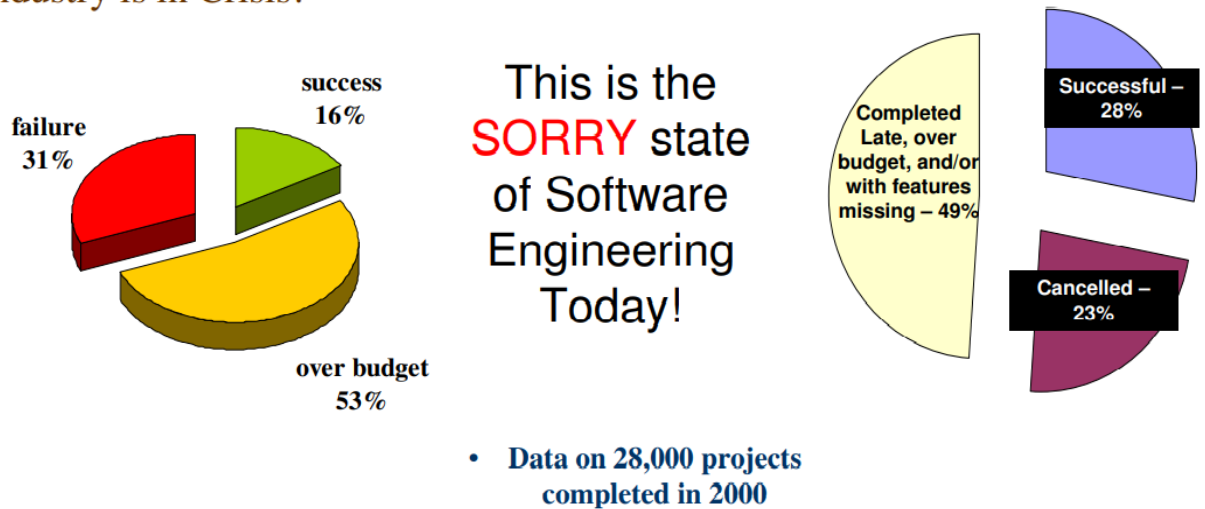


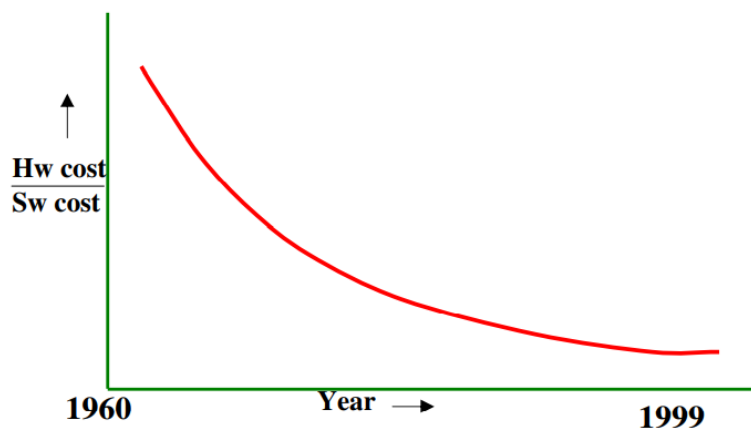
Q1. What is Software Crisis?

Software industry is in Crisis!

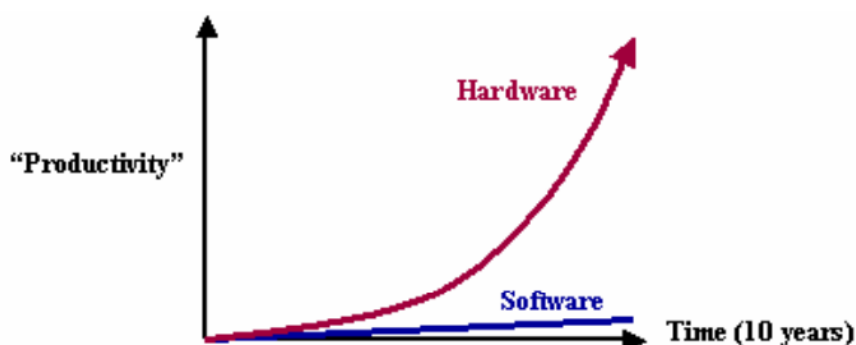


As per the IBM report, “31% of the project get cancelled before they are completed, 53% overrun their cost estimates by an average of 189% and for every 100 projects, there are 94 restarts”.

Q2. Relative Cost of Hardware and Software.



- Moore’s law: processor speed/memory capacity doubles every two years



Q3. What are factors contributing to software crisis?

- Larger problems,
- Lack of adequate training in software engineering,
- Increasing skill shortage,
- Low productivity improvements.

Q4. Some Software Failures.

Ariane 5

It took the European Space Agency **10 years and \$7 billion** to produce Ariane 5, a giant rocket capable of hurling a pair of three-ton satellites into orbit with each launch and intended to give Europe overwhelming supremacy in the commercial space business.

The rocket was destroyed after 39 seconds of its launch, at an altitude of two and a half miles along with its payload of four expensive and uninsured scientific satellites.



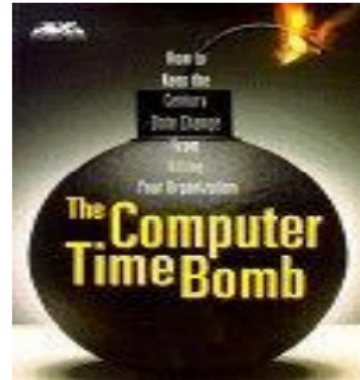
When the guidance system's own computer tried to convert one piece of data the sideways velocity of the rocket from a 64 bit format to a 16 bit format; the number was too big, and an overflow error resulted after 36.7 seconds. When the guidance system shutdown, it passed control to an identical, redundant unit, which was there to provide backup in case of just such a failure. Unfortunately, the second unit, which had failed in the identical manner a few milliseconds before.



Y2K problem:

It was simply the ignorance about the adequacy or otherwise of using only last two digits of the year.

The 4-digit date format, like 1964, was shortened to 2-digit format, like 64.



The Patriot Missile

- o First time used in Gulf war
- o Used as a defense from Iraqi Scud missiles
- o Failed several times including one that killed 28 US soldiers in Dhahran, Saudi Arabia

Reasons:

A small timing error in the system's clock accumulated to the point that after 14 hours, the tracking system was no longer accurate. In the Dhahran attack, the system had been operating for more than 100 hours.



Q5. “No Silver Bullet”

The hardware cost continues to decline drastically.

However, there are desperate cries for a silver bullet something to make software costs drop as rapidly as computer hardware costs do.

But as we look to the horizon of a decade, we see no silver bullet. There is no single development, either in technology or in management technique, that by itself promises even one order of magnitude improvement in productivity, in reliability and in simplicity.

The hard part of building software is the specification, design and testing of this conceptual construct, not the labour of representing it and testing the correctness of representation.

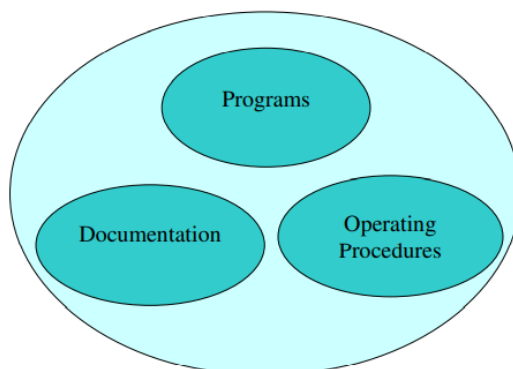
We still make syntax errors, to be sure, but they are trivial as compared to the conceptual errors (logic errors) in most systems. That is why, building software is always hard and there is inherently no silver bullet.

While there is no royal road, there is a path forward.

Is reusability (and open source) the new silver bullet?

The blame for software bugs belongs to: • Software companies • Software developers • Legal system • Universities

Q6. What is software?

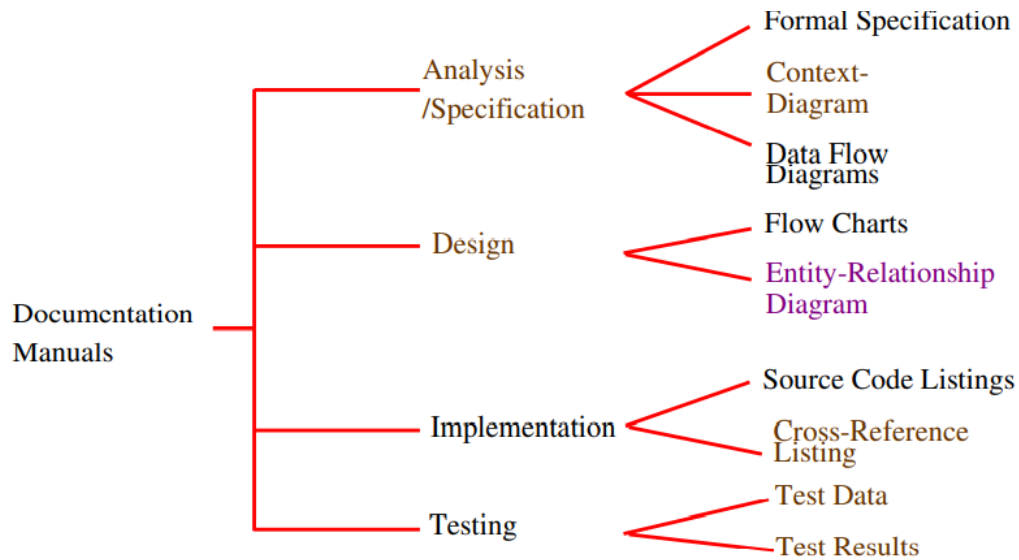


Software=Program+Documentation+Operating Procedures

Components of software

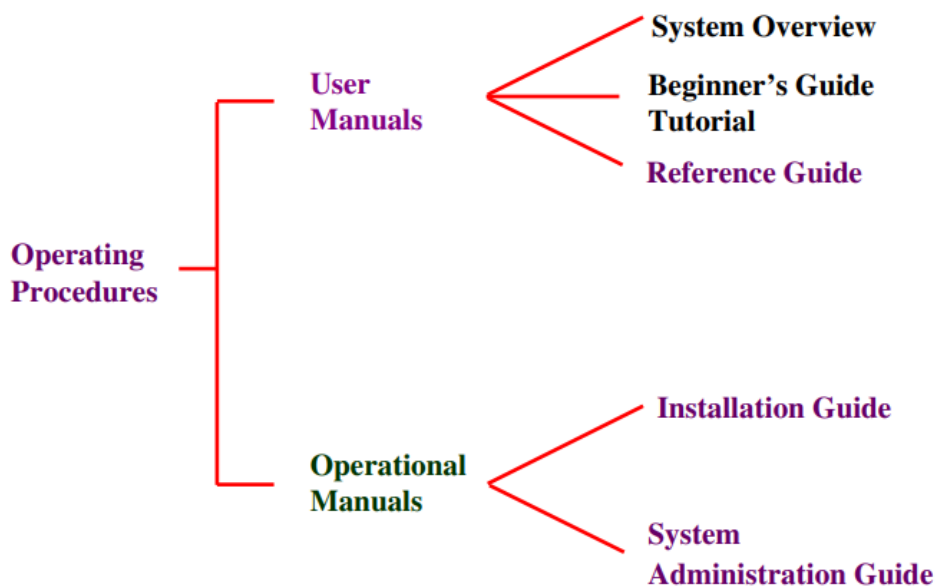


Q6. Documentation of software.



List of documentation manuals

Q7. Operating Procedures.



List of operating procedure manuals.

Q8. Why Software Product developed.

- Software products may be developed for a particular customer or may be developed for a general market Software Product
- Software products may be
 - Generic – developed to be sold to a range of different customers
 - Bespoke (custom) - developed for a single customer according to their specification

Q9. What is software Engineering?

Software engineering is an engineering discipline which is concerned with all aspects of software production

Software engineers should

- adopt a systematic and organised approach to their work
- use appropriate tools and techniques depending on
 - the problem to be solved,
 - the development constraints and
- use the resources available



At the first conference on software engineering in 1968, Fritz Bauer defined software engineering as “The establishment and use of sound engineering principles in order to obtain economically developed software that is reliable and works efficiently on real machines”.

Stephen Schach defined the same as “A discipline whose aim is the production of quality software, software that is delivered on time, within budget, and that satisfies its requirements”.

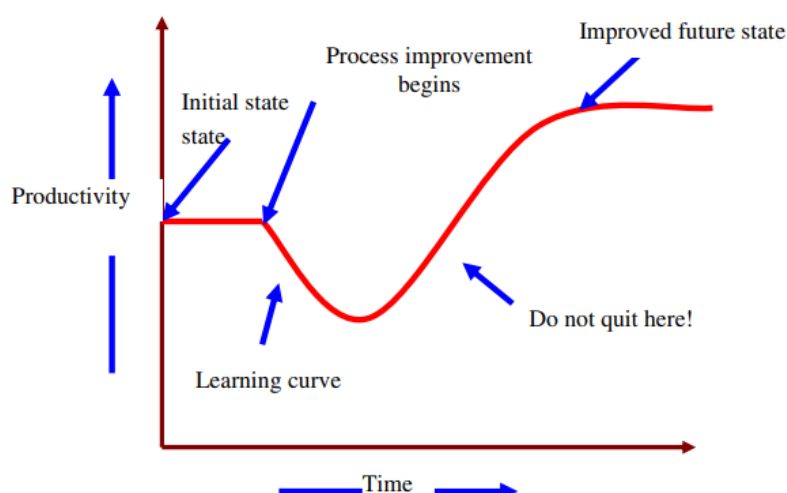
Q10. What is Software Process?

The software process is the way in which we produce software.

A **Software Process** refers to a structured set of activities, methods, and practices used to develop and maintain software systems. It encompasses all phases of software development, from initial conception through to delivery and ongoing maintenance.

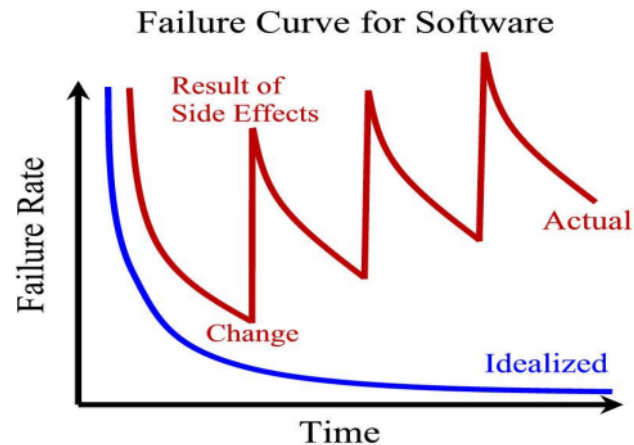
Q11. Why is it difficult to improve software process?

- Not enough time
- Lack of knowledge
- Wrong motivations
- Insufficient commitment



Q12. What are characteristics of a Software?

- Software does not wear out
- Software is not manufactured
- Reusability of component
- Software is flexible

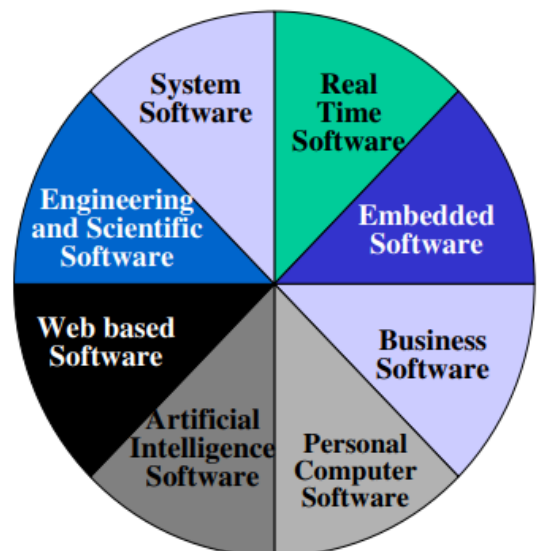


Q13. Changing nature of Software.

Trend has emerged to provide source code to the customer and organizations.

Software where source codes are available are known as open source software.

Examples Open source software: LINUX, MySQL, PHP, Open office, Apache webserver etc.



Q14. Deliverables and Milestones.

Different deliverables are generated during software development. The examples are source code, user manuals, operating procedure manuals etc.

The milestones are the events that are used to ascertain the status of the project. Finalization of specification is a milestone. Completion of design documentation is another milestone. The milestones are essential for project planning and management.

Q15. Product and Process

Product: What is delivered to the customer, is called a product. It may include source code, specification document, manuals, documentation etc. Basically, it is nothing but a set of deliverables only.

Process: Process is the way in which we produce software. It is the collection of activities that leads to (a part of) a product. An efficient process is required to produce good quality products. If the process is weak, the end product will undoubtedly suffer, but an obsessive over reliance on process is also dangerous.

Q16. Productivity and Effort.

Productivity is defined as the rate of output, or production per unit of effort, i.e. the output achieved with regard to the time taken but irrespective of the cost incurred.

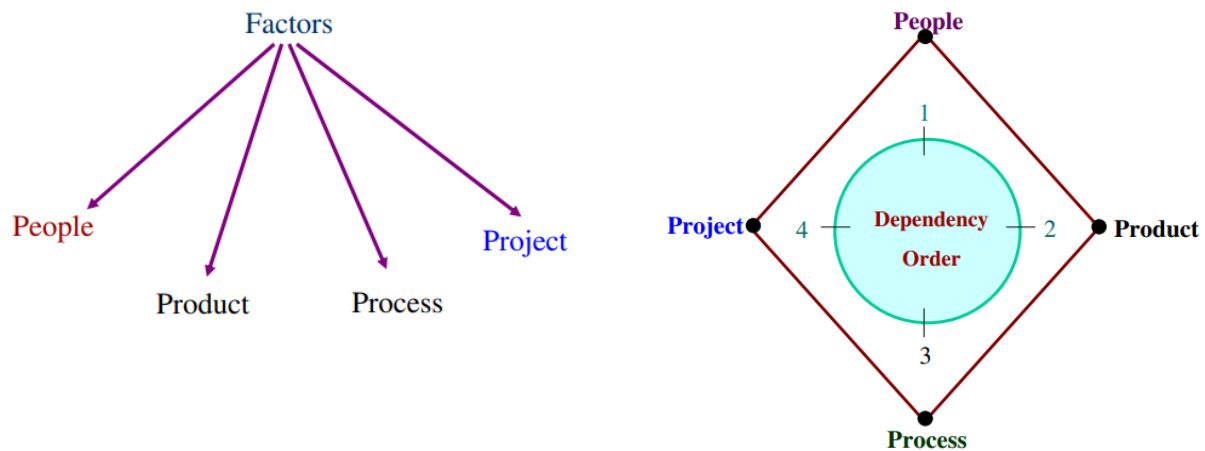
Hence most appropriate unit of effort is Person Months (PMs), meaning thereby number of persons involved for specified months. So, productivity may be measured as LOC/PM (lines of code produced/person month)

Q17. Software Components.

“An independently deliverable piece of functionality providing access to its services through interfaces”.

“A component represents a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces”.

Q18. Role of Management in Software Development.



Q18. Software Life Cycle Models

The goal of Software Engineering is to provide models and processes that lead to the production of well-documented maintainable software in a manner that is predictable.

“The period of time that starts when a software product is conceived and ends when the product is no longer available for use. The software life cycle typically includes a requirement phase, design phase, implementation phase, test phase, installation and check out phase, operation and maintenance phase, and sometimes retirement phase”.

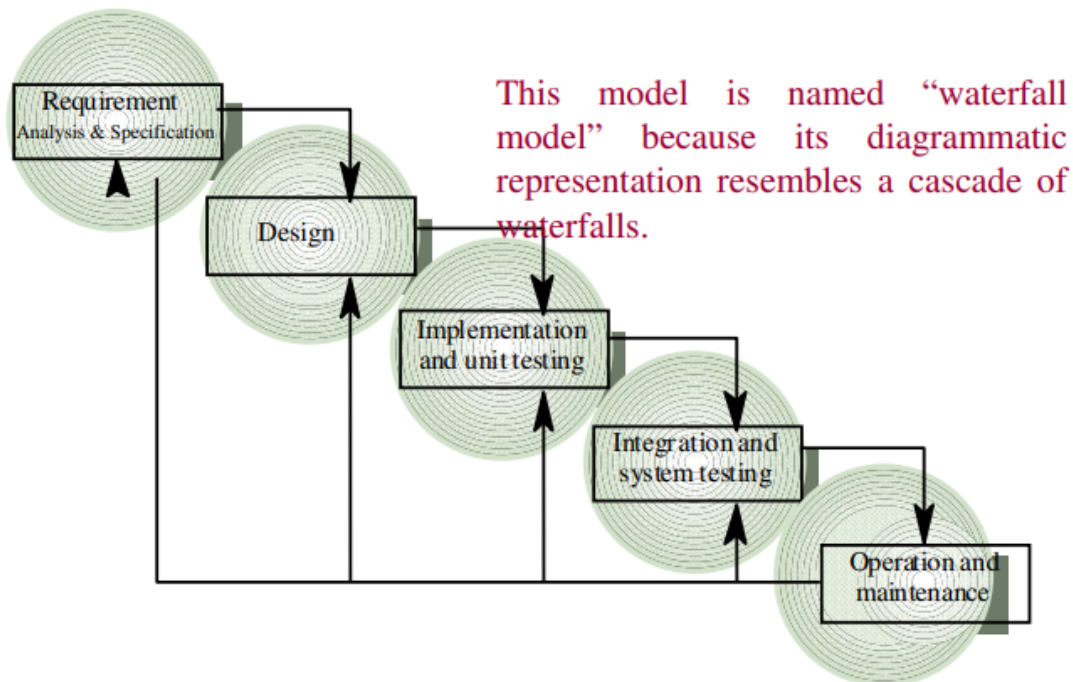
Q19. Waterfall Model

This model is easy to understand and reinforces the notion of “define before design” and “design before code”. The model expects complete & accurate requirements early in the process, which is unrealistic.

Problems of waterfall model

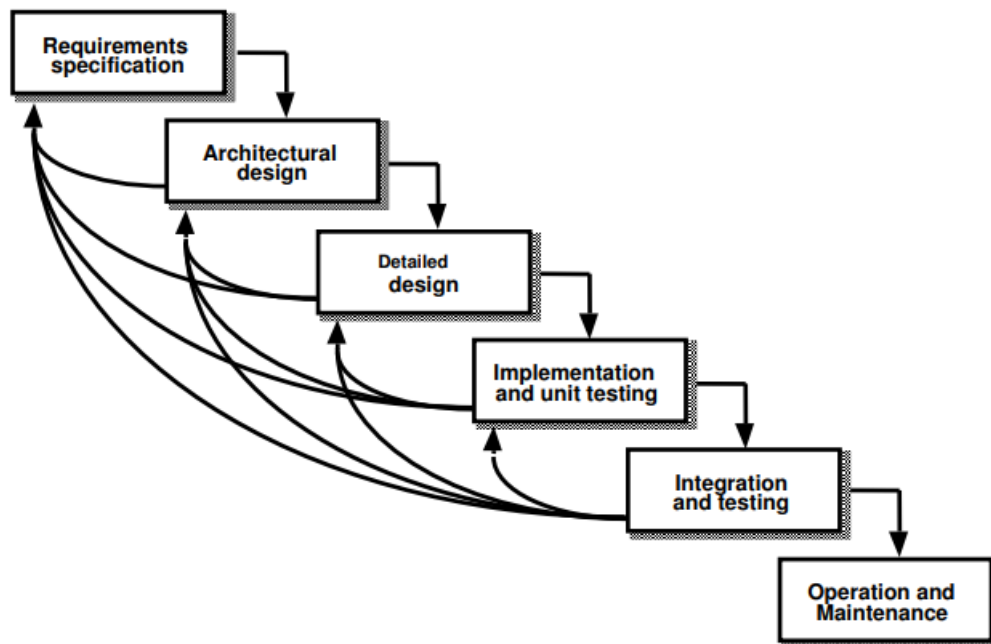
- i. It is difficult to define all requirements at the beginning of a project
- ii. This model is not suitable for accommodating any change
- iii. A working version of the system is not seen until late in the project's life
- iv. It does not scale up well to large projects.
- v. Real projects are rarely sequential

Waterfall Model



Q20. Iterative Enhancement Model

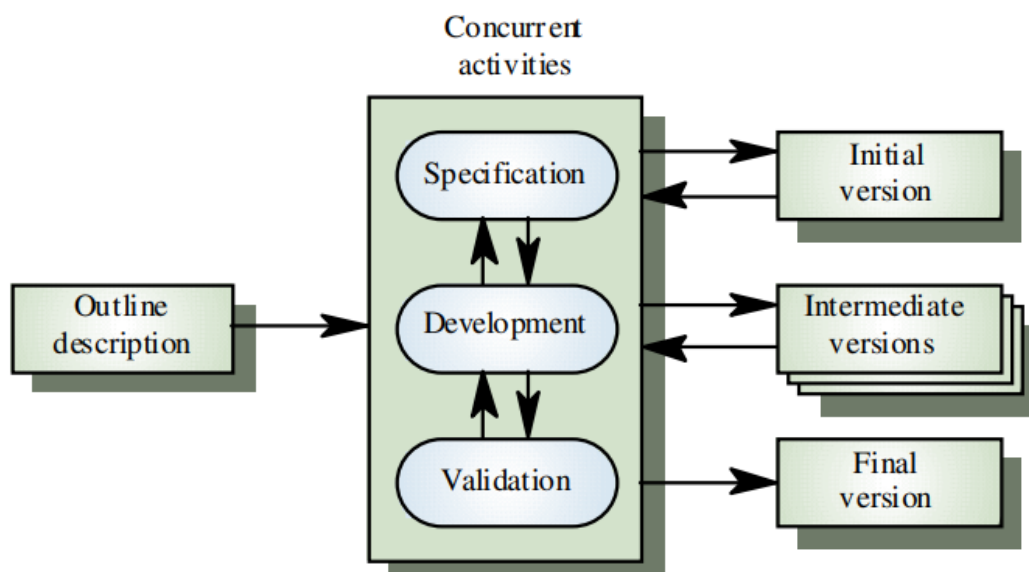
This model has the same phases as the waterfall model, but with fewer restrictions. Generally the phases occur in the same order as in the waterfall model, but they may be conducted in several cycles. Useable product is released at the end of the each cycle, with each release providing additional functionality



Q 21. Evolutionary Process Model

Evolutionary process model resembles iterative enhancement model. The same phases as defined for the waterfall model occur here in a cyclical fashion. This model differs from iterative enhancement model in the sense that this does not require an useable product at the end of each cycle. In evolutionary development, requirements are implemented by category rather than by priority.

This model is useful for projects using new technology that is not well understood. This is also used for complex projects where all functionality must be delivered at one time, but the requirements are unstable or not well understood at the beginning

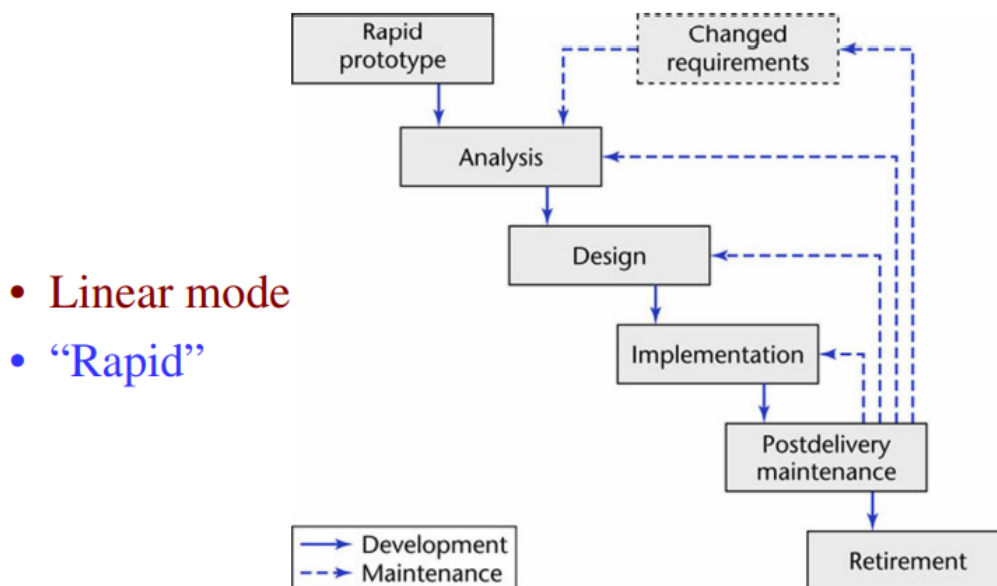


Q22. Prototyping Model

The prototype may be a usable program but is not suitable as the final software product.

The code for the prototype is thrown away. However experience gathered helps in developing the actual system.

The development of a prototype might involve extra cost, but overall cost might turnout to be lower than that of an equivalent system developed using the waterfall model.



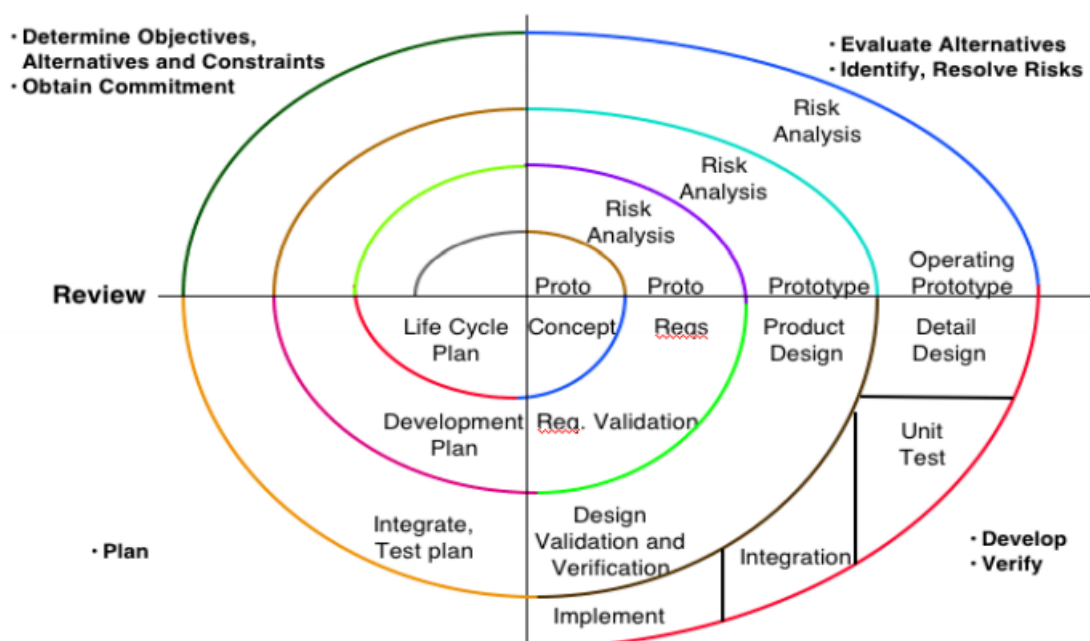
Q23. Spiral Model

Models do not deal with uncertainty which is inherent to software projects.

Important software projects have failed because project risks were neglected & nobody was prepared when something unforeseen happened.

Barry Boehm recognized this and tried to incorporate the “project risk” factor into a life cycle model.

The result is the spiral model, which was presented in 1986



The radial dimension of the model represents the cumulative costs. Each path around the spiral is indicative of increased costs. The angular dimension represents the progress made in completing each cycle. Each loop of the spiral from X-axis clockwise through 360 represents one phase. One phase is split roughly into four sectors of major activities.

- Planning: Determination of objectives, alternatives & constraints.
- Risk Analysis: Analyze alternatives and attempts to identify and resolve the risks involved.
- Development: Product development and testing product.
- Assessment: Customer evaluation

An important feature of the spiral model is that each phase is completed with a review by the people concerned with the project (designers and programmers)

The advantage of this model is the wide range of options to accommodate the good features of other life cycle models.

It becomes equivalent to another life cycle model in appropriate situations.

The spiral model has some difficulties that need to be resolved before it can be a universally applied life cycle model. These difficulties include lack of explicit process guidance in determining objectives, constraints, alternatives; relying on risk assessment expertise; and provides more flexibility than required for many applications.

Q24. Software Requirements Specification (SRS)

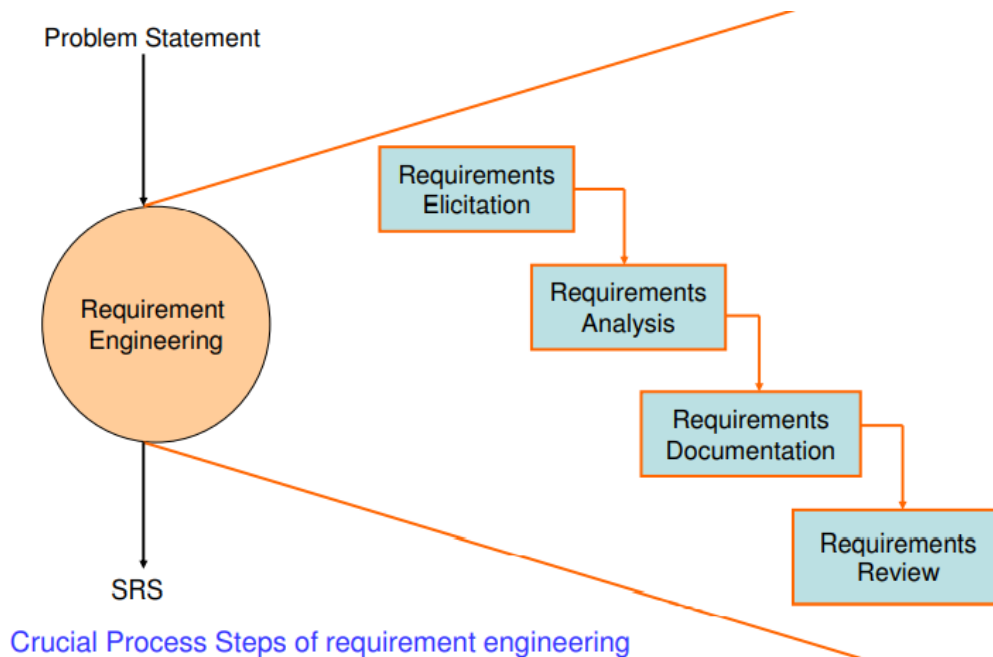
The **Software Requirements Specification (SRS)** is a comprehensive document that outlines the complete software requirements for a project. It serves as a formal agreement between stakeholders and the development team, detailing what the software should do and the constraints under which it must operate.

For example, it will describe the software's ability to handle user input, process data, and deliver appropriate results. It will also address user needs such as speed, security, and user-friendliness, but won't specify how these will be technically achieved. The document ensures clarity between stakeholders and developers, keeping the focus on "what" the system does, not "how."

Requirements describe

What not How

Q25. Requirement Engineering



Requirement Engineering is the disciplined application of proven principles, methods, tools, and notations to describe a proposed system's intended behaviour and its associated constraints.

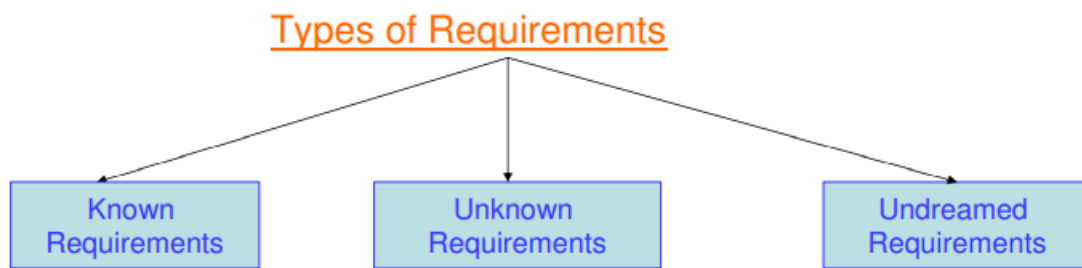
SRS may act as a contract between developer and customer.

State of practice Requirements are difficult to uncover

- Requirements change
- Over reliance on CASE Tools
- Tight project Schedule
- Communication barriers
- Market driven software development
- Lack of resources

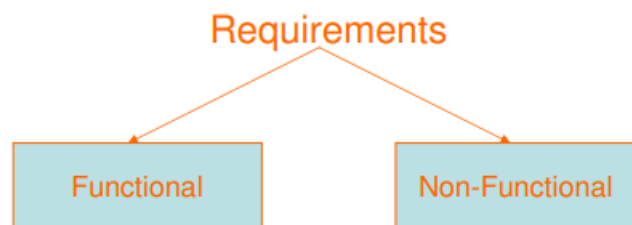
A University wish to develop a software system for the student result management of its M.Tech. Programme. A problem statement is to be prepared for the software development company. The problem statement may give an overview of the existing system and broad expectations from the new software system.

Q27. Types of Requirements



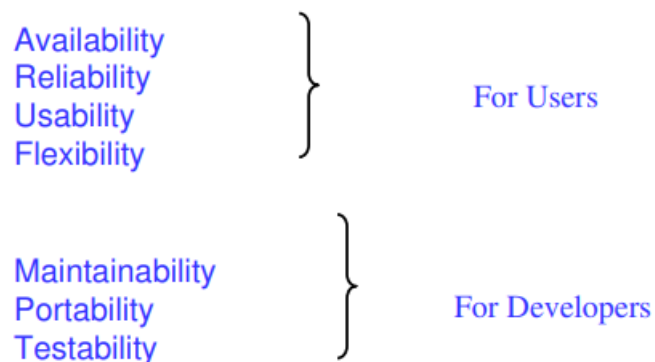
Stakeholder: Anyone who should have some direct or indirect influence on the system requirements.

--- User
--- Affected persons



Functional requirements describe what the software has to do. They are often called product features.

Non Functional requirements are mostly quality requirements. That stipulate how well the software does, what it has to do.



User and system requirements

- User requirement are written for the users and include functional and non functional requirement.
- System requirement are derived from user requirement.
- The user system requirements are the parts of software requirement and specification (SRS) document.

Interface Specification

- Important for the customers.

TYPES OF INTERFACES

- Procedural interfaces (also called Application Programming Interfaces (APIs)).
- Data structures
- Representation of data

Q28. Feasibility Study

A feasibility study checks if a project is practical and worth pursuing. It looks at key areas to decide if the project can be done successfully within time, budget, and resources.

Key parts include:

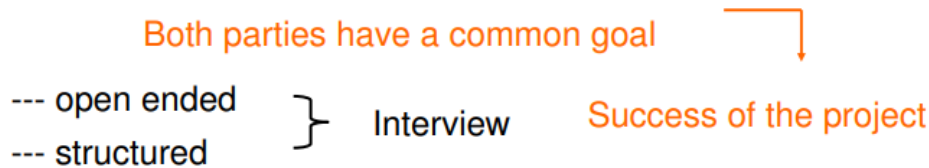
1. **Technical Feasibility** – Can the needed technology and tools be developed or used?
2. **Economic Feasibility** – Is the project cost-effective, with benefits greater than costs?
3. **Operational Feasibility** – Will the system work well with current processes, and will users accept it?
4. **Legal Feasibility** – Does the project follow laws and regulations?
5. **Schedule Feasibility** – Can the project be completed on time?

This helps in deciding whether to move forward with a project.

Q29. Requirements Elicitation

Requirements elicitation is the process of gathering, discovering, and understanding the needs and expectations of stakeholders for a system or product. Techniques for requirements elicitation:

1. Interviews



Selection of stakeholder

1. Entry level personnel
2. Middle level stakeholder
3. Managers
4. Users of the software (Most important)

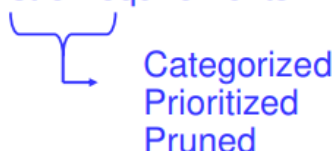
2. Brainstorming Sessions

It is a group technique

group discussions



Prepare long list of requirements



'Idea is to generate views ,not to vet them.

Groups

1. Users
2. Middle Level managers
3. Total Stakeholders

3. Facilitated Application specification Techniques (FAST)

-- Similar to brainstorming sessions. -- Team oriented approach -- Creation of joint team of customers and developers.

Guidelines

1. Arrange a meeting at a neutral site.
2. Establish rules for participation.
3. Informal agenda to encourage free flow of ideas.
4. Appoint a facilitator.
5. Prepare definition mechanism board, worksheets, wall stickier.
6. Participants should not criticize or debate.

FAST session Preparations Each attendee is asked to make a list of objects that are:

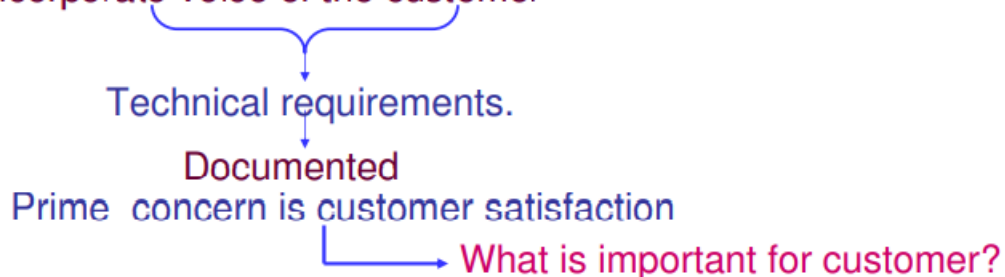
1. Part of environment that surrounds the system.
 2. Produced by the system.
 3. Used by the system.
- A. List of constraints
 - B. Functions
 - C. Performance criteria

Activities of FAST session

1. Every participant presents his/her list
2. Combine list for each topic
3. Discussion
4. Consensus list
5. Sub teams for mini specifications
6. Presentations of mini-specifications
7. Validation criteria
8. A sub team to draft specifications

4. Quality Function Deployment

-- Incorporate voice of the customer



- Normal requirements
- Expected requirements
- Exciting requirements

Steps

1. Identify stakeholders
2. List out requirements
3. Degree of importance to each requirement.

5 Points	:	V. Important
4 Points	:	Important
3 Points	:	Not Important but nice to have
2 Points	:	Not important
1 Points	:	Unrealistic, required further exploration

Requirement Engineer may categorize like:

- (i) It is possible to achieve
- (ii) It should be deferred & Why
- (iii) It is impossible and should be dropped from consideration

First Category requirements will be implemented as per priority assigned with every requirement.

5. The Use Case Approach

The **Use Case Approach**, introduced by Ivar Jacobson, is a method for capturing and modeling functional requirements in software development. It focuses on describing how users (or "actors") interact with a system to achieve specific goals. A use case represents a specific scenario or sequence of steps that the system performs in response to an actor's request. This method is useful in understanding system behavior from the user's perspective.

Use Case captures who (actor) does what (interaction) with the system, for what purpose (goal), without dealing with system internals.

*defines all behavior required of the system, bounding the scope of the system.

Jacobson & others proposed a template for writing Use cases as shown below

1. **Introduction**
Describe a quick background of the use case.
2. **Actors**
List the actors that interact and participate in the use cases.
3. **Pre Conditions**
Pre conditions that need to be satisfied for the use case to perform.
4. **Post Conditions**
Define the different states in which we expect the system to be in, after the use case executes.

5. Flow of events

5.1 Basic Flow

List the primary events that will occur when this use case is executed.

5.2 Alternative Flows

Any Subsidiary events that can occur in the use case should be separately listed. List each such event as an alternative flow.

A use case can have many alternative flows as required.

6.Special Requirements

Business rules should be listed for basic & information flows as special requirements in the use case narration .These rules will also be used for writing test cases. Both success and failures scenarios should be described.

7.Use Case relationships

For Complex systems it is recommended to document the relationships between use cases. Listing the relationships between use cases also provides a mechanism for traceability

Use Case Guidelines

1. Identify all users
2. Create a user profile for each category of users including all roles of the users play that are relevant to the system.
3. Create a use case for each goal, following the use case template maintain the same level of abstraction throughout the use case. Steps in higher level use cases may be treated as goals for lower level (i.e. more detailed), sub-use cases.
4. Structure the use case
5. Review and validate with users.

Use case Diagrams

- represents what happens when actor interacts with a system.
- captures functional aspect of the system.



Actor



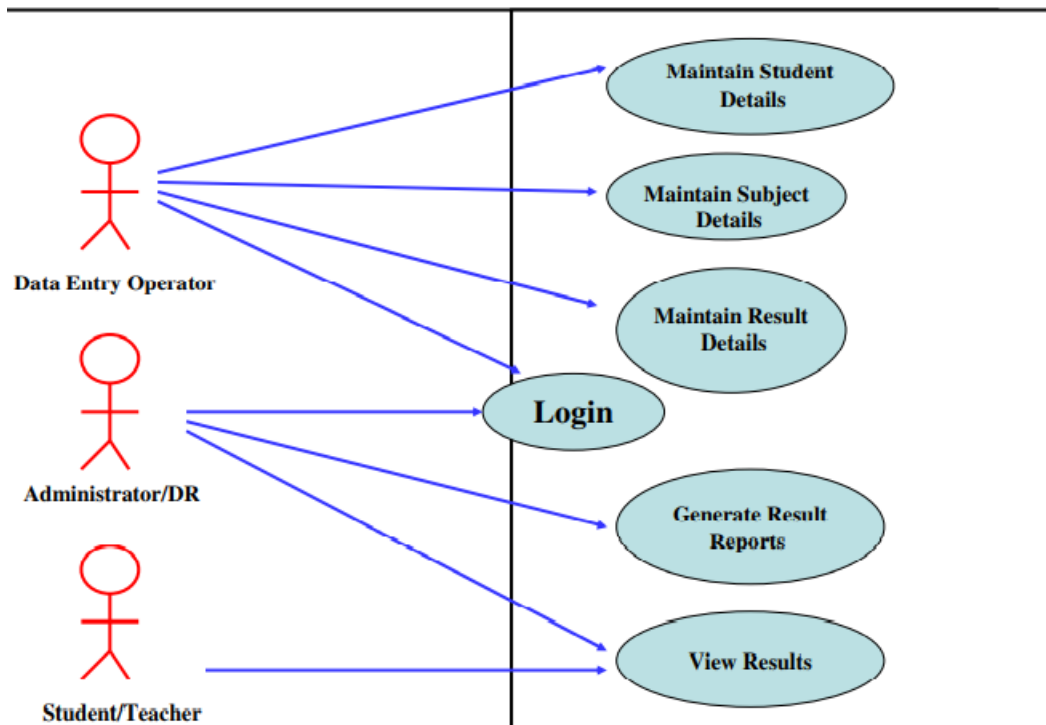
Use Case



Relationship between actors and use case and/or between the use cases.

- Actors appear outside the rectangle.
- Use cases within rectangle providing functionality.
- Relationship association is a solid line between actor & use cases.

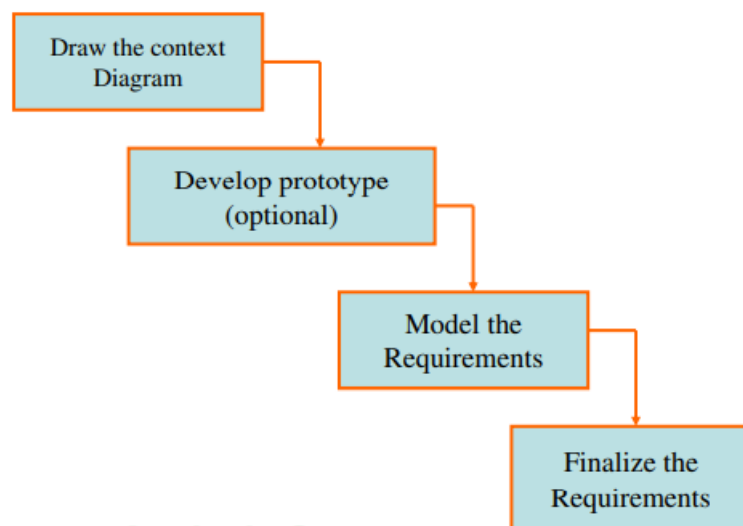
Use case diagram for Result Management System



Q30. Requirements Analysis.

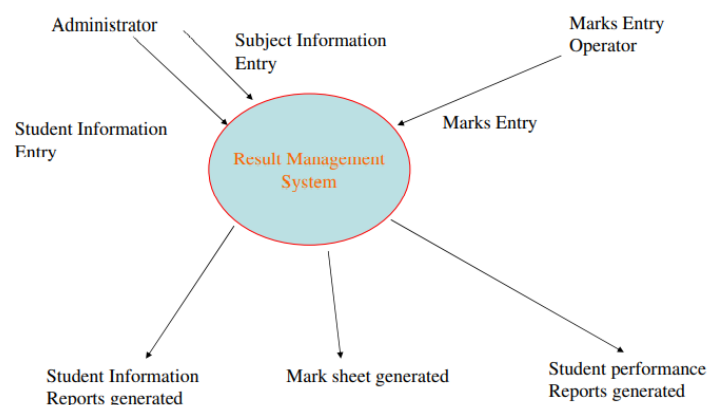
We analyze, refine and scrutinize requirements to make consistent & unambiguous requirements.

Steps



Requirements Analysis Steps


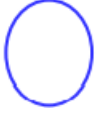
Ex:


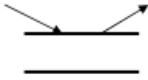


Data Flow Diagrams

DFD show the flow of data through the system.

- All names should be unique
- It is not a flow chart
- Suppress logical decisions
- Defer error conditions & handling until the end of the analysis

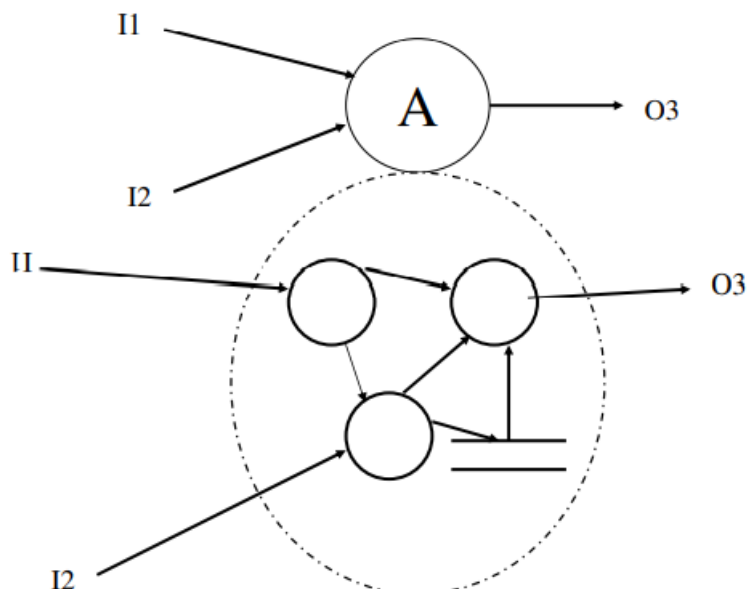
Symbol	Name	Function
	Data Flow	Connect process
	Process	Perform some transformation of its input data to yield output data.

Symbol	Name	Function
	Source or sink	A source of system inputs or sink of system outputs
	Data Store	A repository of data, the arrowhead indicate net input and net outputs

Q32. Leveling In DFD

Leveling DFD represent a system or software at any level of abstraction.

A level 0 DFD is called fundamental system model or context model represents entire software element as a single bubble with input and output data indicating by incoming & outgoing arrows.



Q33. Data Dictionaries

DFD \longrightarrow DD

Data Dictionaries are simply repositories to store information about all data items defined in DFD.

Includes :

Name of data item

Aliases (other names for items)

Description/Purpose

Related data items

Range of values

Data flows

Data structure definition

Notation

Meaning

$x = a + b$

x consists of data element a & b

$x = \{a/b\}$

x consists of either a or b

$x = (a)$

x consists of an optional data element a

$x = y\{a\}$

x consists of y or more occurrences

$x = \{a\}z$

x consists of z or fewer occurrences of a

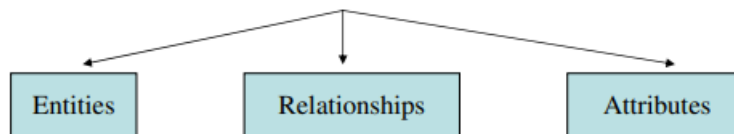
$x = y\{a\}z$

x consists of between y & z occurrences of a

Q34. ER Diagram.

Entity-Relationship Diagrams

It is a detailed logical representation of data for an organization and uses three main constructs.



Entities

Fundamental thing about which data may be maintained. Each entity has its own identity.

Relationships

A relationship is a reason for associating two entity types.

Binary relationships \longrightarrow involve two entity types

A CUSTOMER is insured by a POLICY. A POLICY CLAIM is made against a POLICY.

Attributes

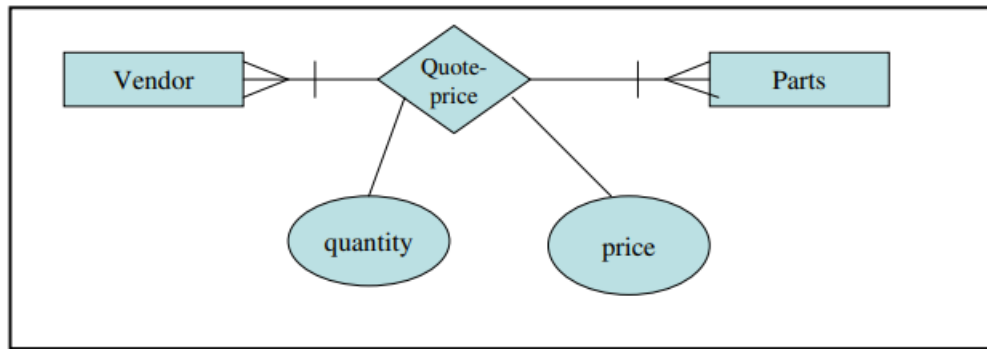
Each entity type has a set of attributes associated with it.

An attribute is a property or characteristic of an entity that is of interest to organization.



Attribute

Vendors quote prices for several parts along with quantity of parts.
Draw an E-R diagram.



Q35. Structured requirements definition (SRD)

Structured Requirements Definition (SRD) outlines steps to document system requirements clearly.

1. **User-Level DFD:** Create individual DFDs for each user, showing inputs (e.g., data provided by users) and outputs (e.g., system responses).
2. **Combined User-Level DFD:** Combine individual DFDs to show how all users interact with the system and each other.
3. **Application-Level DFD:** Focus on internal system processes, data stores, and data flows, showing how the system processes inputs and outputs.
4. **Application-Level Functions:** Define specific system functions based on the processes identified in the application-level DFD (e.g., validate login, process transactions).

Q35. Requirements Documentation

This is the way of representing requirements in a consistent format.

Requirements Documentation is the process of recording and organizing all the system requirements gathered during the development process. It serves as a reference for stakeholders and the development team to ensure everyone has a clear understanding of what the system should do.

- **Functional Requirements:** Specify what the system must do (e.g., user login, data processing).
- **Non-Functional Requirements:** Define system qualities such as performance, security, and usability.
- **Business Requirements:** High-level needs and objectives from the business perspective.
- **Technical Requirements:** Specific technical details, such as platform and technology constraints.
- **Assumptions and Constraints:** Any assumptions made and limitations on the project.

Q36. Nature of SRS

- Correct

An SRS is correct if and only if every requirement stated therein is one that the software shall meet.

- Unambiguous

An SRS is unambiguous if and only if, every requirement stated therein has only one interpretation.

- Complete

An SRS is complete if and only if, it includes the following elements

(i) All significant requirements, whether related to functionality, performance, design constraints, attributes or external interfaces.

(ii) Responses to both valid & invalid inputs.

(iii) Full Label and references to all figures, tables and diagrams in the SRS and definition of all terms and units of measure.

- Consistent

An SRS is consistent if and only if, no subset of individual requirements described in it conflict.

- Ranked for importance and/or Stability

If an identifier is attached to every requirement to indicate either the importance or stability of that particular requirement

- Verifiable

An SRS is verifiable, if and only if, every requirement stated therein is verifiable.

- Modifiable

An SRS is modifiable, if and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining structure and style.

- Traceable

An SRS is traceable, if the origin of each of the requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation.

All of these are characteristics of a good SRS

Q37. Organization of the SRS

IEEE has published guidelines and standards to organize an SRS.

First two sections are same. The specific tailoring occurs in section-3.

1. Introduction

- | | |
|-----|--|
| 1.1 | Purpose |
| 1.2 | Scope |
| 1.3 | Definition, Acronyms and abbreviations |
| 1.4 | References |
| 1.5 | Overview |

2. The Overall Description

2.1 Product Perspective

- 2.1.1 System Interfaces
- 2.1.2 Interfaces
- 2.1.3 Hardware Interfaces
- 2.1.4 Software Interfaces
- 2.1.5 Communication Interfaces
- 2.1.6 Memory Constraints
- 2.1.7 Operations
- 2.1.8 Site Adaptation Requirements

2.2 Product Functions

2.3 User Characteristics

2.4 Constraints

2.5 Assumptions for dependencies

2.6 Apportioning of requirements

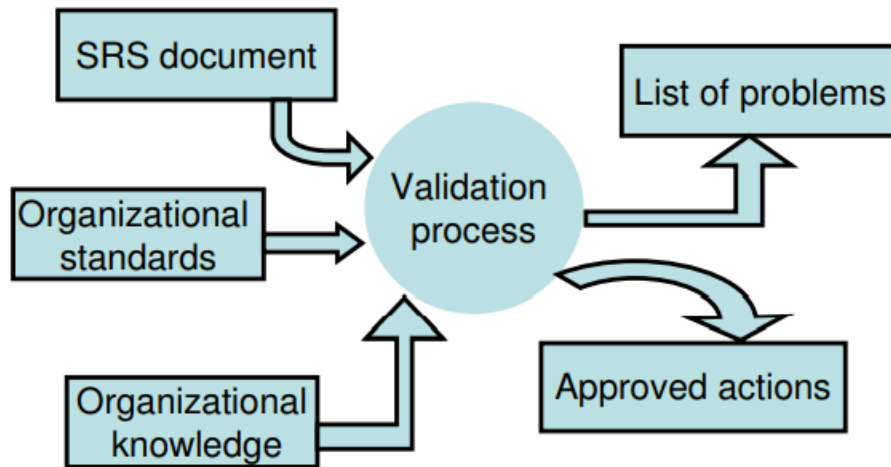
3. Specific Requirements

- 3.1 External Interfaces
- 3.2 Functions
- 3.3 Performance requirements
- 3.4 Logical database requirements
- 3.5 Design Constraints
- 3.6 Software System attributes
- 3.7 Organization of specific requirements
- 3.8 Additional Comments.

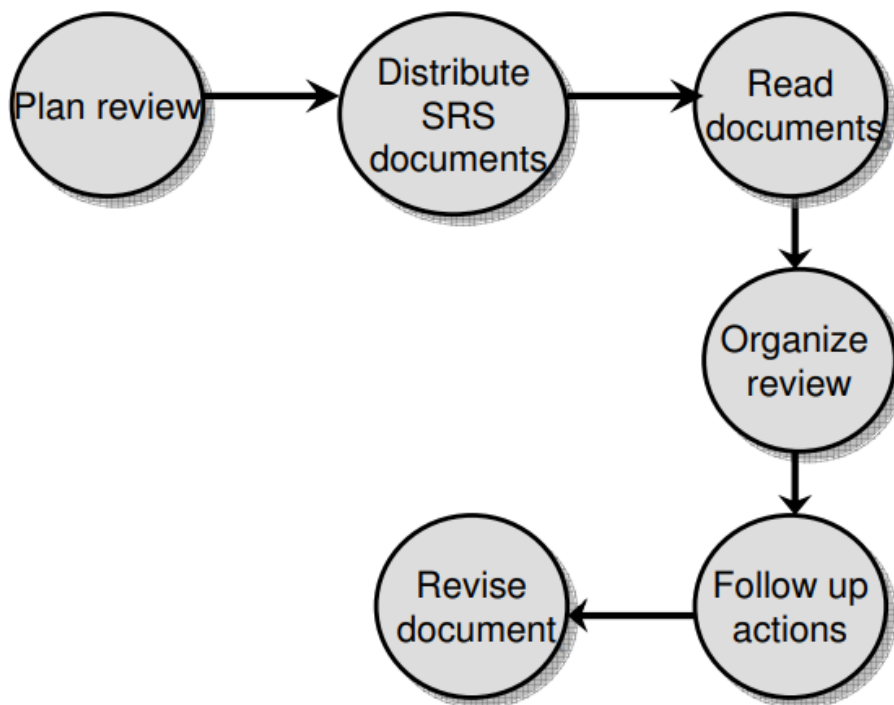
Q38. Requirements Validation

Check the document for:

- ✓ Completeness & consistency
- ✓ Conformance to standards
- ✓ Requirements conflicts
- ✓ Technical errors
- ✓ Ambiguous requirements



Q39. Requirements Review Process



Q40. Review Checklists

- ✓ Redundancy
- ✓ Completeness
- ✓ Ambiguity
- ✓ Consistency
- ✓ Organization
- ✓ Conformance
- ✓ Traceability

Q41. Requirements Management

Requirements Management is the process of handling and controlling requirements throughout the software development lifecycle. It ensures that all requirements are properly documented, tracked, and maintained, adapting to changes as needed. This process is crucial for project success and stakeholder satisfaction.

Key components of Requirements Management include:

1. **Requirements Identification:** Clearly defining and documenting all requirements, including functional and non-functional aspects.
2. **Requirements Traceability:** Maintaining a clear link between requirements and other project artifacts, such as design, implementation, and testing. This helps ensure that all requirements are addressed and facilitates impact analysis when changes occur.
3. **Change Control:** Establishing a formal process for managing changes to requirements. This includes evaluating the impact of changes, obtaining necessary approvals, and updating documentation accordingly.
4. **Requirements Communication:** Ensuring that all stakeholders, including developers, testers, and clients, understand the requirements and any changes made. Effective communication helps minimize misunderstandings and misalignments.
5. **Requirements Validation and Verification:** Regularly reviewing requirements to ensure they are complete, accurate, and feasible. This involves validating requirements with stakeholders and verifying that the system meets these requirements during development and testing.
6. **Requirements Prioritization:** Assessing and ranking requirements based on their importance and urgency, which helps in decision-making regarding resource allocation and scope management.