)Testing: This phase ensures that the software is developed as per the user's requirements. Testing is done to check that the software is running efficiently and with minimum errors. It focuses on the internal logic and external functions of the software and ensures that all the statements have been exercised (tested). Note that testing is a multistage activity, which emphasizes verification and validation of the software.

2. Explain the different quality factors with a software product? [2015][2016]

Ans:-**Flexibility and Extensibility**:
Flexibility is the ability of software to add/modify/remove functionality without damaging current system. Extensibility is the ability of software to add functionality without damaging system, so it may be thought as a subset of flexibility. Those functionality changes may occur acoording to changing requirements, or an obligation if development process is one of the iterative methods. Change is inevitable in software development and so, this is one of the most important properties of quality software

Maintainability and Readability
Maintainability is a little similar with flexibility but it focuses on modifications about error corrections and minor function modifications, not major functional extensibilities. It can be supported with useful interface definitions, documentations and also self-documenting code and/or code documentation. The more correct and useful documentation exists, the more maintainability can be performed.

Performance and Efficiency
Performance is mostly about response time of the software. This response time should be in acceptable intervals (e.g. max. a few seconds), and should not increase if transaction count increases. And also, resources are expensive. Efficiency must be supported with resource utilization. As an exaggerated example, ability of performing a simple function only by using a 32 processor machine or 1 TB disk space is not acceptable. Optimal source/performance ratio must be aimed.

Scalability
A scalable system responds user actions in an acceptable amount of time, even if load increases. Of course more hardware may be added for handling increasing user transaction, but the architecture should not change while doing this. This is called vertical scalability. Ability of running on multiple, increasing count of machines is multiple processing. If the software can perform that type of processing, this is called horizontal scalability. A preffered scalable system should suit both of these methods.
Availability, Robustness, Fault Tolerance and Reliability:
A robust software should not lose its availabilty even in most failure states. Even if some components are broken down, it may continue running. Besides, even if whole application crashes, it may recover itself using backup hardware and data with fault tolerance approaches. There should always be B and even C, D ..plans. Reliability also

stands for the integrity and consistency of the software even under high load conditions. So it is relevant with availability and scalability. An unreliable system is also unscalable.

Usability and Accessability

User interfaces are the only visible parts of software according to the viewpoint of user. So, simplicity, taking less time to complete a job, fast learnability etc. are very important in this case. The most well known principle for this property is KISS (Keep It Simple Stupid). Simple is always the best. A usable software should also support different accessibility types of control for people with disabilities.

Platform Compatibility and Portability

A quality software should run on as much various platforms as it can. So, more people can make use of it. In different contexts we may mention different platforms, this may be OS platforms, browser types etc. And portability is about adapting software that can run on different platforms, for being more platform compatible. In this sense, portability is also related with flexibility

Testability and Managability

Quality software requires quality testing. Source code should be tested with the most coverage and with the most efficient testing methods. This can be performed by using encapsulation, interfaces, patterns, low coupling etc. techniques correctly. Besides testability, a qualified software should be manageable after deployment. It may be monitored for e.g. performance or data usage status, or may enable developer to configure system easily. Creating a successful logging system is another very important issue about managability.

Security

Security is a very important issue on software development, especially for web or mobile based ones which may have millions of users with the ability of remote accessing to system. You should construct a security policy and apply it correctly by leaving no entry points. This may include authorization and authentication techniques, network attack protections, data encryption and so on. all possible types of security leaks should be considered, otherwise one day only one attack may crash your whole applicaion and whole company.

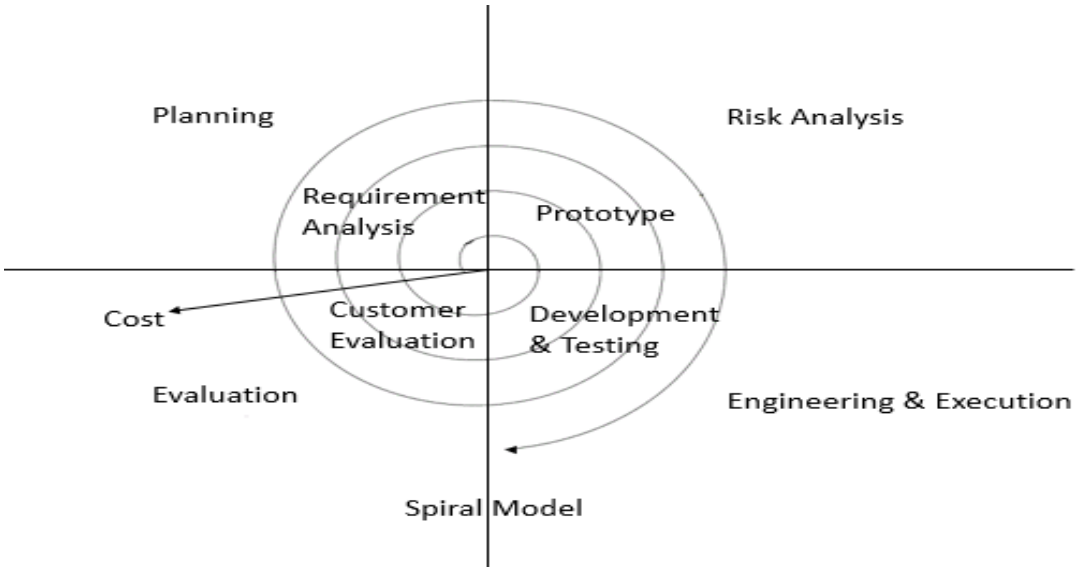
Functionality and Correctness

Functionality (or correctness) is the conformity of the software with actual requirements and specifications. In fact this is the precendition attribute of an application, and maybe not a quality factor but we wanted to point that as the last quality factor, for taking attention: Quality factors are not meaningful when we are talking about unfunctional software. First, perform desired functionality and produce correct software, then apply quality factors on it. If you can perform both paralelly, it is the best.

3. Explain Siral Model? [2015][2017]

Ans:-Spiral Model

The spiral model also known as the spiral life cycle model is a systemsdevelopment life cycle model used in information technology. This model of development combines the features of the prototyping model, the waterfall model and other models. The diagrammatic representation of this model appears like a spiral with many loops.



Exact number of phases through which the product is developed in this model is not fixed. The number of phases varies from one project to another. Each phase in this model is split into four sectors or quadrants:

- **Planning:** Identifies the objectives of the phase and the alternative solutions possible for the phase and constraints.
- **Risk analysis:** Analyze alternatives and attempts to identify and resolve the risks involved.
- **Development:** Product development and testing product.
- **Assessment:** Customer evaluation.

During the first phase planning is performed, risks are analyzed, prototypes are built and customers evaluate the prototype. During the second phase a second prototype is evolved by a fourfold procedure: evaluating the first prototype in terms of its strengths, weaknesses and risks, defining the requirements of the second prototype, constructing and testing the second prototype. The existing prototype is evaluated in the same manner as was the previous prototype and if necessary another prototype is developed. After several iterations along the spiral, all risks are resolved and the software is ready for development. At this point, a waterfall model of software development is adopted.
The radius of the spiral at any point represents the cost incurred in the project till then and the angular dimension represents the progress, made in the current phase.
In the spiral model of development, the project team must decide how exactly to structure the project into phases. The most distinguishing feature of this model is its ability to handle risks. The spiral model uses prototyping as a risk reduction mechanism and also retains the systematic step-wise approach of the waterfall model.

4. What is prototyping? Explain. [2015][2016]

Ans.)The prototyping paradigm can be either close-ended or open-ended. The close-ended approach is called throwaway prototyping and an open-ended approach called evolutionary prototyping.

Prototyping Approach

Throwaway prototyping: Prototype only used as a demonstration of product requirements.
Evolutionary prototyping uses the prototype as the first part of an analysis activity that will be continued into design and construction.
The customer must interact with the prototype, it is essential that:
a) Customer resources must be committed to evaluation and refinement of the prototype.
b) Customer must be capable of making requirements decisions in a timely manner.

Prototyping Tools and Methods

Three generic classes of methods and tools are:
• Fourth generation techniques: Fourth generation techniques (4GT) tools allow software engineer to generate executable code quickly.
• Reusable software components: Assembling prototype from a set of existing software components.
• Formal specification and prototyping environments can interactively create executable programs from software specification models.

5. Explain the classical life cycle model? [2016][2017]

Ans:-This model is called as linear sequential model. This model suggests a systematic approach to software development.
The project development is divided into sequence of well-defined phases. It can be applied for long-term project and well understood product requirement.
The classical waterfall model breaks down the life cycle into an intuitive set of phases. Different phases of this model are:
- Feasibility study
- Requirements analysis and specification
- Design
- Coding and unit testing

- Integration and system testing
- Maintenance

The phases starting from the feasibility study to the integration and system testing phases are known as the development phases. All these activities are performed in a set of sequence without skip or repeat. None of the activities can be revised once closed and the results are passed to the next step for use.

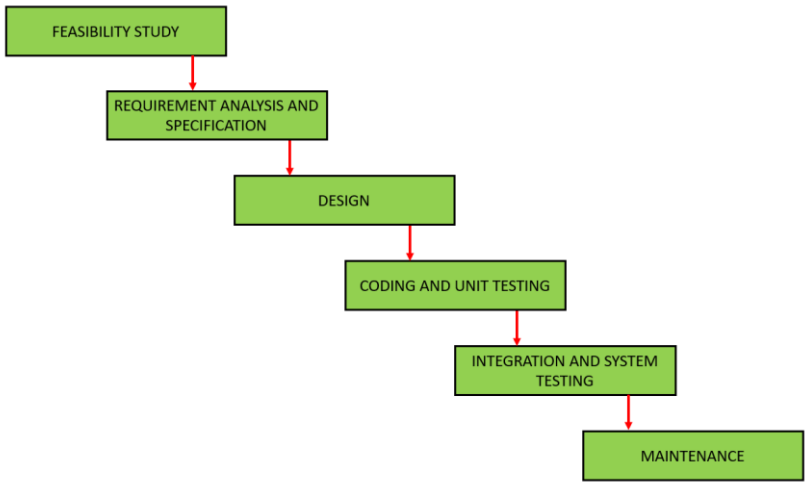
Feasibility Study

The main of the feasibility study is to determine whether it would be financially, technically and operationally feasible to develop the product. The feasibility study activity involves the analysis of the problem and collection of all relevant information relating to the product such as the different data items which would be input to the system, the processing required to be
Fig. 1.1 Classical Waterfall Model

carried out on these data, the output data required to be produced by the system.

Technical Feasibility

Can the work for the project be done with current equipment, existing software technology and available personnel?



Economic Feasibility

Are there sufficient benefits in creating the system to make the costs acceptable?

Operational Feasibility

Will the system be used if it is developed and implemented?

These phases capture the important requirements of the customer, also formulate all the different ways in which the problem can be solved are identified.

Requirement Analysis and Specifications

The goal of this phase is to understand the exact requirements of the customer regarding the product to be developed and to document them properly.

This phase consists of two distinct activities:

- Requirements gathering and analysis.
- Requirements specification.

Requirements Gathering and Analysis

This activity consists of first gathering the requirements and then analyzing

the gathered requirements.

The goal of the requirements gathering activity is to collect all relevant information regarding the product to be developed from the customer with a view to clearly understand the customer requirements.

Once the requirements have been gathered, the analysis activity is taken up.

Requirements Specification

The customer requirements identified during the requirement gathering and analysis activity are organized into a software requirement specification (SRS) document. The requirements describe the “what” of a system, not the

"how". This document written in a natural language contains a description of what the system will do without describing how it will be done. The most important contents of this document are the functional requirements, the nonfunctional requirements and the goal of implementation. Each function can be characterized by the input data, the processing required on the input data and the output data to be produced. The non-functional requirements identify the performance requirements, the required standards to be followed etc. The SRS document may act as a contract between the development team and customer.

Design

The goal of this phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language. Two distinctly different design approaches are being used at present. These are:

- Traditional design approach
- Object-oriented design approach

Traditional Design Approach

The traditional design technique is based on the data flow oriented design approach.

The design phase consists of two activities: first a structured analysis of the requirements specification is carried out, second structured design activity. Structured analysis involves preparing a detailed analysis of the different functions to be supported by the system and identification of the data flow among the functions. Structured design consists of two main activities: architectural design (also called high level design) and detailed design (also called low level design).

High level design involves decomposing the system into modules, representing the interfaces and the invocation relationships among the modules. Detailed design deals with data structures and algorithm of the modules.

Object-Oriented Design Approach

In this technique various objects that occur in the problem domain and the solution domain are identified and the different relationships that exist among these objects are identified.

Coding and Unit Testing

The purpose of the coding and unit testing phase of software development is to translate the software design into source code. During testing the major activities are centred on the examination and modification of the code. Initially small units are tested in isolation from rest of the software product. Unit testing also involves a precise definition of the test cases, testing criteria and management of test cases.

Integration and System Testing

During the integration and system testing phase the different modules are integrated in a planned manner. Integration of various modules are normally

carried out incrementally over a number of steps. During each integration step previously planned modules are added to the partially integration system and the resultant system is tested. Finally, after all the modules have been successfully integrated and tested system testing is carried out.
The goal of system testing is to ensure that the developed system confirms to its requirements laid out in the SRS document. System testing usually consists of three different kinds of testing activities:

- α –testing: α testing is the system testing performed by the development team.
- β –testing: This is the system testing performed by a friendly set of customers.
- Acceptance testing: This is the system testing performed by the customer himself after the product delivery to determine whether to accept the delivered product or to reject it.

System testing is normally carried out in a planned manner according to a system test plan document. The results of information and system testing are documented in the form of a test report.

Maintenance

Software maintenance is a very broad activity that includes error correction, enhancement of capabilities and optimization. The purpose of this phase is to preserve the value of the software over time. Maintenance involvesperforming the following activities.

6. What is prototype? Under what circumstances is it beneficial to contruct a prototype? Describe the prototyping model of software development? [2015][2016][2017]

Ans.)A prototype is an early sample, model, or release of a product built to test a concept or process or to act as a thing to be replicated or learned from.[1] It is a term used in a variety of contexts, including semantics, design, electronics, and software programming. A prototype is generally used to evaluate a new design to enhance precision by system analysts and users.

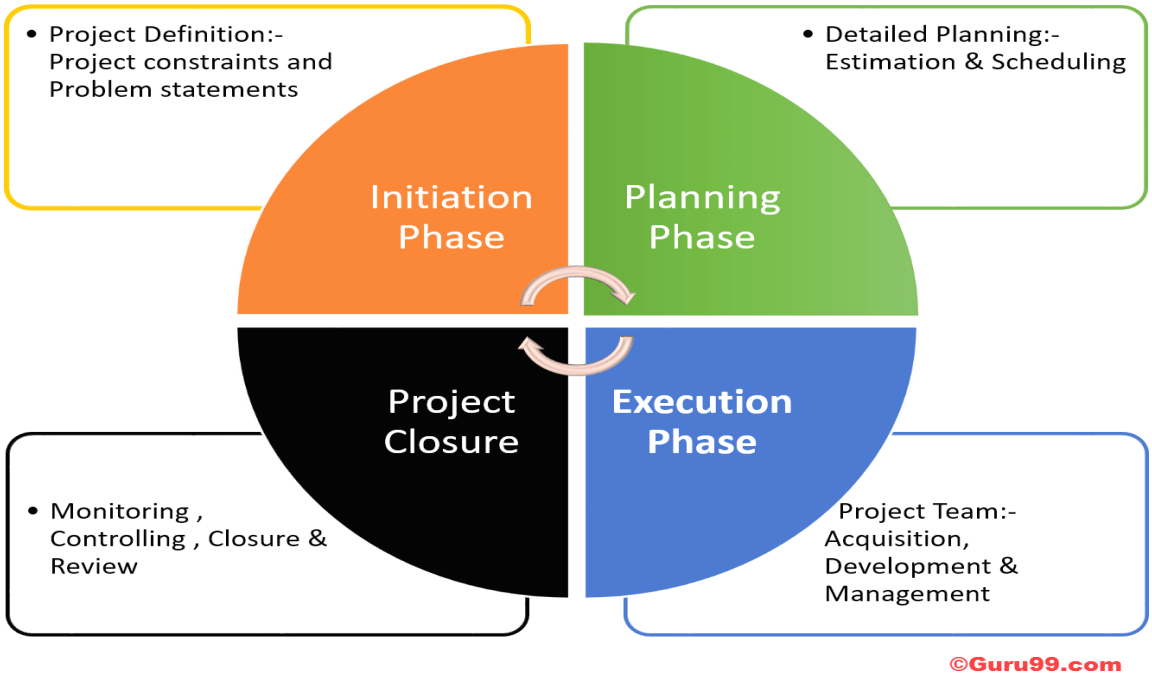
- Prototyping helps to eliminate ambiguities and improve accuracy in interpretation of system requirements and functionality
- Prototyping helps to ensure that the solution does what it is supposed to do - *not what the developer thinks it ought to do, or how*
- Prototyping allows the developer to quickly demonstrate (or *walkthrough*) a system or part thereof, albeit limited in that its purpose may be to simply provide an insight or overview of a system (eg look and feel of the user interface), or to focus on a component part in detail but possibly in isolation (egstand alone rather than integrated)

- Prototyping helps to identify and address problems early on (eg missing, confusing or misunderstood features)
- Prototyping allows the developer to explore ideas and exchange feedback with the client and end-user. This is an important step in preparing to develop a solution that is fit for purpose, does what it needs to do, and does it well
- Prototyping helps to firm up how the final solution will look and function. Acceptance allows the developer to progress to the next stage and to be focussed on what needs to be done
- Prototyping gives the client and end-user a greater sense of involvement, ownership and a better appreciation of the final solution
- Prototyping helps the developer to estimate development costs, timescale, skills and potential resource requirements
- Prototyping serves as a useful reference point - *in that it can be referred back to as necessary (eg as a reminder or even in the event of a dispute further into he development lifecycle)*

7. **When does the project planning activity start and end in a software life cycle? List the major activities, Software projects managers performs during project planning. [2016][2017]**

Ans:-The Project Life Cycle is a series of activities which are essential for accomplishing project objectives or targets. Projects may have different dimensions and difficulty level, but, whatever the size: large or small, may be all projects could be mapped to the given lifecycle structure. This life cycle for the project includes four phases-

- Initiation Phase
- Planning Phase
- Execution Phase
- Monitoring, Controlling & Closing Phase



We will first look into Initiation Phase

Project Initiation Phase

Initiation phase defines those processes that are required to start a new project. The purpose of the project initiation phase is to determine what the project should accomplish.

This phase mainly composed of two main activities

- Develop a Project Charter and
- Identify Stakeholders

All the information related to the project are entered in the Project Charter and Stakeholder Register. When the project charter is approved, the project becomes officially authorized.

Project Charter

The Project Charter defines the project's main elements

- Project goals
- Project constraints and Problem statements
- Assign project manager
- Stakeholder list
- High-level schedule and budget
- Milestones
- Approvals

This document allows a project manager to utilize organizational resources for the sake of the project. To create a project charter, the inputs required will be enterprise environment factor, business case, agreements, a project statement of work and organizational process assets.

A stakeholder can Identifying Stakeholders

influence the success and failure of the project. To note down the information about the stakeholder, a Stakeholder Register is used.

The stakeholder register will have information like

- Type of stakeholder
- Expectation of stakeholder
- Role in Project (Business Analyst, Tech architect, Client PM)
- Designation (Director, Business Lead, etc.)
- Type Communication (Weekly/Monthly)
- Influence on the project (Partial/Supportive/Influensive)

The other activities involved in initiating process group are:

- Assigning the project manager
- Determining the stakeholder needs, expectations and high-level requirements
- Define the project success criteria
- Identify particular budget for particular stage
- Make sure that the project is aligned with the organizations strategic goal

The stakeholder register and project charter are used as inputs to the other process groups such as planning process group.

Project Planning Phase

Project Planning phase covers about 50% of the whole process. Planning phase determines the scope of the project as well as the objective of the project. It begins with the outputs of initiation phase (charter, preliminary scope statement, and project manager). The output of the planning phase serves as the input for the execution phase.

The important aspects of planning process are

- Planning phase should not be executed before your initial planning is finished
- Until the execution process does not start, you should not stop revising plans

Create WBS

For any successful project WBS (Work Breakdown Structure) is important. Following steps are to create WBS.

- Conduct a brainstorm to list all the tasks
- Involve your whole team for brainstorming
- Write down the structure tree of the task also known as WBS (work breakdown structure)
- Further breakdown your top WBS into a hierarchical set of activities, for instance, categories, sub-categories, etc. For example hardware, software, trainee, management teams, etc.
- Define how to record the items into your WBS
- Ask other people - it can be an expert, experienced personnel, etc.
- Granularity- how detailed your task should you have? Estimating cost and time for higher granularity is hard while for lower granularity it will be bogged down with too detailed information
- Granularity should be of right level not too high or not too low

Planning Schedule Management

Plan Scheduling is the process of establishing the procedure, policies and documentation for planning, managing, executing and controlling the project schedule. The inputs in these activities include

- Project management plan
- Project Charter
- Enterprise environmental factors

Organizational process assets

[6Marks question Answer]

1. **Distinguish between a program and a software? [2018][2016]**

PROGRAM	SOFTWARE
->Programs are developed byindividuals for their ersonal use.	->s/w products are develoed by multiple users.
->They are in small size.	->s/w products have large in size.
->It have limited functionality.	->It has good functionality proper users manual, systematically designed ,carefully implemented and thoroughly tested.
->The author of a program himself uses and maintains his program therefore lack good user interface and	->It has a large no of users and therefore have good user interface and good documentation,support.

predocumentation.	
->The programs developed by a student as pert of his class assignment are programs and not s/w product.	->A s/w product consists of only of the program code but also of alltheassociated documents such as the requirements specification document,the design document ,the test document,the user manuals and so on.
->Programmers who write program.	->S/W engineers who develop s/w products .Since a groupof s/w engineers usually works together in a team to developer a s/w product.

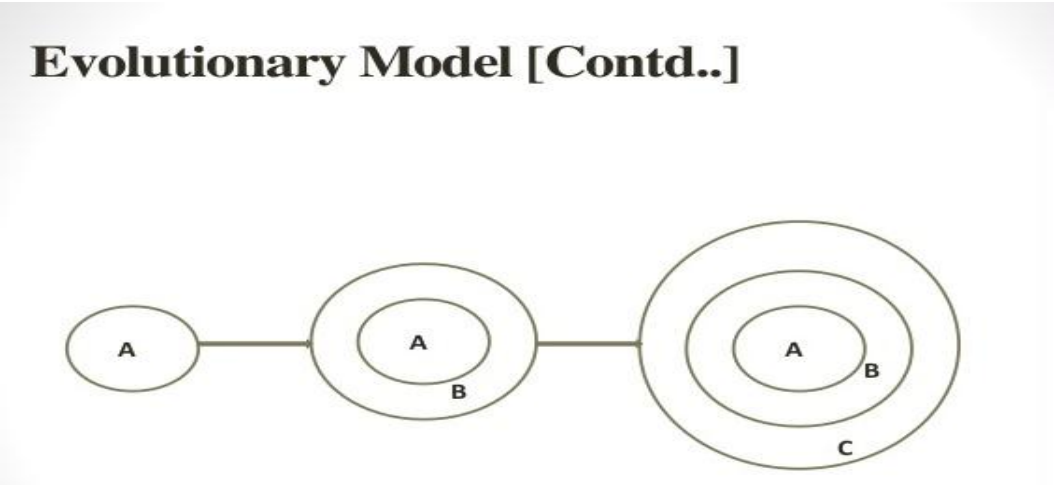
2. **What is a life cycle model? Explain Evolutionary model of s/w development? [2015][2016]**

   Ans:-A s/w life cycle is the series of identifiable stages that a s/w product undergoes during its lifetime.

- The first stage in the life cycle of any software product is usually the feasibility study stage.
- Commonly the subsequent stages are:requirements,analysis,specification,design,coding,testing and maintenance.
- Each of these stages are called a life cycle phase.

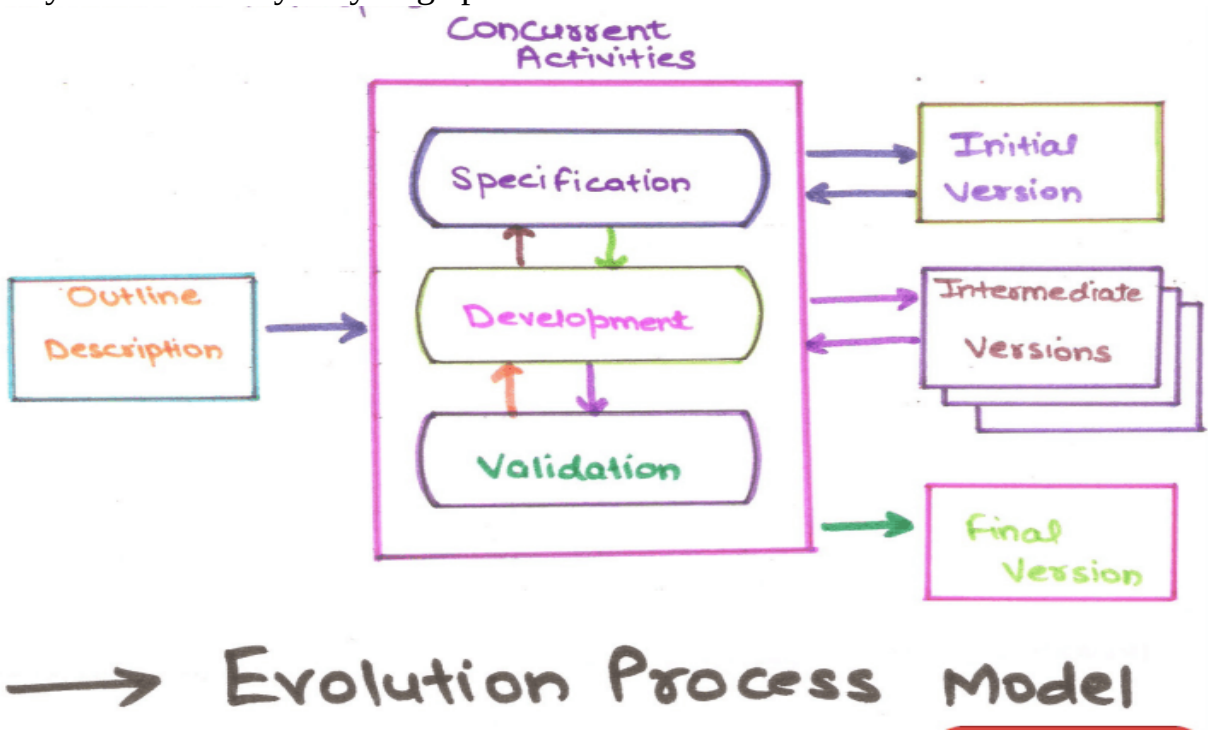
Evolutionary model of software Development

- This life cycle model is also referred to as the "Successive version" modeland sometimes as the "incremental model".
- In this life cycle model ,the s/w is first broken down into several modules.(function units) which can be incrementally constructed and delivered**.**



The evolutionary model is used when the customer prefers to receive the products in increments rather than waiting for the full product to be developed and delivered. The evolutionary model is very popular for object oriented software

development project. The main disadvantage of the successive versions model is that for most practical problems it is difficult to divide the problem into several functional units which can be incrementally implemented and delivered. Theevolutionary model is normally useful for only very large products.



→ Evolution Process Model

[2 marks questions with answer]

1. Define Software? [2016][2017]

Ans:-Software is a program or set of instructions which instructing the computer to do a specific task.

- Software are of two types
  1. System software
  2. Application software

2.What does feasibility mean in software Engineering?

Ans. Software Requirements Analysis (Feasibility Study) In this phase, the development team visits the user and studies their system. They investigate the need for development in the given system. The requirements-gathering process is intensified and focuses specifically on software.

3.What are the main features of software engineering?

Ans:-**Simplicity:** Software code should be written in a simple and concise manner. Simplicity should be maintained in the organization, implementation, and design of the software code.

Modularity: Breaking the software into several modules not only makes it easy to understand but also easy to debug. With the modularity feature, the same code segment can be reused in one or more software programs.

Design: Software code is properly designed if it is presented in a proper manner. The design of the software should be decided before beginning to write the software code. Writing the software code in a specific, consistent style helps other software developers in reviewing it.

Efficiency: A program is said to be efficient if it makes optimal use of the available resources.

4. Differentiate between program and software product?

Software	Program
The software is a broad term which is designed to perform some specific set of operations.	**A program is set of instructions which perform only a specific type of task.**

A software consists of bundles of programs and data files. Programs in a specific software use these data files to perform a dedicated type of tasks.	**A program consists of a set of instructions which are coded in a programming language like C, C++, PHP, Java etc.**
A software can be classified into two categories: application software and system software. An application software comes in wide range of varieties like a text editor, media player, web browser, video player, video editor, image editor. Different types of application software provide a different type of services.	**A program cannot be classified into various categories.** **Source code in a program is written for small jobs.**

Chapter –II

[7 MARKS QUESTION AND ANSWER]

1. Explain the FP and LOC metrics? [2016][2017][2018]

Ans:-The size of a project is obviously not the number of bytes that the source code occupies. The project size is a measure of the problem complexity in terms of the effort and time required to develop the product.
Two metrics are widely used to estimate size:

- Lines of Code (LOC)
- Function Point (FP)

Lines Of Code (LOC)

LOC can be defined as the number of delivered lines of code in software excluding the comments and blank lines. LOC depends on the programming language chosen for the project. The exact number of the lines of code can only be determined after the project is complete since less information about the project is available at the early stage of development.
In order to estimate the LOC count at the beginning of a project, project managers usually divide the problem into modules and each modules into sub modules and a so on until the sizes of the different leaf level modules can be approximately predicted.

Disadvantages:

- LOC is language dependent. A line of assembler is not the same as a line of COBOL.
- LOC measure correlates poorly with the quality and efficiency of the code. A larger code size does not necessary imply better quality or

higher efficiency.

- LOC metrics penalizes use of higher level programming languages, code reuse etc.
- It is very difficult to accurately estimate LOC in the final product from the problem specification. The LOC count can be accurately computed only after the code has been fully developed.

Function Point Metric

♦ Function Points measure software size by quantifying the functionality provided to the user based solely on logical design and functional specifications

♦ Function point analysis is a method of quantifying the size and complexity of a software system in terms of the functions that the system delivers to the user

♦ It is independent of the computer language, development methodology, technology or capability of the project team used to develop the application.

♦ Function point analysis is designed to measure business applications (not scientific applications) .

♦ Function points are independent of the language, tools, or methodologies used for implementation

♦ Function points can be estimated early in analysis and design

♦ Since function points are based on the system user's external view of the system, non-technical users of the software system have a better understanding of what function points are measuring.

Objectives of Function Point Counting

♦ Measure functionality that the user requests and receives

♦ Measure software development and maintenance independently of technology used for implementation.

2. How the size of a software project is estimated by function point method ,discuss?[2018]

Ans:-Function Points measure software size by quantifying the functionality provided to the user based solely on logical design and functional specifications

♦ Function point analysis is a method of quantifying the size and complexity of a software system in terms of the functions that the system delivers to the user

♦ It is independent of the computer language, development methodology, technology or capability of the project team used to develop the application.

Steps of Function Point Counting

♦ Determine the type of function point count

♦ Identify the counting scope and application boundary

♦ Determine the Unadjusted Function Point Count

♦ Count Data Functions

♦ Count Transactional Functions

♦ Determine the Value Adjustment Factor
♦ Calculate the Adjusted Function Point Count

Function point metric estimates the size of a software product directly from the problem specification.

The different parameters are:

- **Number Of Inputs:**

Each data item input by the user is counted.

- **Number Of Outputs:**

The outputs refers to reports printed, screen outputs, error messages produced etc.

- **Number Of Inquiries:**

It is the number of distinct interactive queries which can be made by the users.

- **Number Of Files:**

Each logical file is counted. A logical file means groups of logically related data. Thus logical files can be data structures or physical files.

Number Of Interfaces:

Here the interfaces which are used to exchange information with other systems. Examples of interfaces are data files on tapes, disks, communication links with other systems etc.

Function Point (FP) is estimated using the formula: FP = UFP (Unadjusted Function Point) * TCF (Technical Complexity Factor)

UFP = (Number of inputs) * 4 + (Number of outputs) * 5 + (Number of inquiries) * 4 + (Number of files) * 10 + Number of interfaces) * 10

TCF = DI (Degree of Influence) * 0.01

The unadjusted function point count (UFP) reflects the specific countable functionality provided to the user by the project or application.

Example- Once the unadjusted function point (UFP) is computed, the technical complexity factor (TCF) is computed next. The TCF refines the UFP measure by considering fourteen other factors such as high transaction rates, throughput and response time requirements etc. Each of these 14 factors is assigned a value from 0 (not present or no influence) to 6 (strong influence). The resulting numbers are summed, yielding the total degree of influence (DI). Now, the TCF is computed as (0.65+0.01*DI). As DI can vary from 0 to 70, the TCF can vary from 0.65 to 1.35.

Finally FP = UFP *TCF

3. What is software reliability? Explain the different types of reliability metrics used in software engineering? [2016][2017]

Ans.)Software Reliability is the probability of failure-free software operation for a specified period of time in a specified environment. Software Reliability is also an important factor affecting system reliability.

The size of a project is obviously not the number of bytes that the source

code occupies. The project size is a measure of the problem complexity in terms of the effort and time required to develop the product.
Two metrics are widely used to estimate size:

- Lines of Code (LOC)
- Function Point (FP)

Lines Of Code (LOC)

LOC can be defined as the number of delivered lines of code in software excluding the comments and blank lines. LOC depends on the programming language chosen for the project. The exact number of the lines of code can only be determined after the project is complete since less information about the project is available at the early stage of development.
In order to estimate the LOC count at the beginning of a project, project managers usually divide the problem into modules and each modules into sub modules and a so on until the sizes of the different leaf level modules can be approximately predicted.

Function Point Metric

♦ Function Points measure software size by quantifying the functionality provided to the user based solely on logical design and functional specifications

♦ Function point analysis is a method of quantifying the size and complexity of a software system in terms of the functions that the system delivers to the user

♦ It is independent of the computer language, development methodology, technology or capability of the project team used to develop the application.

♦ Function point analysis is designed to measure business applications (not scientific applications) .

♦ Function points are independent of the language, tools, or methodologies used for implementation

♦ Function points can be estimated early in analysis and design

♦ Since function points are based on the system user's external view of the system, non-technical users of the software system have a better understanding of what function points are measuring.

4. What is risk? Explain the risk management procedures used during software development. [2015][2016]

Ans:-Risk is an expectation of loss, a potential problem that may or may not occur in the future. It is generally caused due to lack of information, control or time.A possibility of suffering from loss in software development process is called a software risk.

Risk is an expectation of loss, a potential problem that may or may not occur in the future. It is generally caused due to lack of information, control or time.A possibility of suffering from loss in software development process is called a

software risk. Loss can be anything, increase in production cost, development of poor quality software, not being able to complete the project on time. Software risk exists because the future is uncertain and there are many known and unknown things that cannot be incorporated in the project plan. A software risk can be of two types (a) internal risks that are within the control of the project manager and (2) external risks that are beyond the control of project manager. Risk management is carried out to:

1. Identify the risk
2. Reduce the impact of risk
3. Reduce the probability or likelihood of risk
4. Risk monitoring

A project manager has to deal with risks arising from three possible cases:

1. Known knowns are software risks that are actually facts known to the team as well as to the entire project. For example not having enough number of developers can delay the project delivery. Such risks are described and included in the Project Management Plan.
2. Known unknowns are risks that the project team is aware of but it is unknown that such risk exists in the project or not. For example if the communication with the client is not of good level then it is not possible to capture the requirement properly. This is a fact known to the project team however whether the client has communicated all the information properly or not is unknown to the project.
3. Unknown Unknowns are those kind of risks about which the organization has no idea. Such risks are generally related to technology such as working with technologies or tools that you have no idea about because your client wants you to work that way suddenly exposes you to absolutely unknown unknown risks.

Software risk management is all about risk quantification of risk. This includes:

1. Giving a precise description of risk event that can occur in the project
2. Defining risk probability that would explain what are the chances for that risk to occur
3. Defining How much loss a particular risk can cause
4. Defining the liability potential of risk

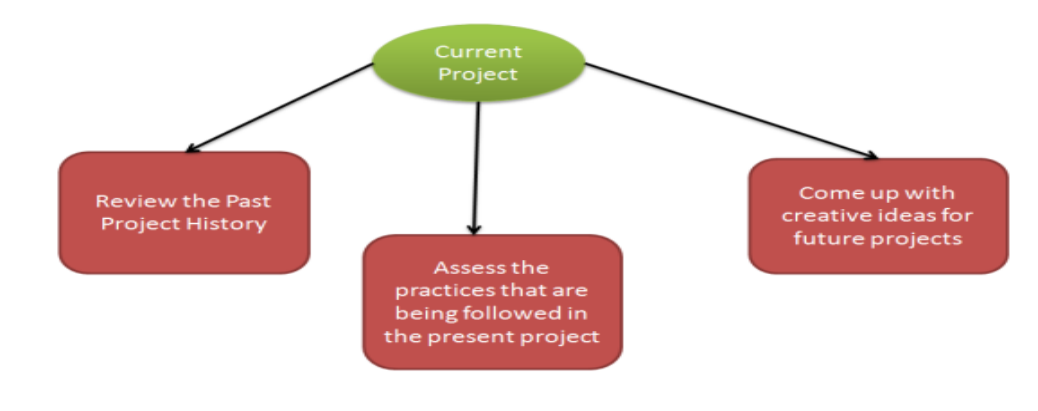
Risk Management comprises of following processes:

1. Software Risk Identification
2. Software Risk Analysis
3. Software Risk Planning
4. Software Risk Monitoring

These Processes are defined below.

Software Risk Identification

In order to identify the risks that your project may be subjected to, it is important to first study the problems faced by previous projects. Study the project plan properly and check for all the possible areas that are vulnerable to some or the other type of risks. The best ways of analyzing a project plan is by converting it to a flowchart and examine all essentialareas. It is important to conduct few brainstorming sessions to identify the known unknowns that can affect the project. Any decision taken related to technical, operational, political, legal, social, internal or external factors should be evaluated properly.



Software Risk Identification

In this phase of Risk management you have to define processes that are important for risk identification. All the details of the risk such as unique Id, date on which it was identified, description and so on should be clearly mentioned.

Software Risk Analysis

Software Risk analysisis a very important aspect of risk management. In this phase the risk is identified and then categorized. After the categorization of risk, the level, likelihood (percentage) and impact of the risk is analyzed. Likelihood is defined in percentage after examining what are the chances of risk to occur due to various technical conditions. These technical conditions can be:

1. Complexity of the technology
2. Technical knowledge possessed by the testing team
3. Conflicts within the team
4. Teams being distributed over a large geographical area
5. Usage of poor quality testing tools

With impact we mean the consequence of a risk in case it happens. It is important to know about the impact because it is necessary to know how a business can get affected:

1. What will be the loss to the customer
2. How would the business suffer
3. Loss of reputation or harm to society
4. Monetary losses
5. Legal actions against the company
6. Cancellation of business license

Level of risk is identified with the help of:

Qualitative Risk Analysis: Here you define risk as:

- High
- Low
- Medium

Quantitative Risk Analysis: can be used for software risk analysis but is considered inappropriate because risk level is defined in % which does not give a very clear picture.

5 .What is software quality? Explain the evolution of software quality managements system? [2016][2017]

Ans.)software quality refers to two related but distinct notions that exist wherever quality is defined in a business context:
Software functional quality reflects how well it complies with or conforms to a given design, based on functional requirements or specifications. That attribute can also be described as the fitness for purpose of a piece of software or how it compares to competitors in the marketplace as a worthwhile product.[1] It is the degree to which the correct software was produced.
Software structural quality refers to how it meets non-functional requirements that support the delivery of the functional requirements, such as robustness or maintainability. It has a lot more to do with the degree to which the software works as needed.

Chapter 3. Software processes

Table of ContentsSoftware process models.

- The waterfall model
- Evolutionary development
- Component-based software engineering
- Process iteration
- Incremental delivery
- Spiral development

- Process activities
- Software specification
- Software design and implementation
- Software validation
- Software evolution
- The Rational Unified Process
- Exercises

A software development process, also known as a software development lifecycle, is a structure imposed on the development of a software product. A software process is represented as a set of work phases that is applied to design and build a software product. There is no ideal software process, and many organisations have developed their own approach to software development. Software development processes should make a maximum use of the capabilities of the people in an organisation and the specific characteristics of the systems that are being developed [1][14][15].

There are some fundamental activities that are common to all software processes:

Software specification: In this activity the functionality of the software and constraints on its operation must be defined.

Software design and implementation: The software that meets the specification is produced.

Software validation: The software must be validated to ensure that it has all the functionalities what the customer needs.

Software evolution: he software must evolve to meet changing customer needs.

Software process models:

A software process model is an abstract representation of a software process. In this section a number of general process models are introduced and they are presented from an architectural viewpoint. These models can be used to explain different approaches to software development. They can be considered as process frameworks that may be extended and adapted to create more specific software engineering processes. In this chapter the following process models will be introduced:

The waterfall model. In this model of software process the fundamental process activities of specification, development, validation and evolution are represented as sequential process phases such as requirements specification, software design, implementation, testing and so on.

Evolutionary development. This approach interleaves the activities of specification, development and validation. An initial system is rapidly developed from abstract specifications. Then the initial system is refined by customer inputs to produce a system that satisfies the customer's needs.

Component-based software engineering. The process models that use this approach are based on the existence of a significant number of reusable components. The system development process focuses on integrating these components into a system rather than developing them.

These three generic process models are widely used in current software engineering practice. They are not mutually exclusive and are often used together, especially for large systems development. Sub-systems within a larger system may be developed using different approaches. Therefore, although it is convenient to discuss these models separately, in practice, they are often combined.

The waterfall model:

The waterfall model was the first software process model to be introduced (Figure 3.1.). It is also referred to as a linear-sequential life cycle model. The principal stages of the model represent the fundamental development activities:

Requirements analysis and definition. Software requirements specification establishes the basis for agreement between customers and contractors or suppliers on what the software product is to do. Software requirements specification permits a rigorous assessment of requirements before design can begin. It should also provide basis for estimating product costs, risks, and schedules.

System and software design. Design activity results in the overall software architecture. Software design involves identifying and describing the fundamental software system components and their relationships. The systems design process partitions the requirements to either hardware or software components.

Implementation and unit testing. During this phase, the software design is realised as a set of software components. Components are tested ensuring each component meets its specification.

Integration and system testing. The program units or components are integrated and tested as a complete system to ensure that the software requirements have been met. After successful testing, the software system is delivered to the customer.

Operation and maintenance. The system is delivered and deployed and put into practical use. Maintenance involves correcting errors which were not discovered in

earlier stages of the life cycle, improving the implementation of system units and providing new functionalities as new requirements emerge.

6 .What are PERT charts and GANTT charts and when to use them?

Ans.)PERT charts are generally used before a project begins to plan and determine the duration of each task. Gantt charts are used while a project is happening to break projects into smaller tasks and highlight scheduling constraints.

Like PERT charts, Gantt charts break projects into smaller tasks and highlight scheduling constraints. However, project managers use Gantt charts while a project is happening—they schedule tasks by date and show how much work has been completed. Every activity is represented with a bar that stretches from the start date to the end date of that activity.

Gantt chart example

PERT charts are generally used before a project begins to plan and determine the duration of each task—so they don't have to show the actual dates of your project. They also do a better job of showing whether certain tasks need to be completed in order or whether they can be completed simultaneously. Use a PERT chart if you need to:

Show the interdependency of certain tasks.
Anticipate the amount of time it'll take to complete a project.
Determine the critical path to meet your deadlines.
Plan for large or more complex projects.
If you decide that a Gantt chart will best fit your needs, see how you can create a simple Gantt chart right in Lucidchart. If you decide that you need a PERT chart, continue onward! This article will show you how to create one.

7. Explain different types of project estimation techniques? [2016][2017][2018]

Ans:- There are basically two approaches for estimating project parameters [4]. They are:

1. Top-down estimation approach
2. Bottom-up estimation approach

Top-down estimation approach:

Top-down estimation approach is usually used at the initial stages of the project. This estimation is usually carried out by the top managers who have little knowledge of the processes involved in the completion of the project. The input to this estimation is either information or the experience of the manager

carrying out the estimation. These top-down estimation methods are often used to evaluate the project proposal. In most cases, the best results can be achieved in estimation only when one used both top-down and bottom-up estimation methods. However, it is practically not possible to carry out bottom-up methods until the Work Breakdown Structure (WBS) are clearly defined. In such cases, top-down estimates are used until the WBS becomes available.

There are many methods in top-down approach listed below [4]:

Consensus methods: This estimation method uses experience of a group of people to estimate the project parameters. This method involves project meetings, a place where these people can discuss, argue and finally come to a conclusion from their best guess estimate. The Delphi method comes under this category.

Ratio methods: These estimation methods use ratios to estimate project times and costs. For example, in a construction work, the total cost of the project can be estimated by knowing the number of square feet. Likewise, a software project is estimated by its complexity and its features.

Approximation methods: This estimation method is very useful when the project to be estimated is closely related to any of the previous projects in terms of its features and costs. By using the historical data of the estimates, good estimates can be approximated with very little effort.

Function point methods: Many software projects are usually estimated using weighted macro variables called "function points". Function points can be number of inputs, number of outputs, number of inquiries, number of data files, and number of interfaces. These function points are weighted again with a complexity level and summed up to get the total cost or duration estimates of the project.

Bottom-up estimation approach:

Top-down estimation approach can usually be put in practice once the project is defined or once there is some progress in the project. This means, this estimation is more into work package level, which are responsible for low-cost estimates and efficient methods. It is often recommended that this estimation is usually carried out by people most knowledgeable about the estimate needed. The cost, time, resource estimates from the work packages can be checked with the associated accounts to major deliverables. Also, these estimates in later

stages can be consolidated into phased networks, resource schedules, and budgets that used for control. Additionally, customer will get an opportunity to compare the low-cost, efficient method with any imposed restrictions, using bottom-up approach [4].

There are many methods in top-down approach listed below [4]:

Template methods: If the project to be estimated is similar to any of the past projects, then estimates of the past projects can be used as starting point estimates for the new project. This is similar to approximation estimation in top-down approach.

Parametric procedures: These parametric procedures are same like ratio methods in top-down approach. However, here the parametric procedures are applied on specific tasks.

Detailed estimates for WBS work packages: This is usually most reliable method of all estimation methods. The reason for this is that here the estimates are performed by people responsible for the work packages in Work Breakdown Structure. These people have prior knowledge or experience upon the tasks they perform specified in WBS, because of which the estimates are usually most reliable.

In addition to the top-down and bottom-up approaches, there is another kind of estimating which is a hybrid of the above two approaches. This is called as Phase Estimating. When there is unusual amount of uncertainty is surrounded by the project, people go for phase estimating. In this approach, two-estimate system is used over the life-cycle of the project. The whole project is initially divided into phases. Then a detailed estimate is developed for the immediate phase, and a macro-estimate is mode for the remaining phases of the project.

[5 marks question]

1. **What do you mean by good software design? What are its characteristics? [2015][2016][2018]**

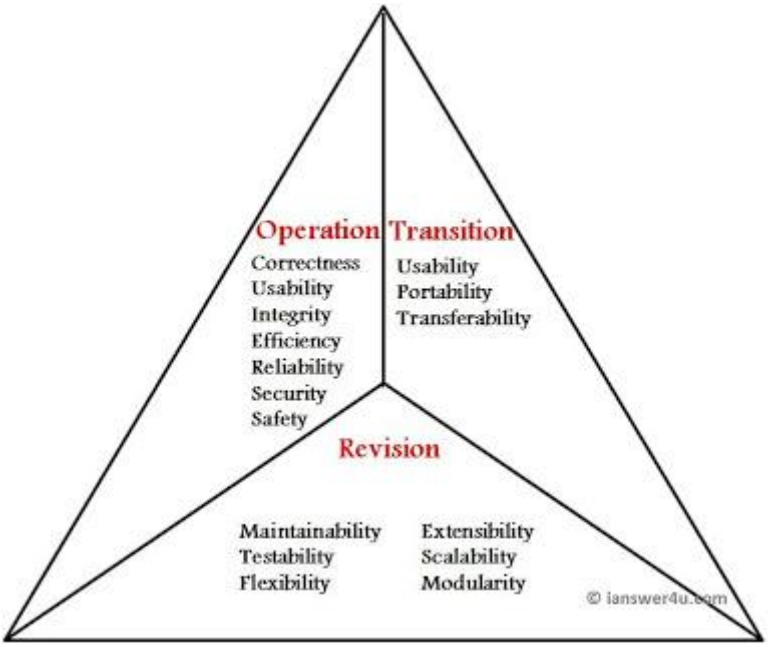
- While developing any kind of software product, the first question in any developer's mind is, “What are the qualities that a good software should have ?" Well before going into technical characteristics, I would like to state the obvious expectations one has from any software. First and foremost, a software product

must meet all the requirements of the customer or end-user. Also, the cost of developing and maintaining the software should be low. The development of software should be completed in the specified time-frame.

Well these were the obvious things which are expected from any project (and software development is a project in itself). Now lets take a look at Software Quality factors. These set of factors can be easily explained by Software Quality Triangle. The three characteristics of good application software are :-

1) Operational Characteristics
2) Transition Characteristics
3) Revision Characteristics

- **Software Quality Triangle**



Software Quality Triangle with characteristics

- 16 Characteristics of a Good Software

- **What Operational Characteristics should a software have ?**
- These are functionality based factors and related to 'exterior quality' of software. Various Operational Characteristics of software are :

a) **Correctness:** The software which we are making should meet all the specifications stated by the customer.

b) **Usability/Learnability:** The amount of efforts or time required to learn how to use the software should be less. This makes the software user-friendly even for IT-illiterate people.

c) **Integrity :** Just like medicines have side-effects, in the same way a software may have a side-effect i.e. it may affect the working of another application. But a quality software should not have side effects.

d) **Reliability :** The software product should not have any defects. Not only this, it shouldn't fail while execution.

e) **Efficiency :** This characteristic relates to the way software uses the available resources. The software should make effective use of the storage space and execute command as per desired timing requirements.

f) **Security :** With the increase in security threats nowadays, this factor is gaining importance. The software shouldn't have ill effects on data / hardware. Proper measures should be taken to keep datasecure from external threats.

g) **Safety :** The software should not be hazardous to the environment/life.

(c) What are PERT charts and GANTT charts and when to use them?

Ans.)PERT charts are generally used before a project begins to plan and determine the duration of each task. Gantt charts are used while a project is happening to break projects into smaller tasks and highlight scheduling constraints.

- Like PERT charts, Gantt charts break projects into smaller tasks and highlight scheduling constraints. However, project managers use Gantt charts while a project is happening—they schedule tasks by date and show how much work has been completed. Every activity is represented with a bar that stretches from the start date to the end date of that activity.
-
- Gantt chart example
-
- PERT charts are generally used before a project begins to plan and determine the duration of each task—so they don’t have to show the actual dates of your project. They also do a better job of showing whether certain tasks need to be completed in order or whether they can be completed simultaneously. Use a PERT chart if you need to:
-
- Show the interdependency of certain tasks.
- Anticipate the amount of time it’ll take to complete a project.

- Determine the critical path to meet your deadlines.
- Plan for large or more complex projects.
- If you decide that a Gantt chart will best fit your needs, see how you can create a simple Gantt chart right in Lucidchart. If you decide that you need a PERT chart, continue onward! This article will show you how to create one.

2 .**Explain different Metrics to measure software quality? [2015][2016][2017]**

Ans:-The size of a project is obviously not the number of bytes that the source code occupies. The project size is a measure of the problem complexity in terms of the effort and time required to develop the product.
Two metrics are widely used to estimate size:
- Lines of Code (LOC)
- Function Point (FP)

Lines Of Code (LOC)

LOC can be defined as the number of delivered lines of code in software excluding the comments and blank lines. LOC depends on the programming language chosen for the project. The exact number of the lines of code can only be determined after the project is complete since less information about the project is available at the early stage of development.
In order to estimate the LOC count at the beginning of a project, project managers usually divide the problem into modules and each modules into sub modules and a so on until the sizes of the different leaf level modules can be approximately predicted.
Disadvantages:
- LOC is language dependent. A line of assembler is not the same as a line of COBOL.
- LOC measure correlates poorly with the quality and efficiency of the code. A larger code size does not necessary imply better quality or higher efficiency.

Function Point Metric

♦ Function Points measure software size by quantifying the functionality provided to the user based solely on logical design and functional specifications
♦ Function point analysis is a method of quantifying the size and complexity of a software system in terms of the functions that the system delivers to the user
♦ It is independent of the computer language, development methodology, technology or capability of the project team used to develop the application.
♦ Function point analysis is designed to measure business applications (not scientific applications) .

♦ Function points are independent of the language, tools, or methodologies used for implementation
♦ Function points can be estimated early in analysis and design
♦ Since function points are based on the system user's external view of.

3. List the major responsibilities of a software project manager? [2015][2016][2017]

*Ans:-**1. Planning the activities***

A project manager needs to set an impact strategy that includes a full list of activities that are important for the project. The key responsibility of a project manager includes planning. The project manager needs to define the scope of the project and develop a project schedule accordingly. In general, when a project manager is planning the activities it is important to target the activities effectively to do less but well. The procedures should be efficient enough to deliver the projects within specified time and budget. Also, a backup plan should be created if the situation demands.

2. Organizing a project team to perform work

Another major role of project managers has focused their team's efforts on elaborate spreadsheets, long checklists, and whiteboards. They need to develop a plan that will support the team to reach their goal easily without hindering the performance. It is their duty to organize their team to show their full potential. A project manager will have have to sometimes put on the duties of human resources like negotiating current employees' job responsibilities, managing their times and achieving their commitment to the project, bids may be required and contracts will need to be reviewed and keeping everyone in check to make sure that the team's moves along in accordance with the plan.

3. Delegating the teams

In many situations like a big project, or various tasks involved in a project, it becomes critical to delegate responsibilities to teams wisely. It is a leadership style that every project manager has to abide with and be good at it and eventually it becomes the responsibility of a project manager that needs to be learned over time. A manager should not misuse this responsibility in putting blames or degrading the team members. The tasks need to prioritize the tasks so prioritized to the team members so that they become more effective in their abilities. The managers should also understand the strength and weakness of their teams and accordingly delegate the tasks to them. So, be a good leader who creates an environment that fosters trust through meaningful delegation.

4. Controlling time management

To make a good impression on stakeholders and clients, the project managers need to look for whether the project has succeeded or failed. A project manager needs to be able to negotiate achievable deadlines and discuss the same with the team. They need to develop a project that has the following features:

- Objective
- Process
- Estimating duration
- Schedule development
- Schedule control

5. Managing deliverables

The Project Manager is also responsible for ensuring that the deliverables are delivered on time and within budget as per the business requirements. Their job is concerned with asking questions like:

- What are the changes being made in the organization?
- What is the team doing?
- Why are we doing it?
- Is there a business opportunity or risk?
- How are we going to do it?
- What are the popular project management techniques?
- Who is doing what?
- Where are the records and project documents?
- What are the specifications, schedule, meetings etc?
- When are the things being done?

[2MARKS QUESTION]

1. **What is software configuration management? [2017]**

Ans:- The deadline is rapidly approaching and the team is assembled, ready to implement the recent revision changes made to both the system hardware and software. The installation has gone well, and your team is making the final testing

arrangements and preparing to demonstrate the results to the waiting clientele. The first test is executed and the initial screen display fails, panic erupts and your team scrambles to identify the problem

A successful and effective CM program will result in:

Improved performance

System compliance with established specifications and guidelines

Lowered probability of errors

Enhanced online availability

2. What is Project Scheduling?[2018]

Ans:-Project scheduling is a mechanism to communicate what tasks need to get done and which organizational resources will be allocated to complete those tasks in what timeframe. A project schedule is a document collecting all the work needed to deliver the project on time.

3. What is project planning? [2016]

ANS:-Project Planning is an aspect of Project Management that focuses a lot on Project Integration. The project plan reflects the current status of all project activities and is used to monitor and control the project.

- The Project Planning tasks ensure that various elements of the Project are coordinated and therefore guide the project execution.
- Project Planning helps in
  – Facilitating communication
  – Monitoring/measuring the project progress, and
  – Provides overall documentation of assumptions/planning decisions

4 .**What is activity network and why it is used? [2015][2016]**

Ans:-An activity network diagram tool is used extensively in and is necessary for the identification of a project's critical path (which is used to determine the expected completion time of the project). Example: Suppose the team is tasked with improving the process of building a house.

CHAPTER-III

[7 marks question answer]

1. What is a SRS document?Exaplain the characteristics and organisation of SRS document?[2018]

Ans:-

1. **Complete**

A complete requirements specification must precisely define all the real world situations that will be encountered and the capability's responses to them. It must not include situations that will not be encountered or unnecessary capability features.

2. **Consistent**

System functions and performance level must be compatible and the required quality features (reliability, safety, security, etc.) must not contradict the utility of the system. For example, the only aircraft that is totally safe is one that cannot be started, contains no fuel or other liquids, and is securely tied down.

3. **Correct**

The specification must define the desired capability's real world operational environment, its interface to that environment and its interaction with that environment. It is the real world aspect of requirements that is the major source of difficulty in achieving specification correctness. The real world environment is not well known for new applications and for mature applications the real world keeps changing. The Y2K problem with the transition from the year 1999 to the year 2000 is an example of the real world moving beyond an application's specified requirements.

4. **Modifiable**

Related concerns must be grouped together and unrelated concerns must be separated. Requirements document must have a logical structure to be modifiable.

5. **Ranked**

Ranking specification statements according to stability and/or importance is established in the requirements document's organization and structure. The larger and more complex the problem addressed by the requirements specification, the more difficult the task is to design a document that aids rather than inhibits understanding.

6. **Testable**

A requirement specification must be stated in such as manner that one can test it against pass/fail or quantitative assessment criteria, all derived from the specification itself and/or referenced information. Requiring that a system must be "easy" to use is subjective and therefore is not testable.

7. **Traceable**

Each requirement stated within the SRS document must be uniquely identified to achieve traceability. Uniqueness is facilitated by the use of a consistent and logical scheme for assigning identification to each specification statement within the requirements document.

8. **Unambiguous**

A statement of a requirement is unambiguous if it can only be interpreted one way. This perhaps, is the most difficult attribute to achieve using natural language. The use of

weak phrases or poor sentence structure will open the specification statement to misunderstandings.

9. **Valid**

To validate a requirements specification all the project participants, managers, engineers and customer representatives, must be able to understand, analyze and accept or approve it. This is the primary reason that most specifications are expressed in natural language.

10. **Verifiable**

In order to be verifiable, requirement specifications at one level of abstraction must be consistent with those at another level of abstraction. Most, if not all, of these attributes are subjective and a conclusive assessment of the quality of a requirements specification requires review and analysis by technical and operational experts in the domain addressed by the requirements.

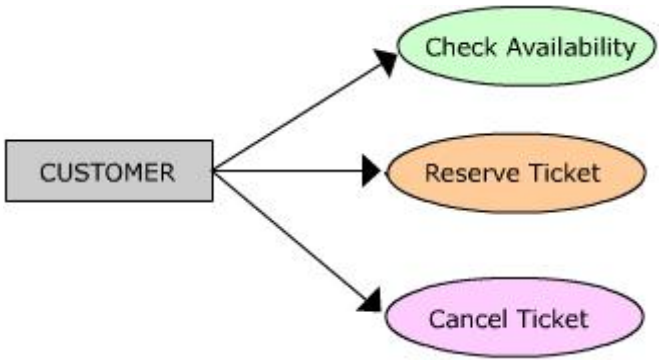
CHAPTER-4

[7MARKS QUESTION AND ANSWER]

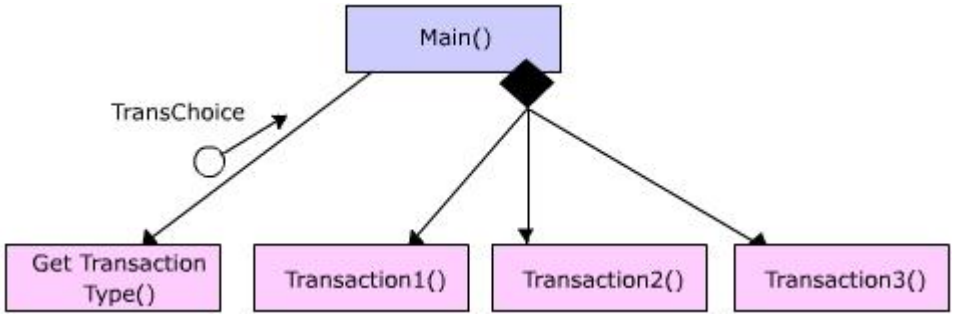
1.WHAT IS STRUTURE OF CHARTANDSTRUCTUREDDESIGN?DESCRIBE THE METHODS TO TRANSFORM THE DFD MODEL INTO A STRUCTURE CHART?
[2015][2016][2017]

Ans:-Transaction Analysis

The transaction is identified by studying the discrete event types that drive the system. For example, with respect to railway reservation, a customer may give the following transaction stimulus:



The three transaction types here are: Check Availability (an enquiry), Reserve Ticket (booking) and Cancel Ticket (cancellation). On any given time we will get customers interested in giving any of the above transaction stimuli. In a typical situation, any one stimulus may be entered through a particular terminal. The human user would inform the system her preference by selecting a transaction type from a menu. The first step in our strategy is to identify such transaction types and draw the first level breakup of modules in the structure chart, by creating separate module to co-ordinate various transaction types. This is shown as follows:



The Main ( ) which is a over-all coordinating module, gets the information about what transaction the user prefers to do through TransChoice. The TransChoice is returned as a parameter to Main ( ). Remember, we are following our design principles faithfully in decomposing our modules. The actual details of how GetTransactionType ( ) is not relevant for Main ( ). It may for example, refresh and print a text menu and prompt the user to select a choice and return this choice to Main ( ). It will not affect any other components in our breakup, even when this module is changed later to return the same input through graphical interface instead of textual menu. The modules Transaction1 ( ), Transaction2 ( ) and Transaction3 ( ) are the coordinators of transactions one, two and three respectively. The details of these transactions are to be exploded in the next levels of abstraction.

We will continue to identify more transaction centers by drawing a navigation chart of all input screens that are needed to get various transaction stimuli from the user. These are to be factored out in the next levels of the structure chart (in exactly the same way as seen before), for all identified transaction centers.

Transform Analysis

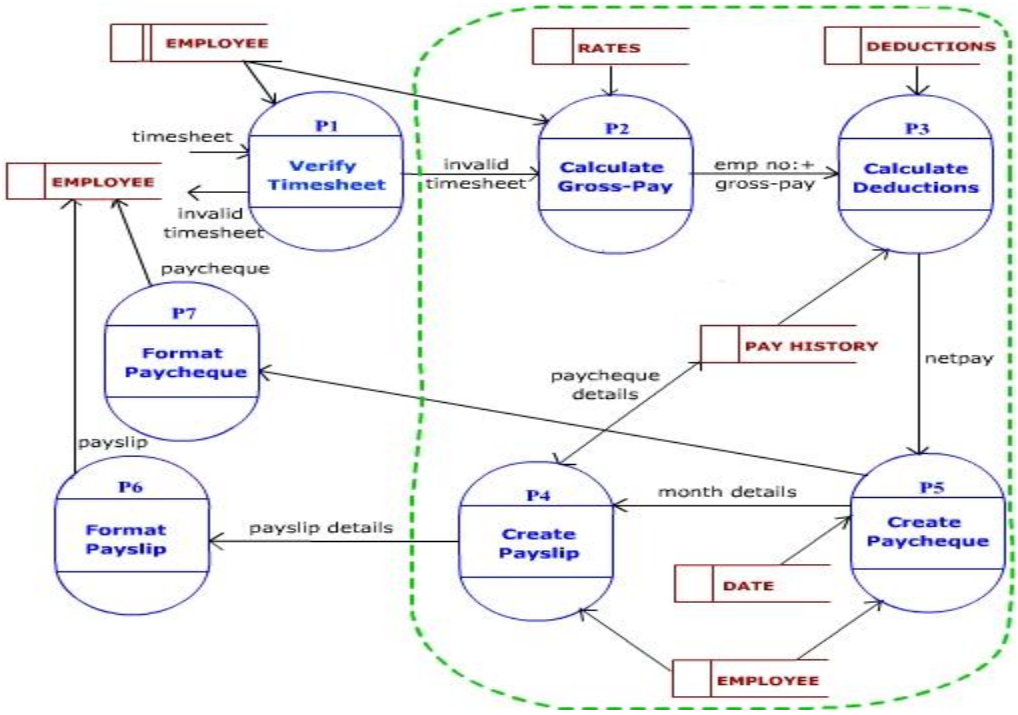
Transform analysis is strategy of converting each piece of DFD (may be from level 2 or level 3, etc.) for all the identified transaction

centers. In case, the given system has only one transaction (like a payroll system), then we can start transformation from level 1 DFD itself. Transform analysis is composed of the following five steps [Page-Jones, 1988]:

1. Draw a DFD of a transaction type (usually done during analysis phase)
2. Find the central functions of the DFD
3. Convert the DFD into a first-cut structure chart
4. Refine the structure chart
5. Verify that the final structure chart meets the requirements of the original DFD

   Let us understand these steps through a payroll system example:

- **Identifying the central transform**

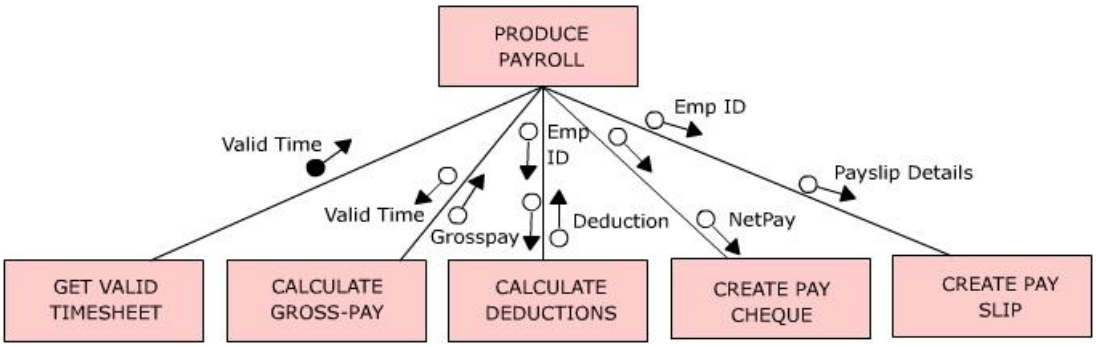


The central transform is the portion of DFD that contains the essential functions of the system and is independent of the particular implementation of the input and output. One way of identifying central transform (Page-Jones, 1988) is to identify the centre of the DFD by pruning off its afferent and efferent branches. Afferent stream is traced from outside of the DFD to a flow point inside, just before the input is being transformed into some form of output (For example, a format or validation process only refines the input – does not transform it). Similarly an efferent stream is a flow point from where

output is formatted for better presentation. The processes between afferent and efferent stream represent the central transform (marked within dotted lines above). In the above example, P1 is an input process, and P6 & P7 are output processes. Central transform processes are P2, P3, P4& P5 - which transform the given input into some form of output.

- **First-cut Structure Chart**

To produce first-cut (first draft) structure chart, first we have to establish a boss module. A boss module can be one of the central transform processes. Ideally, such process has to be more of a coordinating process (encompassing the essence of transformation). In case we fail to find a boss module within, a dummy coordinating module is created



In the above illustration, we have a dummy boss module "Produce Payroll" – which is named in a way that it indicate what the program is about. Having established the boss module, the afferent stream processes are moved to left most side of the next level of structure chart; the efferent stream process on the right most side and the central transform processes in the middle. Here, we moved a module to get valid timesheet (afferent process) to the left side (indicated in yellow). The two central transform processes are move in the middle (indicated in orange). By grouping the other two central transform processes with the respective efferent processes, we have created two modules (in blue) – essentially to print results, on the right side.

The main advantage of hierarchical (functional) arrangement of module is that it leads to flexibility in the software. For instance, if "Calculate Deduction" module is to select deduction rates from multiple rates, the module can be split into two in the next level – one to get the selection and another to calculate. Even after this change, the "Calculate Deduction" module would return the same value.

- **Refine the Structure Chart**

Expand the structure chart further by using the different levels of DFD. Factor down till you reach to modules that correspond to processes that access source / sink or data stores. Once this is ready, other features of the software like error handling, security, etc. has to be added. A module name should not be used for two different modules. If the same module is to be used in more than one place, it will be demoted down such that "fan in" can be done from the higher levels. Ideally, the name should sum up the activities done by the module and its sub-ordinates.

- **Verify Structure Chart vis-à-vis with DFD**

Because of the orientation towards the end-product, the software, the finer details of how data gets originated and stored (as appeared in DFD) is not explicit in Structure Chart. Hence DFD may still be needed along with Structure Chart to understand the data flow while creating low-level design.

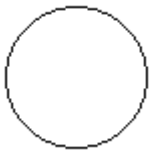
2. What is Data Flow Diagram?Write the function of each symbols used in DFD.Give an example in developing DFD Model for a system? [2016][2017]

Ans:-Data flow diagram (DFD) represents the flows of data between different processes in a business. It is a graphical technique that depicts information flow and the transforms that are applied as data move form input to output. It provides a simple, intuitive method for describing business processes without focusing on the details of computer systems. DFDs are attractive technique because they provide what users do rather than what computers do..

Representation of Components

DFDs only involve four symbols. They are:

- Process
- Data Object
- Data Store
- External entity



Process
Transform of incoming data flow(s) to outgoing flow(s).

Data Flow
Movement of data in the system.

Data Store
Data repositories for data that are not moving. It may be as simple as a buffer or a queue or a s sophisticated as a relational database.

External Entity
Sources of destinations outside the specified system boundary.

Relationship and Rules

Relationship

The DFD may be used for any level of data abstraction. DFD can be partitioned into levels. Each level has more information flow and data functional details than the previous level.

Highest level is Context Diagram. Some important points are:

- 1 bubble (process) represents the entire system.
- Data arrows show input and output.
- Data Stores NOT shown. They are within the system.

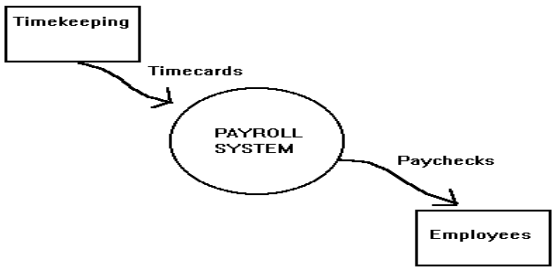


Diagram above is an example of Context Level DFD

Next Level is Level 0 DFD. Some important points are:

- Level 0 DFD must balance with the context diagram it describes.
- Input going into a process are different from outputs leaving the process.
- Data stores are first shown at this level.

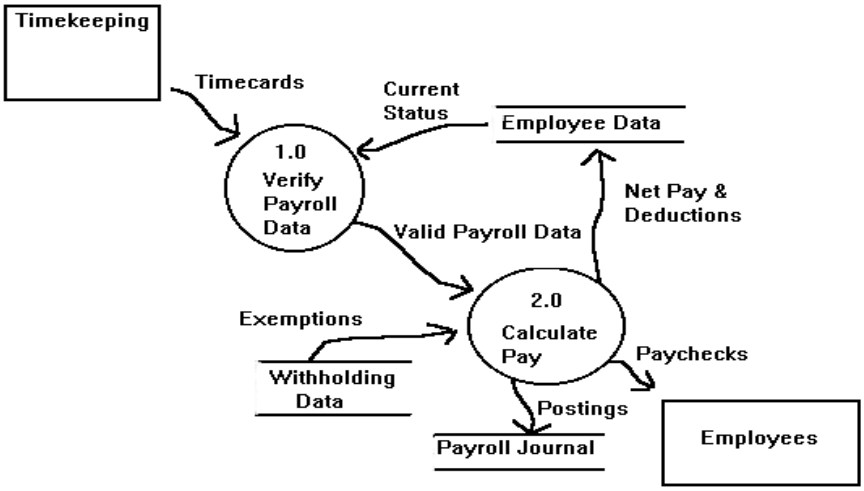


Diagram above show an example of Level 1 DFD

Next level is Level 1 DFD. Some important points are:

- Level 1 DFD must balance with the Level 0 it describes.
- Input going into a process are different from outputs leaving the process.
- Continue to show data stores.

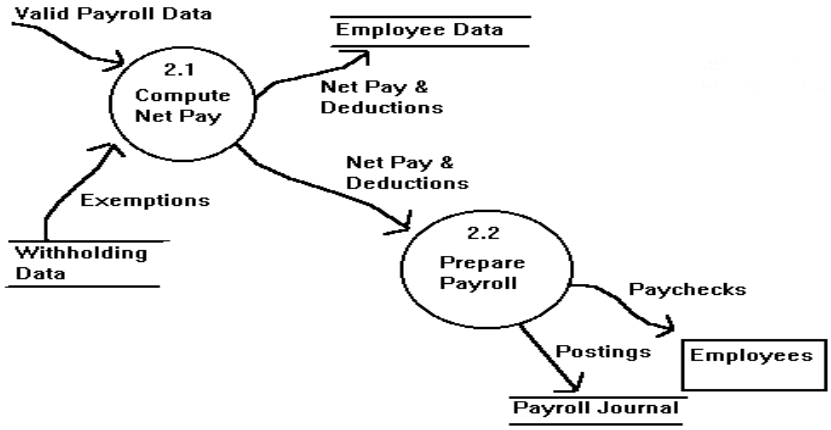


Diagram above show an example of Level 1 DFD

A DFD may look similar to a flow chart. However, there is a significant difference with the data flow diagram. The arrows in DFDs show that there is a flow of data between the two components and not that the component is sending the data that must be executed in the following component. A component in DFD may not continue execution when

sending data and during execution of the component receiving the data. The component sending data can send multiple sets of data along several connections. In fact, a DFD node can be a component that never ends.

Rules

- In DFDs, all arrows must be labeled.
- The information flow continuity, that is all the input and the output to each refinement, must maintain the same in order to be able to produce a consistent system.

3. Compare the characteristics of function oriented design and object oriented design?

1.FOD: The basic abstractions, which are given to the user, are real world functions.
OOD: The basic abstractions are not the real world functions but are the data abstraction where the real world entities are represented.

2.FOD: Functions are grouped together by which a higher level function is Page on obtained.an eg of this technique isSA/SD.
OOD: Functions are grouped together on the basis of the data they operate since the classes are associated with their methods.

3.FOD: In this appproach the state information is often represented in a centralized shared memory.
OOD: In this approach the state information is not represented in a centralized memory but is implemented or distributed among the objects of the system.

4.FOD approach is mainly used for computation sensitive application,
OOD: whereas OOD approach is mainly used for evolving system which mimicks a business process or business case.

5. In FOD – we decompose in function/procedure level
OOD: – we decompose in class level

6. FOD: TOp down Approach
OOD: Bottom up approach

7. **FOD**: It views system as Black Box that performs high level function and later decompose it detailed function so to be maaped to modules.
OOD: Object-oriented design is the discipline of defining the objects and their interactions to solve a problem that was identified and documented during object-oriented analysis.

4.What are different types of cohesion and coupling used in software design? [2015]

Ans:-**Cohesion:**

- With the help of cohesion the information hiding can be done.

- A cohesive subsystem performs only “one task” in software procedure with little interaction with other modules. In other cohesive subsystem performs only one thing.
- Different types of cohesion:

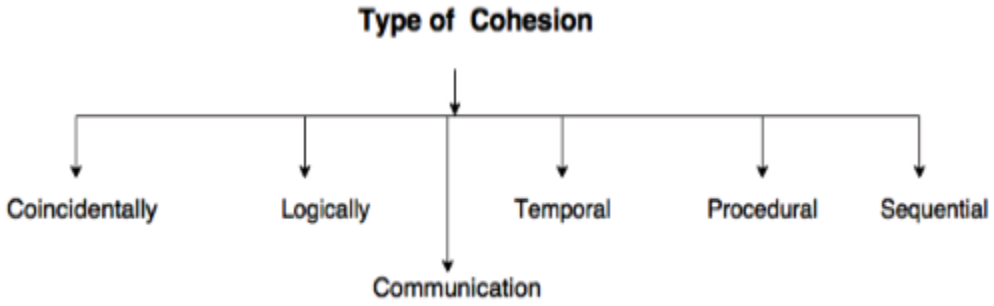


Fig. Different types of Cohesion

1. Coincidentally cohesive: The subsystem sin which the set of tasks are related with each other loosely then such subsystems are called coincidentally cohesive.

2. Logically cohesive: A subsystem that performs the tasks that are logically related with each other is called logically cohesive.

3. Temporal cohesive: The subsystem in which the tasks need to be executed in some specific time span is called temporal cohesive.

4. Procedural cohesive: When processing elements of a subsystem are related with one another and must be executed in some specific order, such subsystems is called Procedural cohesive.

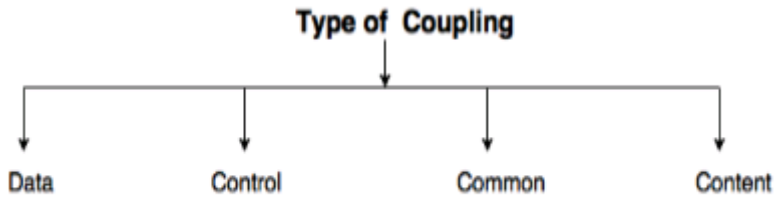
5. Communication cohesion: when the processing elements of a subsystem share the data then such subsystem is called communication cohesive.

6. Sequential cohesion: when the output of 1 subsystem is given as input for other subsystem is called Sequential cohesion.

Following fog shows layer cohesion.

Coupling:

- Coupling effectively represents how the subsystems can be connected with other subsystem or with the outside world.
- Coupling is a measure of interconnection among subsystems in a program structure.
- Coupling depends on the interface complexity between subsystems.
- The gaol is to strive for the possible coupling among the subsystems in software design.
- The property of good coupling is that it should reduce or avoid change impact and ripple effects.it should also reduce the cost in program changes, testing and maintenance.
- Various type of coupling:



2. **Fig. Different types of Coupling**

5. Data coupling: The data coupling is possible by parameter passing or data interaction.

6. Control coupling: The modules share related control data in control coupling.

7. Common coupling: In common coupling common data or global data is shared among the modules.

8. Content coupling: Content coupling occurs when one module makes use of data or control information maintained in another module.

[CHAPTER-5]

[7MARKS LONG QUESTION]

1. **What is user interface ?Describe the characteristics of a good user interface?[2017]**

Ans:-**The points to be kept in mind while designing good user interface are:**

1. **Clear and Simple** : A good user interface provides a clear understanding of what is happening behind the scenes or provides visibility to the functioning of the system. The whole purpose of user interface design is to enable the user to interact with your system by communicating meaning and function. Obviously, if the interface too complex to navigate, it might annoy the user and make him or her leave the page quickly and move on to some thing else. Make sure the interface is understandable and simple to navigate through.

2. **Creative but familiar** : When the users are familiar with something and know how it behaves, navigation becomes easier. In effect, the user expects to see what is familiar to him or her. It is good to identify things that your users are accustomed to and integrate them into your user interface. At the same time, users appreciate some thing creative, not so run-off-the- mill experience. But while being creative it should be kept in mind not to lose the familiarity component.
3. **Intuitive and consistent** : The controls and information must be laid out in an intuitive and consistent way for an interface to be easy to use and navigate. It's not good to drastically change the lay out to achieve the changing functionality the business may require from time to time. The design process should be based on the logic of usability -features that are the most frequently used should be the most prominent in the UI and controls should be made consistent so that users know how to repeat their action.
4. **Responsive** : If the interface fails to keep up with the demands of the user, this will significantly diminish their experience and can result in frustration, particularly when trying to perform basic tasks. Wherever possible, the interface should move swiftly in pace with the user. Being responsive means being fast. The interface, if not the program behind it, should work fast. Waiting for things to load might make the user frustrated.
5. **Maintainable** : A UI should have the capacity for and changes to be integrated without causing a conflict of interest. For instance, you may need to add an additional feature to the software, if your interface is so convoluted that there is no space to draw attention to this feature without compromising something else or appearing unaesthetic, then this signifies a flaw in design.

2. Define user interface .State and Explain the interface design activities?[2018]

Ans:-User interface design (UI) or user interface engineering is the design of user interfaces for machines and software, such as computers, home appliances, mobile devices, and other electronic devices, with the focus on maximizing usability and the user experience.

-> User interface is the front-end application view to which user interacts in order to use the software. User can manipulate and control the software as well as hardware by means of user interface. Today, user interface is found at almost every place where digital technology exists, right from computers, mobile phones, cars, music players, airplanes, ships etc.

User interface is part of software and is designed such a way that it is expected to provide the user insight of the software. UI provides fundamental platform for human-computer interaction.

UI can be graphical, text-based, audio-video based, depending upon the underlying hardware and software combination. UI can be hardware or software or a combination of both.

The software becomes more popular if its user interface is:

- Attractive
- Simple to use
- Responsive in short time
- Clear to understand
- Consistent on all interfacing screens

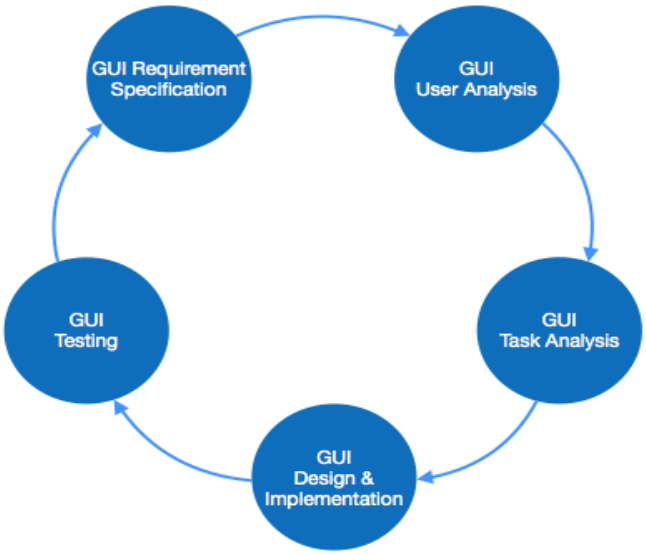
UI is broadly divided into two categories:

- Command Line Interface
- Graphical User Interface

User Interface Design Activities

There are a number of activities performed for designing user interface. The process of GUI design and implementation is alike SDLC. Any model can be used for GUI implementation among Waterfall, Iterative or Spiral Model.

A model used for GUI design and development should fulfill these GUI specific steps.



- **GUI Requirement Gathering** - The designers may like to have list of all functional and non-functional requirements of GUI. This can be taken from user and their existing software solution.
- **User Analysis** - The designer studies who is going to use the software GUI. The target audience matters as the design details change according to the knowledge and competency level of the user. If user is technical savvy, advanced and complex GUI can be incorporated. For a novice user, more information is included on how-to of software.
- **Task Analysis** - Designers have to analyze what task is to be done by the software solution. Here in GUI, it does not matter how it will be done. Tasks can be represented in hierarchical manner taking one major task and dividing it further into smaller sub-tasks. Tasks provide goals for GUI

presentation. Flow of information among sub-tasks determines the flow of GUI contents in the software.

- **GUI Design & implementation** - Designers after having information about requirements, tasks and user environment, design the GUI and implements into code and embed the GUI with working or dummy software in the background. It is then self-tested by the developers.
- **Testing** - GUI testing can be done in various ways. Organization can have in-house inspection, direct involvement of users and release of beta version are few of them. Testing may include usability, compatibility, user acceptance etc.

[5MARKS QUESTION AND ANSWER]

1. Discuss the several types of designing approaches used in software engineering?[2018]

- **Ans:-Requirements Analysis**
  - Extracting the requirements of a desired software product is the first task in creating it. While customers probably believe they know what the software is to do, it may require skill and experience in software engineering to recognize incomplete, ambiguous or contradictory requirements.
- **Specification**
  - Specification is the task of precisely describing the software to be written, in a mathematically rigorous way. In practice, most successful specifications are written to understand and fine-tune applications that were already well-developed, although safety-critical software systems are often carefully specified prior to application development. Specifications are most important for external interfaces that must remain stable.
- **Software architecture**
  - The architecture of a software system refers to an abstract representation of that system. Architecture is concerned with making sure the software system will meet the requirements of the product, as well as ensuring that future requirements can be addressed.
- **Implementation**
  - Reducing a design to code may be the most obvious part of the software engineering job, but it is not necessarily the largest portion.
- **Testing**
  - Testing of parts of software, especially where code by two different engineers must work together, falls to the software engineer.
- **Documentation**

- An important task is documenting the internal design of software for the purpose of future maintenance and enhancement.

- **Training and Support**

- A large percentage of software projects fail because the developers fail to realize that it doesn't matter how much time and planning a development team puts into creating software if nobody in an organization ends up using it. People are occasionally resistant to change and avoid venturing into an unfamiliar area, so as a part of the deployment phase, its very important to have training classes for the most enthusiastic software users (build excitement and confidence), shifting the training towards the neutral users intermixed with the avid supporters, and finally incorporate the rest of the organization into adopting the new software. Users will have lots of questions and software problems which leads to the next phase of software.

- **Maintenance**

- Maintaining and enhancing software to cope with newly discovered problems or new requirements can take far more time than the initial development of the software. Not only may it be necessary to add code that does not fit the original design but just determining how software works at some point after it is completed may require significant effort by a software engineer. About 60% of all software engineering work is maintenance, but this statistic can be misleading. A small part of that is fixing bugs. Most maintenance is extending systems to do new things, which in many ways can be considered new work.

[2marks question answer]

1. **What is software configuration management?**

Ans:-Configuration Management helps organizations to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle. It is abbreviated as the SCM process. It aims to control cost and work effort involved in making changes to the software system. The primary goal is to increase productivity with minimal mistakes.

2. **What is software documentation?**

Ans:- All software documentation can be divided into **two main categories:**

- **Product documentation**
- **Process documentation**

Product documentation describes the product that is being developed and provides instructions on how to perform various tasks with it. Product documentation can be broken down into:

- System documentation and

- User documentation

System documentation represents documents that describe the system itself and its parts. It includes requirements documents, design decisions, architecture descriptions, program source code, and help guides.
User documentation covers manuals that are mainly prepared for end-users of the product and system administrators. User documentation includes tutorials, user guides, troubleshooting manuals, installation, and reference manuals.
Process documentation represents all documents produced during development and maintenance that describe… well, process. The common examples of process-related documents are standards, project documentation, such as project plans, test schedules, reports, meeting notes, or even business correspondence.

3.**Define CASE**?
Ans:-Computer-aided **software engineering** (**CASE**) is the domain of **software** tools used to design and implement applications. ... **CASE software** is often associated with methods for the development of information systems together with automated tools that can be used in the **software** development process.

[CHAPTER-6]

[7MARKS LONG QUESTION]

1. **What is performance testing is carried out? Discuss some performance test? [2018]**

Ans:-Performance testing is the process of determining the speed or effectiveness of a computer, network, software program or device. This process can involve quantitative tests done in a lab, such as measuring the response time or the number of MIPS (millions of instructions per second) at which a system functions. Qualitative attributes such as reliability, scalability and interoperability may also be evaluated. Performance testing is often done in conjunction with stress testing.

Types of Performance Testing

- **Load testing -** checks the application's ability to perform under anticipated user loads. The objective is to identify performance bottlenecks before the software application goes live.

- **Stress testing -** involves testing an application under extreme workloads to see how it handles high traffic or data processing. The objective is to identify breaking point of an application.
- **Endurance testing -** is done to make sure the software can handle the expected load over a long period of time.
- **Spike testing -** tests the software's reaction to sudden large spikes in the load generated by users.
- **Volume testing** - Under Volume Testing large no. of. Data is populated in database and the overall software system's behavior is monitored. The objective is to check software application's performance under varying database volumes.
- **Scalability testing** - The objective of scalability testing is to determine the software application's effectiveness in "scaling up" to support an increase in user load. It helps plan capacity addition to your software system.
- **Performance Testing Process**
- **Identify your testing environment -** Know your physical test environment, production environment and what testing tools are available. Understand details of the hardware, software and network configurations used during testing before you begin the testing process. It will help testers create more efficient tests. It will also help identify possible challenges that testers may encounter during the performance testing procedures.
- **Identify the performance acceptance criteria -** This includes goals and constraints for throughput, response times and resource allocation. It is also necessary to identify project success criteria outside of these goals and constraints. Testers should be empowered to set performance criteria and goals because often the project specifications will not include a wide enough variety of performance benchmarks. Sometimes there may be none at all. When possible finding a similar application to compare to is a good way to set performance goals.
- **Plan & design performance tests -** Determine how usage is likely to vary amongst end users and identify key scenarios to test for all possible use cases. It is necessary to simulate a variety of end users, plan performance test data and outline what metrics will be gathered.
- **Configuring the test environment -** Pre

2. What do you mean by coding?How code walk through is conducted.explain? [2016][2017]

It also refers to methods for the **development** of information systems together with automated tools that **can** be used in the **software development** process. ... The CASE functions include analysis, **design**, and **programming**.

- **Static Testing v/s Dynamic Testing**

- *Static testing* is done basically to test the software work products , requirement specifications, test plan , user manual etc. They are not executed, but tested with the set of some tools and processes. It provides a powerful way to improve the quality and productivity of software development.

Static Testing v/s Dynamic Testing

Dynamic Testing is basically when execution is done on the software code as a technique to detect defects and to determine quality attributes of the code. With dynamic testing methods,
software is executed using a set of inputs and its output is then compared to the the expected results.

- **Static Review and its advantages**

Static Review provides a powerful way to improve the quality and productivity of software development to recognize and fix their own defects early in the software development process.
Nowadays, all software organizations are conducting reviews in all major aspects of their work including requirements, design, implementation, testing, and maintenance.

Advantages of Static Reviews:-

1. Types of defects that can be found during static testing are: deviations from standards, missing requirements, design defects, non-maintainable code and inconsistent interface specifications.

2. Since static testing can start early in the life cycle, early feedback on quality issues can be established, e.g. an early validation of user requirements and not just late in the life cycle during
acceptance testing.

3. By detecting defects at an early stage, rework costs are relatively low and thus a relatively cheap improvement of the quality of software products can be achieved.

4. The feedback and suggestions document from the static testing process allows for process improvement, which supports the avoidance of similar errors being made in the future.

- **Roles and Responsibilities in a Review**

There are various roles and responsibilities defined for a review process. Within a review team, four types of participants can be distinguished: moderator, author, scribe.

3..**Explain different types of white box testing used to design test cases for testing the software? [2015][2016][2017]**

Ans:-If we go by the definition, “White box testing” (also known as clear, glass box or structural testing) is a testing technique which evaluates the code and the internal structure of a program.

White box testing involves looking at the structure of the code. When you know the internal structure of a product, tests can be conducted to ensure that the internal operations performed according to the specification. And all internal components have been adequately exercised.

White Box Testing is coverage of the specification in the code:
1. Code coverage
2. Segment coverage: Ensure that each code statement is executed once.
3. Branch Coverage or Node Testing: Coverage of each code branch in from all possible was.
4. Compound Condition Coverage: For multiple conditions test each condition with multiple paths and combination of the different path to reach that condition.
5. Basis Path Testing: Each independent path in the code is taken for testing.
6. Data Flow Testing (DFT): In this approach you track the specific variables through each possible calculation, thus defining the set of intermediate paths through the code.DFT tends to reflect dependencies but it is mainly through sequences of data manipulation. In short, each data variable is tracked and its use is verified. This approach tends to uncover bugs like variables used but not initialize, or declared but not used, and so on.
7. Path Testing: Path testing is where all possible paths through the code are defined and covered. It’s a time-consuming task.
8. Loop Testing: These strategies relate to testing single loops, concatenated loops, and nested loops. Independent and dependent code loops and values are tested by this approach.
Why we perform WBT?
To ensure:

- That all independent paths within a module have been exercised at least once.
- All logical decisions verified on their true and false values.
- All loops executed at their boundaries and within their operational bounds internal data structures validity.

To discover the following types of bugs:

- Logical error tend to creep into our work when we design and implement functions, conditions or controls that are out of the program
- The design errors due to difference between logical flow of the program and the actual implementation
- Typographical errors and syntax checking

Does this testing requires detailed programming skills?
We need to write test cases that ensure the complete coverage of the program logic.
For this we need to know the program well i.e. We should know the specification and the code to be tested. Knowledge of programming languages and logic is required for this type of testing.

4 . What are the various debugging approaches and guidelines used in s/w development?[2017]

- On successful culmination of software testing, debugging is performed. Debugging is defined as a process of analyzing and removing the error. It is considered necessary in most of the newly developed software or hardware and in commercial products/ personal application programs. For complex products, debugging is done at all the levels of the testing.
- Debugging is considered to be a complex and time-consuming process since it attempts to remove errors at all the levels of testing. To perform debugging, debugger (debugging tool) is used to reproduce the conditions in which failure occurred, examine the program state, and locate the cause. With the help of debugger, programmers trace the program execution step by step (evaluating the value of variables) and halt the execution wherever required to reset the program variables. Note that some programming language packages include a debugger for checking the code for errors while it is being written.

The Debugging Process

During debugging, errors are encountered that range from less damaging (like input of an incorrect function) to catastrophic (like system failure, which lead to economic or physical damage). Note that with the increase in number of errors, the amount of effort to find their causes also increases.

Once errors are identified in a software system, to debug the problem, a number of steps are followed, which are listed below.

1. **Defect confirmation/identification:** A problem is identified in a system and a defect report is created. A software engineer maintains and analyzes this error report and finds solutions to the following questions.
1. Does a .defect exist in the system?
2. Can the defect be reproduced?
3. What is the expected/desired behavior of the system?
4. What is the actual behavior?
2. **Defect analysis:** If the defect is genuine, the next step is to understand the root cause of the problem. Generally, engineers debug by starting a debugging tool (debugger) and they try to understand the root cause of the problem by following a step-by-step execution of the program.

3. **Defect resolution:** Once the root cause of a problem is identified, the error can be resolved by making an appropriate change to the system by fixing the problem.

   When the debugging process ends, the software is retested to ensure that no errors are left undetected. Moreover, it checks that no new errors are introduced in the software while making some changes to it during the debugging process.

Debugging Strategies

As debugging is a difficult and time-consuming task, it is essential to develop a proper debugging strategy. This strategy helps in performing the process of debugging easily and efficiently. The commonly-used debugging strategies are debugging by brute force, induction strategy, deduction strategy, backtracking strategy, and debugging by testing.

[CHAPTER-7]

1.Q:-Importance, Requirement and Procedure to Gain ISO 9000 Certification for Software Industry?[2018]

Ans:- ISO (International Standards Organization) is a consortium of 63 countries established to formulate and foster standardisation. ISO published its 9000 series of standards in 1987.

The ISO 9000 standard specifies the guidelines for maintaining a quality system. ISO 9000 specifies a set of guidelines for repeatable and high quality product development. ISO 9000 is a series of three standards: ISO 9001, ISO 9002, and ISO 9003.

ISO 9001: This standard applies to the organisations engaged in design,development, production, and servicing of goods.

This standard is applicable to most software development organisations.

ISO 9002: This standard applies to those organisations which do not design products but are only involved in production. Examples include steel and car \ manufacturing industries.

ISO 9003: This standard applies to organisations involved only in installation and testing of the products.

Requirement of ISO 9000 Certification

- Confidence of customers in an organisation increases when the organisation qualifies for ISO 9001 certification.
- ISO 9000 requires a well-documented software production process.
- ISO 9000 makes the development process focused, efficient, and cost-effective.
- ISO 9000 certification points out the weak points of an organization and recommends remedial action.
- ISO 9000 sets the basic framework for the development of an optimal process.

Procedure to gain ISO 9000 Certification

An organisation intending to obtain ISO 9000 certification applies to a ISO 9000 registrar for registration. The ISO 9000 registration process consists of the following stages:

- Application: Once an organisation decides to go for ISO 9000 certification, it applies to a register for registration.
- Pre-assessment: During this stage, the registrar makes a rough assessment of the organisation.
- Document Review and Adequacy of Audit : During this stage, the registrar reviews the documents submitted by the organisation and makes suggestions for possible improvements.
- Compliance audit: During this stage, the registrar checks whether the suggestions made by it during review have been complied with by the organisation or not.
- Continued Surveillance: The registrar continues to monitorthe organisation, though periodically.