Times asked 7 times

6 times

5 times

4 times

3 times

2 times

1 time

Software Engineering Question bank

1. Introduction to SE and Process models (20-25 marks)

1. Define Software Engineering. Explain Software process framework.
2. Explain Umbrella activities of SE.
3. What is Capability Maturity Model (CMM). Explain different CMM levels.
4. Explain Waterfall model and its limitations.
5. Explain Spiral model and its limitations.
6. Explain Agile process model.
7. Explain SCRUM model.
8. Compare SCRUM and Kanban.

2. Software Requirements Analysis and Modelling (15-25 marks)

1. Explain the requirement model.
2. Explain the development of use case. / Explain UML Model.
3. Draw Use case diagram for Hospital Management System.
4. What is SRS? Explain characteristics of SRS document.
5. Explain the general format of SRS.
6. Develop an SRS for: (Previously asked:)
    a. Online Movie booking system
    b. University management system
    c. Hospital management system
7. Explain the different levels of DFD.
8. Design DFD for Library management system.

3. Software Estimation and Metrics (15-25 marks)

1. Explain function point-based (FP) estimation technique in detail.
2. What is project and what are the different metrics used for software measurement.
3. Explain the LOC method.
4. Explain COCOMO model in detail.
5. Explain 3Ps in software project spectrum.
6. Explain project scheduling and tracking.

4. Software Design (20-25 marks)

1. Explain Cohesion and Coupling. Explain different types of Cohesion and Coupling.
2. What are the software design principles. Explain in detail with examples.
3. What is User interface design? How does it help web technology and IT industry?
4. Design user interface for Online shopping system.

5. Software Testing (25-30 marks)

1. What is software testing? Explain different types of software testing.
2. Differentiate between White box testing & Black box testing.
3. Explain the software testing process.
4. List out different software testing strategies.
5. Explain different techniques in White box testing.
6. Differentiate between Alpha & Beta testing.
7. What is maintenance? Explain the different types of maintenance.
8. Explain software re-engineering.
9. Explain software reverse engineering.
10. Design the test cases for Medical Management Application.

6. Software Configuration Management, Quality Assurance and Maintenance (30 marks)

1. Define risk. What are the different categories of risk.
2. Explain RMMM plan with suitable example.
3. Explain Software Quality Assurance(SQA) in detail.
4. Explain FTR.
5. Compare FTR and Walkthrough.
6. Explain Walkthrough
7. Explain Software configuration (SCM) process.
8. Explain change control and version control. How is change control different than version control.

	1	2	3	4	5	6
2024 May	25	15	10	25	25	30
2023 Dec	5	25	25	20	20	30
2023 May	25	30	15	10	30	20
2022 Dec	20	25	20	20	30	25
Last 4 Avg	**20**	**25**	**15**	**20**	**25**	**30**
*2022 May	5	20	5	15	25	25
2019 Dec	15	5	15	15	45	35
2019 May	15	25	15	25	35	15
Total	110	145	105	130	210	185

Software Engineering Answer bank

1. Introduction to SE and Process models

1. Define Software Engineering. Explain Software process framework.

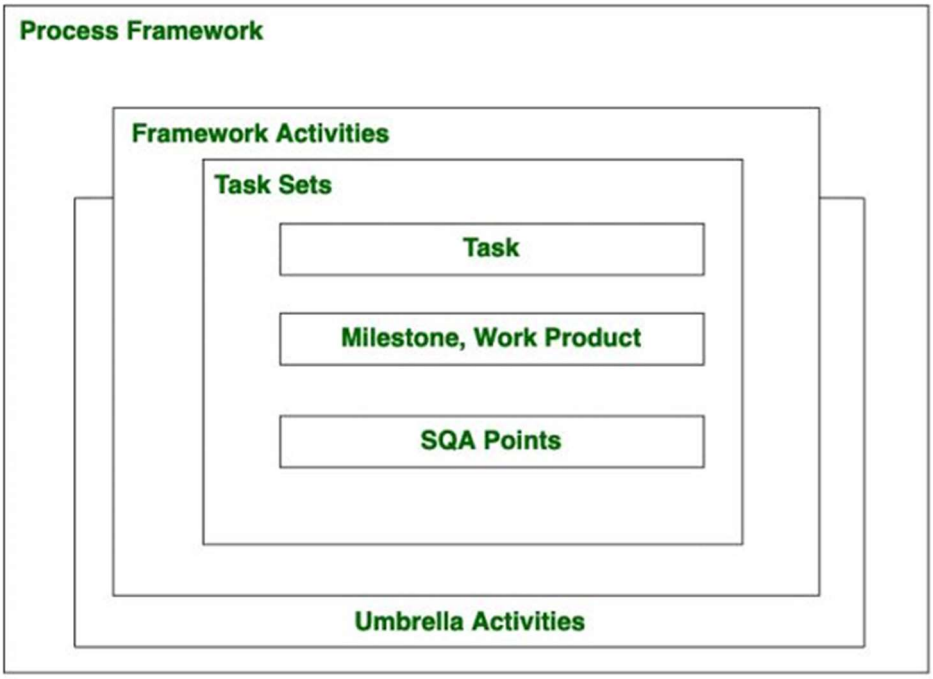
- **Software Engineering (SE)** is a disciplined approach to designing, developing, maintaining, testing, and managing software systems.
- It combines engineering principles with computer science to create reliable, efficient, and scalable software solutions.
- The field focuses on applying structured methodologies, tools, and techniques to build software that meets specific requirements while considering quality, cost, and time constraints.

Key Aspects of Software Engineering:

- Systematic Approach
- Quality Focus
- Lifecycle Management
- Team Collaboration

Software Process Framework:

- A Software Process Framework is a standard method used to build and deploy applications.
- It forms the basis for the entire software engineering process.
- The framework includes all umbrella activities needed for software development.
- It also outlines key tasks that apply to all software projects.



A generic process framework encompasses five activities which are given below one by one:

Communication: Communication with the client is required to understand the needs, demands, criteria, and parameters of the project.
Team communication makes sure that everyone is on the same page.

Planning: Involves making a map or blueprint to break down the process of development and document goals, milestones, and plans.

Modelling: Developers create a model so the client can visualize the finished product.

Construction: The actual coding and testing of the product.

Deployment: The team delivers an actual version of the software for testing, evaluation, and feedback.

2. Explain Umbrella activities of SE.

- Software engineering is a collection of co-related steps.
- These steps are presented or accessed in different approaches in different software process models.
- Umbrella activities are a set of steps or procedure that the software engineering team follows to maintain the progress, quality, change and risks of the overall development tasks.

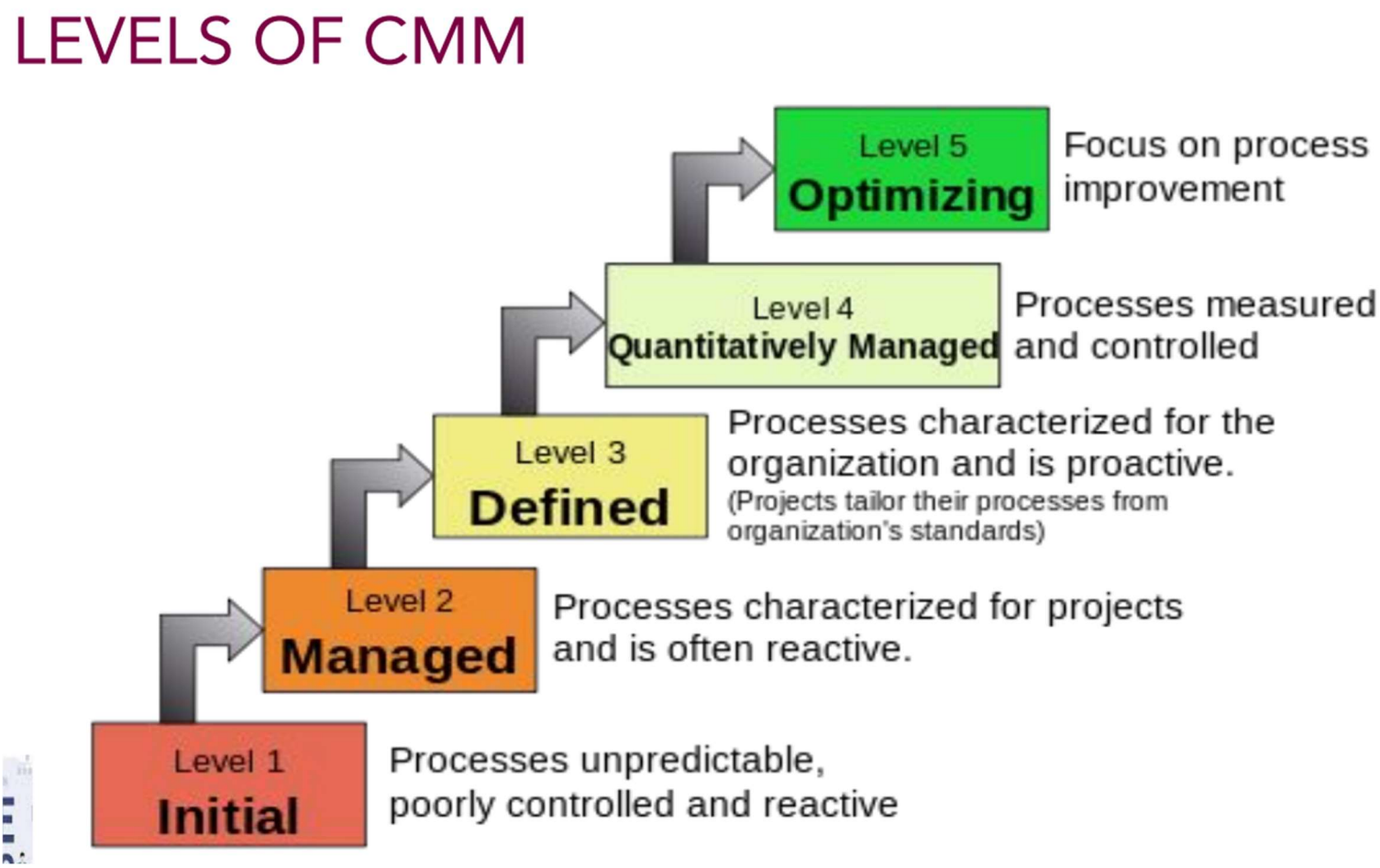
Umbrella Activities are as follows:

1. **Software Project Tracking and Control:**
   Before the actual development begins, a schedule for developing the software is created. Based on that schedule, the development will be done.
2. **Formal Technical Reviews:**
   Software engineering is done in clusters or modules; after completing each module, it is good practice to review the completed module to find out and remove errors so their propagation to the next module can be prevented.
3. **Software Quality Assurance:**
   The quality of the software such as user experience, performance, load handling capacity etc. should be tested and confirmed after reaching predefined milestones.
4. **Software Configuration Management:**
   Software configuration management (SCM) is a set of activities designed to control change by identifying the work products that are likely to change, establishing relationships among them, defining mechanisms for managing different versions of these work products.
5. **Document preparation and production:**
   All the project planning and other activities should be hard copied and the production get started here.
6. **Reusability Management:**
   This includes the backing up of each part of the software project they can be corrected, or any kind of support can be given to them later to update or upgrade the software at user/time demand.
7. **Measurement & Metrics:**
   Include all the measurement (Efforts) of every aspects of the software project.
   Direct measures: cost, line of code, size of team and software .
   Indirect measures: quality of software (efficiency, usability etc)
8. **Risk Management:**
   Risk management is a series of steps that help a software team to understand and manage uncertainty. e.g. data loss, availability, response, security.

3. What is Capability Maturity Model (CMM). Explain different CMM levels.

- **Capability Maturity Model (CMM)** is a methodology used to develop and refine an organization's software development process
- CMM is not a software process model
- CMM is used as a benchmark to measure maturity of an organization's software process
- Describes a five-level evolutionary improvement path for software organizations from an ad hoc, immature process to a mature, disciplined one.

It provides the guidelines to further enhance the maturity.



Maturity Level 1 – Initial: *Work Performed informally.*

• Company has no standard process for software development. Processes are disorganized, ad hoc and even chaotic.

Maturity Level 2 – Managed/ Repeatable: *Work is planned and Tracked*

• Company has installed basic software management processes and controls to track cost, schedule, and functionality.

• The process is in place to repeat the earlier successes on projects with similar applications.

Maturity Level 3 – Defined: *Work is well defined*

• Company has pulled together a standard set of processes and controls for the entire organization so that developers can move between projects more easily and customers can begin to get consistency from different groups.

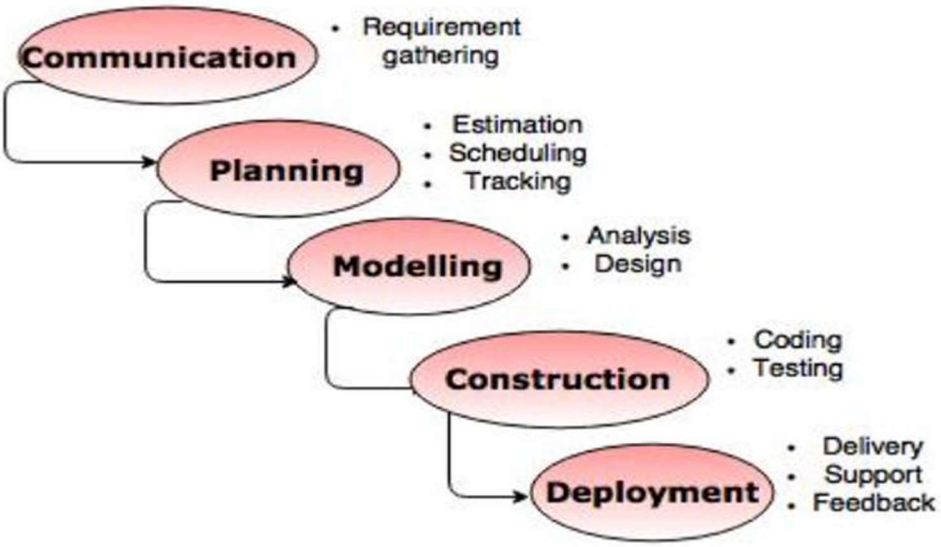
Maturity Level 4 – Quantitatively Managed: *Work is Quantitatively Controlled*

• In addition to implementing standard processes, company has installed systems to measure the quality of those processes across all projects.

Maturity Level 5 – Optimizing: *Continuous Improvement*

• Company has accomplished all of the above and can now begin to see patterns in performance over time, so it can tweak its processes in order to improve productivity and reduce defects in software development across the entire organization.

4. Explain Waterfall model and its limitations.



- It is also referred to as a ”linear-sequential model” or “Classic life cycle model”
- In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.
- This model is used for small projects.
- Feedback is taken after each phase to ensure that the project is on the right path
- Testing part starts only after the development is complete.
- It is used when user requirements are fixed and clear that may not change

Communication:

In waterfall model customer must state all requirements at the beginning of project.

Planning:

It includes complete estimation(e.g. cost estimation of project) and scheduling (complete timeline chart for project development) and testing.

Modelling:

It includes detailed requirement analysis and project design(algorithm, flowchart etc).

Construction:

It includes coding and testing steps:

i. **Coding**: Design details are implemented using appropriate programming language

ii. **Testing:** Testing is carried out to check whether flow of coding is correct, to check out the errors of program

Deployment:

It includes software delivery, support and feedback from customer.

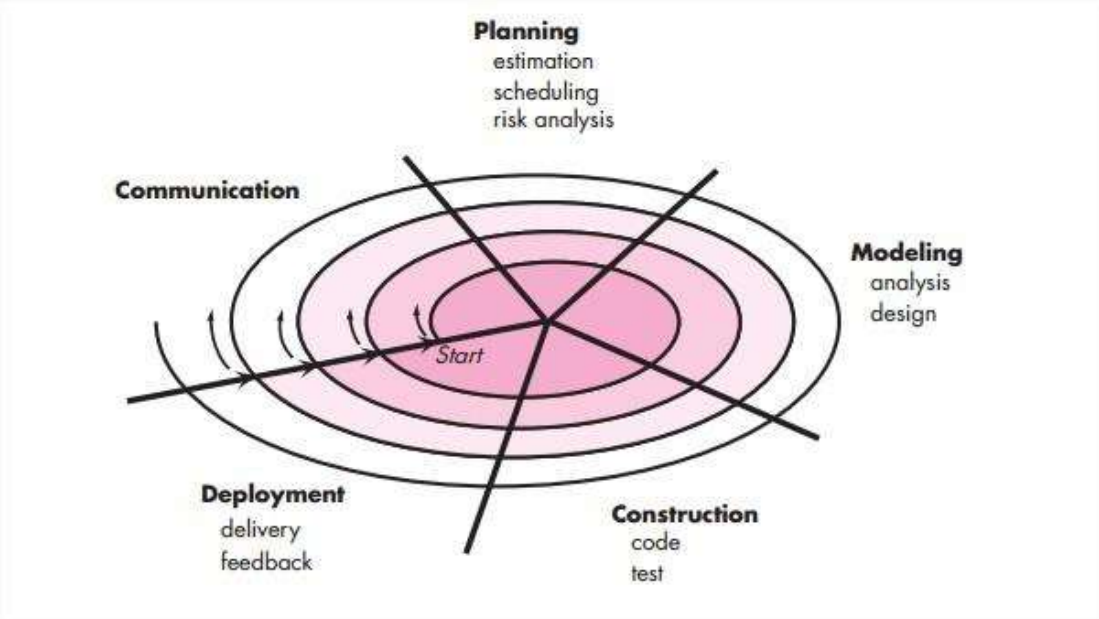
Advantages of waterfall model:

1. Simple and easy to understand and use
2. Phases are processed and completed one at a time.
3. Clearly defined stages.

Disadvantages or limitations of waterfall model:

1. "Blocking states" members of one team has to wait for other team members to complete the dependant tasks.
2. No working software is produced until late during the life cycle.
3. High amounts of risk and uncertainty.
4. Not a good model for complex projects.

5. Explain Spiral model and its limitations.



- Proposed by Boehm
- Combines the idea of iterative development with the systematic and controlled aspects of the waterfall model.
- Combination of iterative development process model and sequential linear development model i.e. the waterfall model with a very high emphasis on risk analysis.
- Using spiral model, software is developed as series of evolutionary releases.
- It allows incremental releases of the product or incremental refinement through each iteration around the spiral.
- Each framework activity represents one segment of spiral
- The initial activity is shown from centre of the circle and developed in clockwise direction

Communication:

The software development process starts with communication between customer and developer.

Planning:

It includes complete estimation(e.g. cost estimation of project) and scheduling (complete timeline chart for project development) and Tracking and **Risk Analysis.**

Modelling:

It includes detailed requirement analysis and project design(algorithm, flowchart etc)

Flowchart shows complete pictorial flow of program whereas algorithm is step-by-step solution of problem

Construction: It includes coding and testing steps**:**

i. **Coding**: Design details are implemented using appropriate programming language
ii. **Testing**: Testing is carried out to check whether flow of coding is correct, to check out the errors of program

Deployment:

It includes software delivery, support and feedback from customer.

If customer suggest some corrections, or demands additional capabilities then changes are required for such corrections or enhancement.

Advantages of Spiral Model:

1. **Flexibility in Requirements**: Allows for evolving or changing requirements throughout the project.
2. **Customer Involvement**: Regular feedback from the customer at each iteration.
3. **Iterative Development**: Software is developed incrementally, allowing for gradual improvement.

Limitations of Spiral Model:

1. **Complexity**: Managing multiple cycles can become complicated.
2. **Costly**: High cost due to extensive planning and risk analysis.
3. **Not Suitable for Small Projects**: Overhead is too large for small-scale projects.

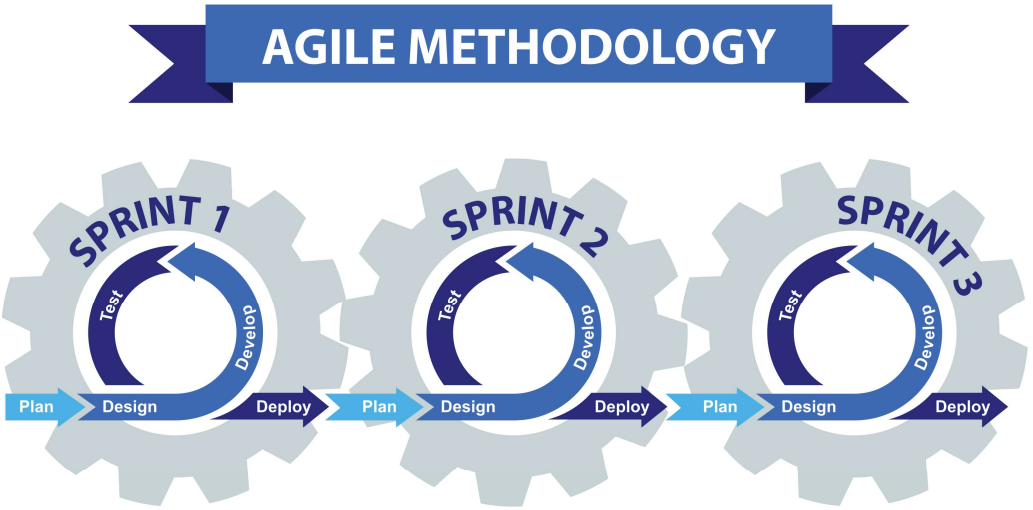
6. Explain Agile process model.

- In software development, Agile means ability to respond quickly to changes – changes in Requirements, Technology or people.
- Agile SDLC model is a combination of iterative and incremental process model with focus on customer satisfaction by rapid delivery of working software product.
- It offers direct collaboration with customer.
- Agile methods break the product into smaller incremental build, these builds are provided in increments.

- Each iteration typically lasts from about 1 to 3 weeks
- Each release is tested to ensure quality is maintained

The most popular Agile methods include:

• Extreme Programming(XP) • Scrum • Kanban



7. Explain SCRUM model.

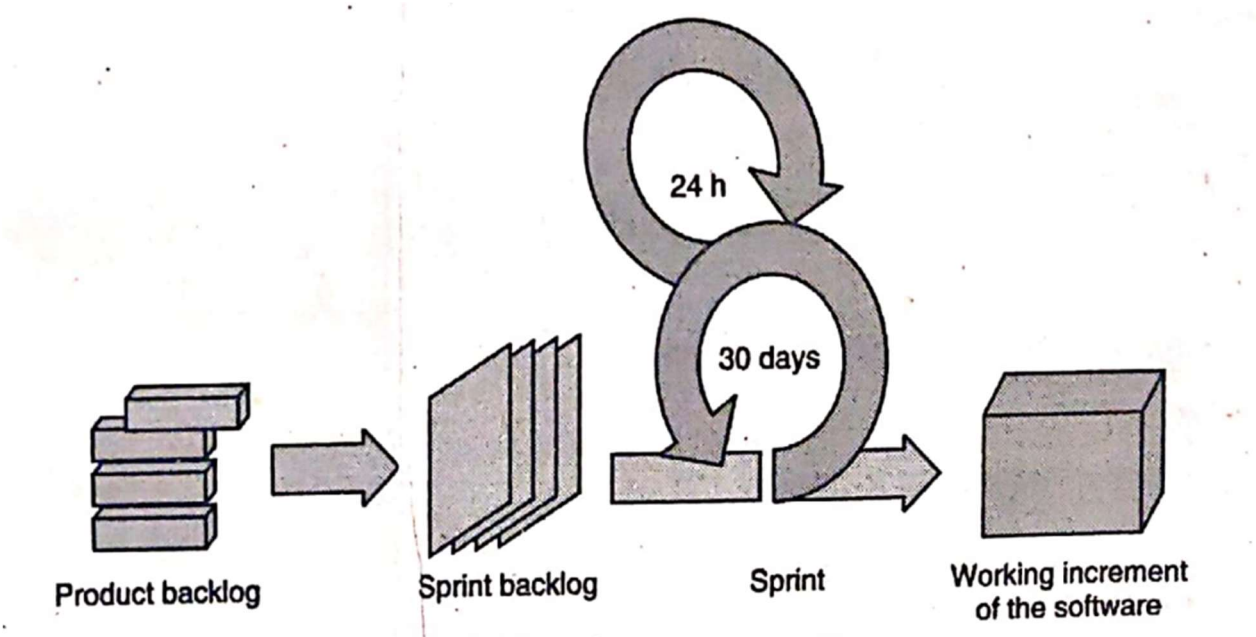


Fig. 1.13.2 : The scrum process

- It is an Iterative and Incremental Software Development framework
- It is an agile process model used to develop complex and sustainable products
- It enables teams to self-organize by collaborations of team members.
- Scrum is a framework utilizing an agile mindset for developing, delivering, and sustaining products in a complex environment

SCRUM ROLES

Different roles are used in scrum process. Three different roles are as follows:

1. Product owner: A Product owner orders the work for a complex problem into a Product Backlog.

2. Development team: The Development team turns a selection of the work into an Increment of value during a Sprint.

3. Scrum Master: Person responsible for scrum process, acts as a facilitator for the Product owner and the team

Sprints are a short, time-boxed period for Scrum team that works to complete a set /amount of work. Sprints are the core component of Scrum and agile methodology.

Sprint Planning: At the start of each sprint, a sprint planning meeting is held.

SCRUM CYCLE DESCRIPTION

Scrum cycle is divided into different activities:

1. Define
2. Plan
3. Build
4. Review
5. Retrospect

ADVANTAGES:

i. Large projects can be broken down into small sprints.
ii. Customers are involved, so best results are ensured.

DISADVANTAGES (Limitations)

i. Once the sprint backlog items are decided, it cannot be changed.
ii. If any member leaves in the middle of the project, it can have huge negative impact on the project.

8. Compare SCRUM and Kanban.

Aspect	Scrum	Kanban
Framework	Fixed-length sprints (2-4 weeks)	Continuous flow, no fixed iterations
Iterations	Work done in sprints	Work flows continuously
Work Management	Planned at start of each sprint	Pulled as capacity allows
Roles	Scrum Master, Product Owner, Team	No fixed roles
WIP Limits	Managed by sprint goals	Explicit WIP limits per stage
Meetings	Daily stand-ups, planning, reviews, retros	No required meetings, optional stand-ups
Changes During Work	No changes during a sprint	Changes anytime
Focus	Deliver set of features per sprint	Optimize flow and delivery
Metrics	Velocity (story points/sprint)	Cycle time, lead time
Best For	Structured, time-boxed work	Flexible, continuous work

2. Software Requirements Analysis and Modelling

1. Explain the requirement model.

1) The procedure which collects the software requirements from customer, analyse and document them is called as Requirement Engineering.
2) Requirement modelling is implemented after the requirements and constraints have been collected and further analysed.
3) Requirement modelling is essential to ensure the consistency and completeness of requirements. Different approaches are used to model functional requirements, quality attributes, and constraints, depending on the system type and organizational standards.
4) There are many ways requirements can be modelled; some of the most common requirement models are as follows:
    i. Use-case diagram
    ii. Data flow diagram
    iii. Sequence diagram
5) Activities involved in creating a Requirement model:
    i. Requirement Inception
    ii. Requirement Elicitation
    iii. Requirement Elaboration
    iv. Requirement Negotiation
    v. Requirement Specification
    vi. Requirement Validation
    vii. Requirements management

2. Explain the development of use case. / Explain UML Model.

Unified Modelling Language (UML)

- Also known as scenario-based modelling.
- A **use case diagram** represents the system's dynamic behaviour.
- It shows the system's functionality using use cases, actors, and their interactions.

Components of a Use Case Diagram:

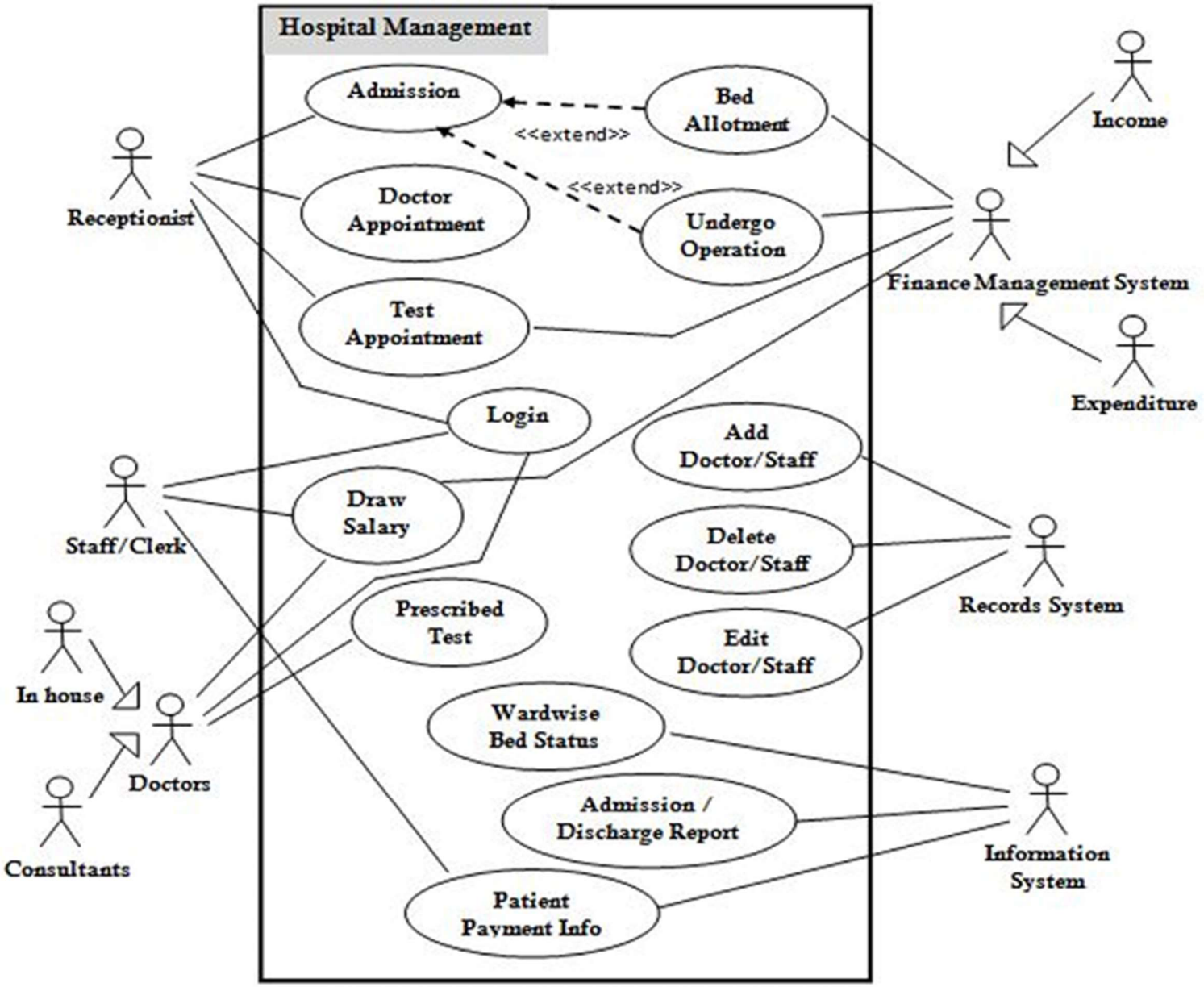
- **Use Cases**: Ovals representing different actions or uses a user may have.
- **Actors**: Stick figures representing the people or systems interacting with the use cases.
- **Associations**: Lines connecting actors and use cases, showing who interacts with what. For complex diagrams, it’s important to clarify which actors are linked to which use cases.
- **System Boundary Boxes**: A box that defines the system’s scope for use cases. Any use case outside the box is outside the system’s scope.
- **Packages**: A UML shape used to group related elements, similar to file folders. Just like in component diagrams, these groups organize elements logically.

Rules for Drawing a Use Case Diagram:

- Assign clear, meaningful names to actors and use cases.

- Ensure the communication between actors and use cases is easy to understand.
- Use appropriate notations wherever necessary.
- Highlight the most important interactions between actors and use cases.
- For large or complex diagrams, keep them generalized to avoid confusion.

3. Draw Use case diagram for Hospital Management System.



4. What is SRS? Explain characteristics of SRS document.

SRS (Software Requirements Specification) is a detailed document that describes the software system to be developed. It outlines the system's functional and non-functional requirements, performance criteria, design constraints, and interfaces with other systems. The SRS serves as a formal agreement between stakeholders (clients, users, developers) and acts as a reference throughout the software development lifecycle.

Characteristics of a Good SRS Document:

1. **Correctness**:
    - The SRS must accurately describe the system that the stakeholders need, ensuring no requirement is missed or misinterpreted.

2. **Unambiguous**:
    - Requirements must be clearly defined in a manner that is easily understood by all stakeholders and leaves no room for multiple interpretations.
3. **Completeness**:
    - The SRS should include all necessary requirements, covering both functional and non-functional aspects of the system without leaving any ambiguities or gaps.
4. **Ranked for importance:**
    - Requirement should be ranked using different factors like stability, security, ease or difficulty.
5. **Modifiability**:
    - The SRS should be organized in a way that allows easy updating or modification of requirements. This is important as requirements may evolve during the development process.
6. **Traceability**:
    - Every requirement should be traceable, meaning it should be possible to trace each requirement from the SRS document to its implementation, and vice versa.
7. **Verifiable**:
    - Every requirement in the SRS should be testable and measurable. This means you should be able to verify, through testing or inspection, that the system meets the stated requirements.

5. Explain the general format of SRS.

1. Introduction:

- Brief overview of the SRS document.
    - **i. Purpose**: Defines the purpose of the SRS and the intended audience.
    - **ii. Scope**: Explains the system's benefits, objectives, behaviour, limitations, and boundaries.
    - **iii. Definitions, Acronyms, Abbreviations**: Clarifies terms used in the SRS to avoid confusion.
    - **iv. References**: Lists documents referred to in the SRS.
    - **v. Overview**: Summarizes the document, goals, and system objectives.

2. Overall Description:

- Provides general information about the system requirements.
    - **i. Product Perspective**: Explains the advantages of the product compared to others.
    - **ii. Product Function**: Summarizes the product's functionality, using diagrams if needed.

- **iii. User Characteristics**: Describes the necessary knowledge or qualifications users need.
- **iv. Constraints**: Lists limitations like hardware restrictions.
- **v. Assumptions and Dependencies**: Details factors that may affect the SRS.
- **vi. Apportioning of Requirements**: Specifies the order of requirement fulfilment.

3. Specific Requirements:

- Breaks down detailed system requirements.
  - **i. Interfaces**: Describes hardware, software, system, user, and communication interfaces.
  - **ii. Database**: Information on the database for storing system data.
  - **iii. Performance**: Specifies performance metrics like speed, response time, and throughput.
  - **iv. Software System Attributes**: Discusses system reliability, security, and maintainability.

4. Change Management Process:

- Details how changes due to user requirements will be handled and how the SRS will be updated accordingly.

5. Document Approvals:

- Requires acceptance from both the customer and developer, including approval date, time, and signatures.

6. Supporting Information:

- Provides guidelines, index, and references related to the use of the SRS.

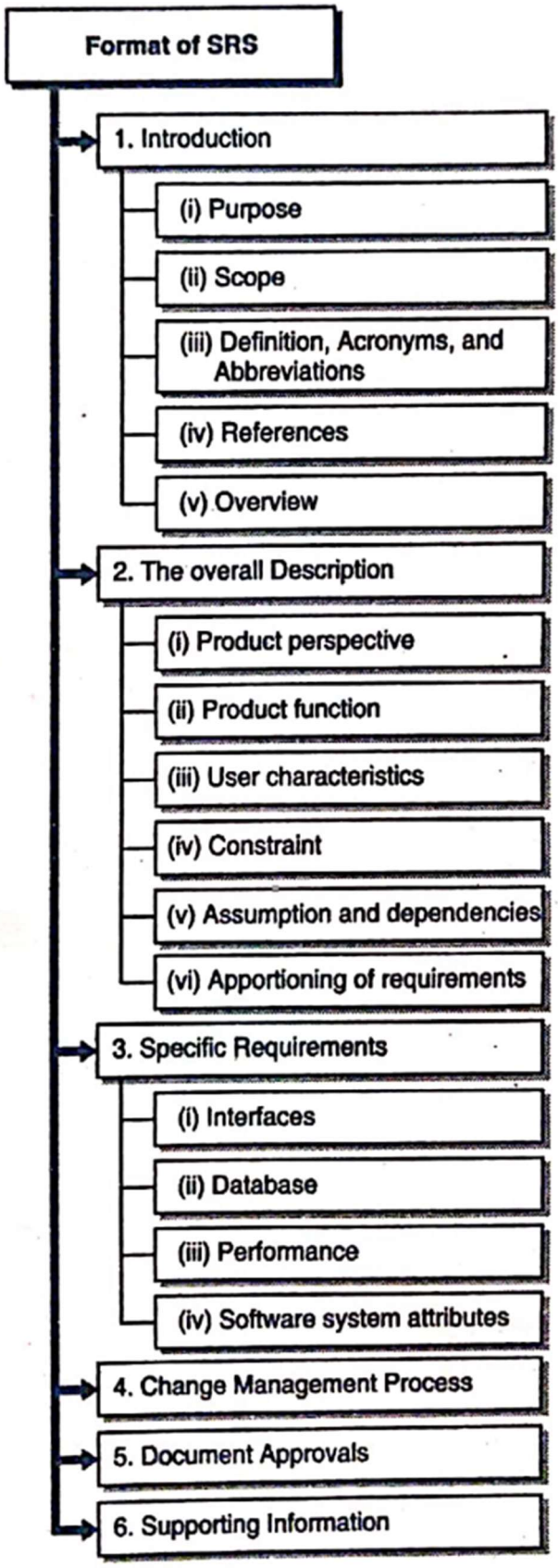


Fig. 2.5.2 : Format of SRS

6. Develop an SRS for: (Previously asked)

a. Online Movie booking system

b. University management system

c. Hospital management system

Title

System Requirement Specification Document for Hospital Management System

Objective

To get with preparing requirement document, which will be used to capture and document all the requirements at the start of project. We mainly focus on functional requirements.

Introduction

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

ii. Document Conventions:

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

Scope of the Project

• The purpose of this specification is to document requirements for a system hospital.

• The HMS will manage a waiting list of patients requiring different treatments.

• The availability of beds will be determined and if beds are available the next appropriate patient on the list will be notified.

• Nurse will be allocated to wards depending on ward sizes, what type of nursing is needed, operating schedules etc.

• The current manual method of managing patients, nurses, and beds is time consuming and error prone. It is also difficult to manage the large paper flow involved in this process.

• The HMS will allow hospital administrative staff to access information efficiently and effectively

• The goal of HMS is to manage nurses, patients, beds, and patients' medical information in ancon effective manner.

• All of these sub-systems (managing nurses, beds, patient medical information) need to be designed and implemented so that HMS can run effectively

Overall Description

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

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

• Operating Environment: Following are the requirements for running the software successfully

- Processor - Pentium III or Higher.
- Ram – 512 MB or Higher
- Disk Space – 10 GB or Higher
- OS – Windows XP or Above

Design and Implementation Constraints

• GUI only in English.

• Login and password is used for identification of user and there is facility for guest.

Assumption and Dependencies

• It is assumed that one hundred compatible computers will be available before the system is installed and tested.

• It is assumed that Hospital will have enough trained staff to take care of the system.

External Interface Requirements

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

• Hardware Interface: These are the minimum hardware interfaces:

- Processor: Pentium III or Higher
- Ram: 512 MB or Higher.
- Disk Space: 10 GB or Higher
- Software Interface: These are the minimum software interfaces:
- Technologies : CE Net 20
- Database : SQL server (standard edition)
- Operating system : Windows XP or above.

System Features

• Work Scheduling : Assigning nurses to doctors and doctors to patients.

• Admissions : Admitting patients to appropriate wards.

• Patient Care: Monitoring patients while they are the hospital

• Surgery Management : Planning and organizing the work that surgeons and nurses

perform in the operating rooms

• Ward Management : Planning and coordinating the management of wards and rooms.

• Waiting List: Monitoring to see if there are any patients waiting for available beds, assigning them to doctors and beds once these become available.

Non-functional Requirements

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

- Password Management
- Regular Database Archiving
- Virus Protection

• Safety Requirements: Humans are error-prone, but the negative effects of common errors should be limited.

• Security Requirements:

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

• Software Quality Attributes:

• The Quality of the system is maintained in such a way that it can be very user-friendly

• The software quality attributes are assumed as follows:

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

7. Explain the different levels of DFD.

Data Flow Diagrams (DFD) represent the flow of information within a system, from inputs to processes and outputs. DFDs are divided into multiple levels to show varying degrees of detail in the system. Here are the key levels:

1. Level 0 DFD (Context Diagram)

- **Overview**: This is the most basic DFD, representing the entire system as a single process. It shows the system's interaction with external entities (such as users, external systems) without diving into internal processes.

- **Purpose**: Provides a high-level view of the system.

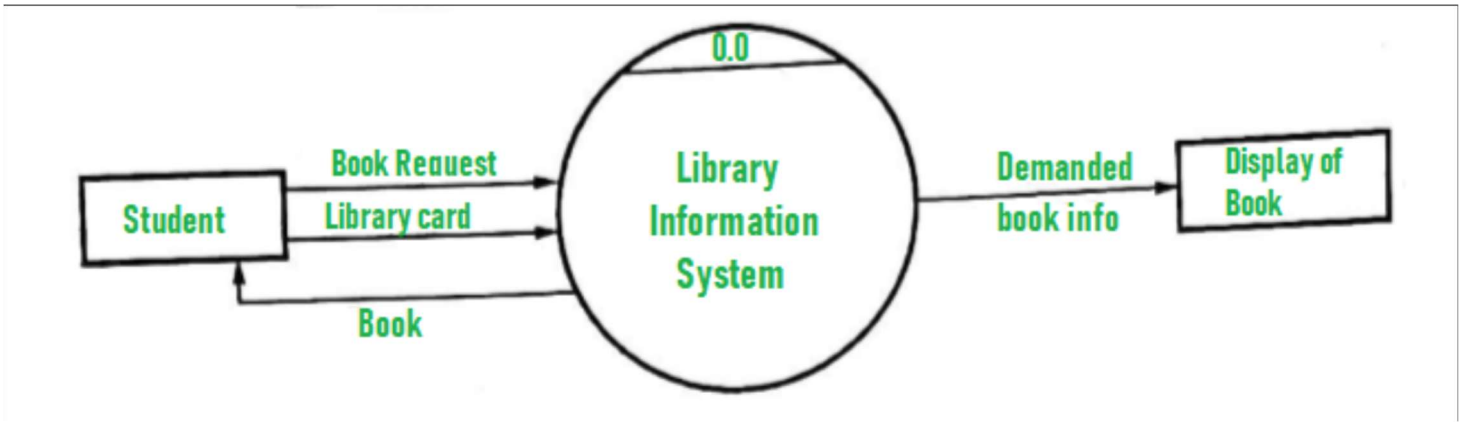
2. Level 1 DFD

- **Overview**: This level breaks down the single process from the context diagram into **major sub-processes**. It shows more details about how the system functions internally, while still maintaining a broad view.
- **Purpose**: Describes the main functions or sub-processes within the system.

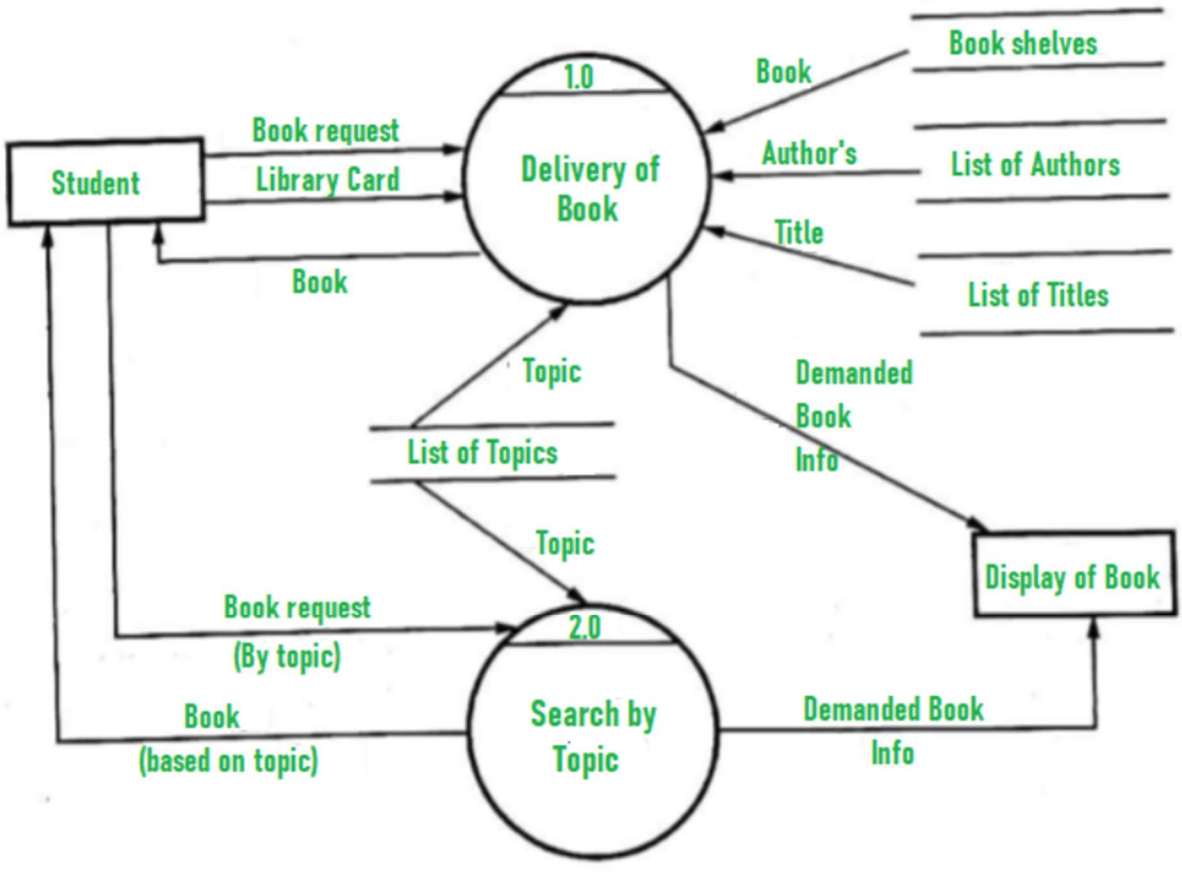
3. Level 2 DFD

- **Overview**: This level further decomposes the processes from Level 1 into more specific, detailed sub-processes. It provides a granular view of how data moves within each sub-process.
- **Purpose**: Describes in detail how data is processed within each sub-process.

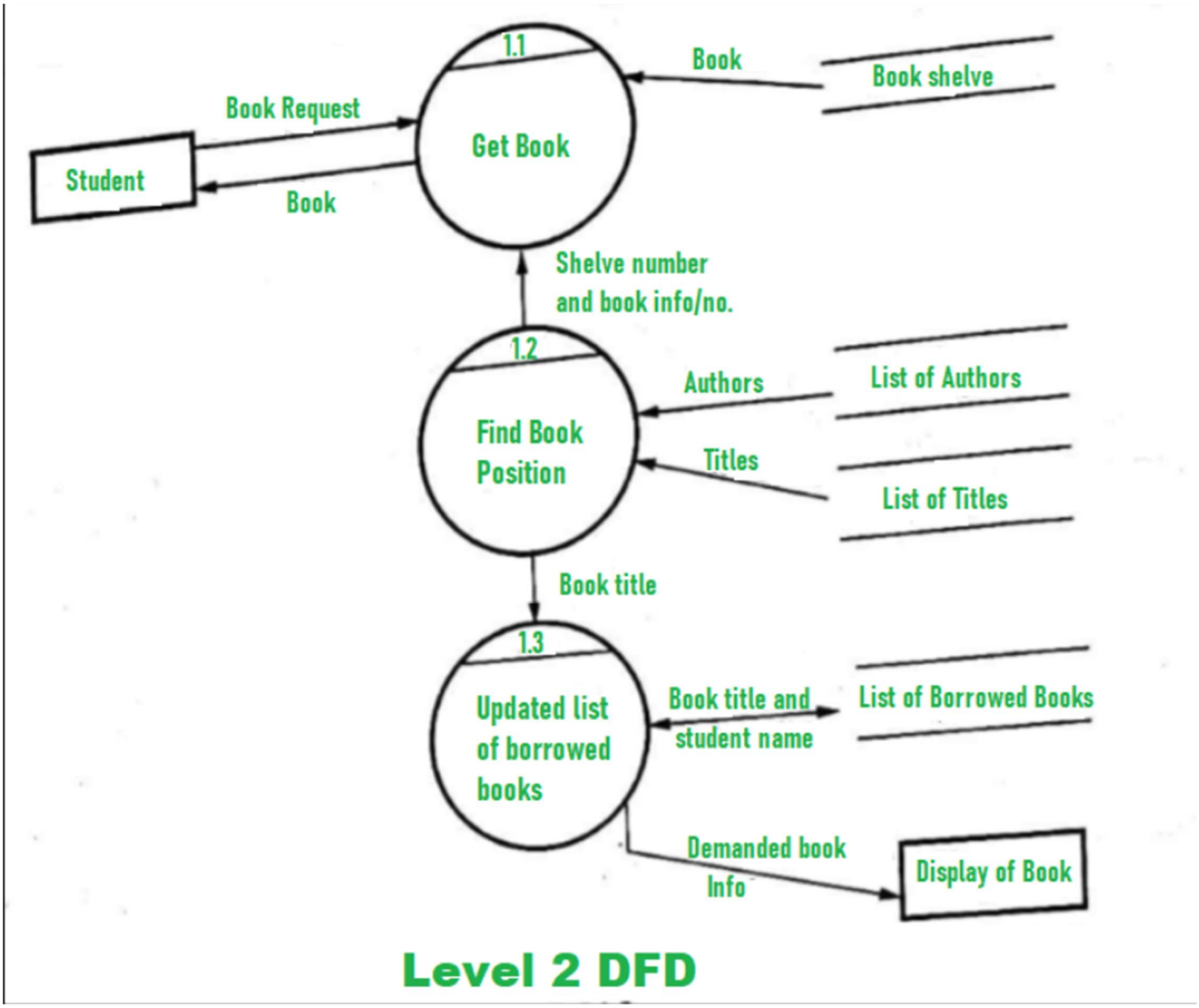
8. Design DFD for Library management system.



Level 0



Level 1 DFD


1.1
Get Book
Book
Book shelve
Book Request
Student
Book
Shelve number
and book info/no.
1.2
Find Book
Position
Authors
List of Authors
Titles
List of Titles
Book title
1.3
Updated list
of borrowed
books
Book title and
student name
List of Borrowed Books
Demanded book
Info
Display of Book

Level 2 DFD

3. Software Estimation and Metrics

1. Explain function point-based (FP) estimation technique in detail.

Function Point (FP) estimation is a widely used method for measuring the size and complexity of a software project. It is especially useful in estimating effort, cost, and time required for software development, as it focuses on the functionality delivered to the user, rather than the lines of code.

Steps Involved in FP Estimation:

1. **Identify System Components**: FP estimation involves identifying five main components of a software system:
    - **External Inputs (EI)**: User inputs into the system, like forms or data entered.
    - **External Outputs (EO)**: Outputs generated by the system, like reports or screens.
    - **External Inquiries (EQ)**: Requests for data, which require inputs and outputs but no processing.
    - **Internal Logical Files (ILF)**: Databases or files maintained within the system.
    - **External Interface Files (EIF)**: Files used by the system but maintained externally.
2. **Assign Weights to Components**: Each component is assigned a weight based on its complexity (low, average, high). The table below shows typical weights for each component:

Component	Low	Average	High
External Input (EI)	3	4	6
External Output (EO)	4	5	7
External Inquiry (EQ)	3	4	6
Internal Logical File (ILF)	7	10	15
External Interface File (EIF)	5	7	10

3. **Calculate Unadjusted Function Points (UFP)**:
    - Multiply the count of each component by its weight (based on complexity) and sum them up. This gives the **Unadjusted Function Points (UFP)**.

$$UFP = (EI \times weight) + (EO \times weight) + (EQ \times weight) + (ILF \times weight) + (EIF \times weight)$$

4. **Determine the Value Adjustment Factor (VAF)**:
    - There are 14 general system characteristics (GSCs) that are evaluated to assess the system's complexity.
    - Each characteristic is rated from 0 (no influence) to 5 (strong influence).
    - Calculate the **Total Degree of Influence (TDI)** by summing the scores.
    - The VAF is then calculated as: CAF = 0.65+(0.01×TDI)

5. **Calculate Function Point (FP)**:
    - The final **Function Point (FP)** is calculated by multiplying the **UFP** by the **VAF**.

      FP = UFP × VAF

Example of FP Estimation:

Component	Count	Complexity	Weight
External Input	5	Average	4
External Output	7	High	7
External Inquiry	4	Low	3
Internal File	3	Average	10
External File	2	High	10

- **UFP** = (5 × 4) + (7 × 7) + (4 × 3) + (3 × 10) + (2 × 10) = 20 + 49 + 12 + 30 + 20 = **131 UFP**
- Suppose the **TDI** is 35. Then, the **VAF** = 0.65 + (0.01 × 35) = **1.00**.
- **FP** = 131 × 1.00 = **131 AFP**

Advantages of FP Estimation:

- **Language-Independent**: FP estimation does not depend on the programming language, making it useful across different technologies.
- **Focus on Functionality**: It focuses on what the system does for the user rather than the internal technical details.
- **Improved Accuracy**: Provides a structured approach for estimating the effort required for a project.

Limitations:

- **Subjectivity**: Assigning weights and complexity can be subjective, leading to variations.
- **Requires Expertise**: Needs a good understanding of the system and experience with the FP method for accurate estimates.

2. What is a project and what are the different metrics used for software measurement.

In Software Engineering , a **project** is a temporary effort to develop a unique software product or system. It has specific goals, a start and end date, a defined scope, and a set of tasks to deliver a functioning software solution within a certain time frame and budget. A project involves various phases like planning, design, coding, testing, and deployment.

Different Metrics Used for Software Measurement:

Product Metrics:

1. **Size**: Measures the size of the software (e.g., Lines of Code, Function Points).
2. **Complexity**: Assesses how complex the software design and logic are (e.g., Cyclomatic Complexity).

3. **Design Features**: Evaluates specific features included in the software design.
4. **Performance**: Measures how well the system performs (e.g., speed, response time).
5. **Quality Level**: Assesses the overall quality of the software (e.g., defects, bugs).
6. **Reliability**: Measures the system's ability to perform without failure.
7. **Functionality**: Assesses whether the software meets its functional requirements.

Process Metrics:

1. **Effort Required**: Measures the amount of effort (person-hours or person-months) required.
2. **Time to Produce Product**: Assesses the time taken to develop the product.
3. **Effects of Development Techniques and Tools**: Evaluates how tools and techniques impact the development process.
4. **Number of Defects Found**: Tracks the defects identified during development or testing.
5. **Productivity**: Measures output in relation to the resources used.
6. **Failure Rate**: Tracks how often the system fails or encounters issues.
7. **Quality Efficiency**: Measures the efficiency of the process in maintaining quality.

Project Metrics:

1. **Number of Software Developers**: Tracks the total number of developers working on the project.
2. **Staffing Pattern**: Evaluates how the project team is structured and staffed.
3. **Cost**: Measures the cost of developing the software (e.g., budget, actual expenses).
4. **Schedule**: Tracks whether the project is on schedule or experiencing delays.
5. **Productivity**: Assesses the efficiency of the project team in delivering the product.

3. Explain the LOC method.

The **Lines of Code (LOC)** method is a simple software metric used to measure the size of a software project by counting the number of lines in its source code. LOC counts the number of lines written in the code. Blank lines and comments are excluded as they do not contribute to any kind of functionality.

Types of LOC:

- **Physical LOC (PLOC)**: Counts all lines, including comments and blank spaces.
- **Logical LOC (LLOC)**: Counts only lines with executable code, excluding comments and blank spaces.

Advantages:

- Simple to understand and apply.

- Helps provide a rough estimate of project size.

Limitations:

- This measure is programming language dependent.
- Sometimes, it is very difficult to estimate LOC in early stage of development.
- Though it is simple to measure, it is very hard to understand for users.
- Bad software design may cause an excessive line of code

4. Explain COCOMO model in detail.

1) **COCOMO** (Constructive Cost Model) is a regression model based on LOC, i.e. number of Lines of Code.
2) It is a cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality.
3) It was proposed by Barry Boehm in 1970 and is based on the study of 63 projects, which make it one of the best-documented models.
4) The key parameters which define the quality of any software products, which are also an outcome of the COCOMO are primarily
   i. Effort: Amount of labour that will be required to complete a task. It is measured in person-months units.
   ii. Schedule: Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put. It is measured in the units of time such as weeks, months.

There are 3 modes of development:

Organic –

1. Team size required is adequately small
2. The problem is well understood and has been solved in the past
3. The team members have a good experience regarding the problem

Semi-detached –

1. Team size, experience, and knowledge are moderate, between Organic and Embedded.
2. Projects are less familiar and more challenging than Organic, needing average experience, guidance, and creativity.

Embedded –

1. Requires high complexity, creativity, and experienced developers.
2. Needs a larger, highly skilled team for complex, innovative projects.

Types of COCOMO Models:

1. Basic COCOMO Model
2. Intermediate COCOMO Model
3. Detailed COCOMO Model

The first level, Basic COCOMO can be used for quick and slightly rough calculations of Software Costs. Its accuracy is somewhat restricted due to the absence of sufficient factor considerations.

Intermediate COCOMO takes Cost Drivers into account and Detailed COCOMO additionally accounts for the influence of individual project phases, i.e. in case of Detailed it accounts for both cost drivers and also calculations are performed phase wise henceforth producing a more accurate result.

COCOMO II MODEL

1. Developed by Barry Boehm
2. Revised version of original COCOMO
3. COCOMO was first published in 1981 so it was called as COCOMO 81.
4. In 1997 COCOMO II was developed and finally published in 2000
5. COCOMO II is the successor of COCOMO 81 and is better suited for estimating modern software development projects.
6. It provides more support for modern software development processes and an updated project database.

5. Explain 3Ps in software project spectrum.

The **3Ps** in software project management are:

1. **People**:
   - This includes everyone involved in the project, such as developers, testers, project managers, and stakeholders. The success of a project largely depends on having skilled and motivated individuals. Good communication and teamwork are essential to ensure that everyone works efficiently toward the project's goals.
2. **Process**:
   - This refers to the approach or methodology used to manage and complete the project. Examples include Agile, Waterfall, or DevOps. A well-defined process helps keep the project organized, ensuring tasks are completed on time, within budget, and meet the required quality standards.
3. **Product**:
   - The product is the software or system being developed. It should meet the needs of the users, function as expected, and be of high quality. The focus is on building a reliable and functional product that satisfies the customer’s requirements.

6. Explain project scheduling and tracking.

1. Project Scheduling:

- **Project scheduling** involves creating a timeline for all the tasks and activities in the project. It defines what needs to be done, by whom, and when it should be completed.
- **Steps in Project Scheduling**:
    1. **Break down tasks**: Divide the project into smaller, manageable tasks or phases.
    2. **Assign resources**: Allocate team members and resources to each task.
    3. **Define timelines**: Set deadlines for each task based on priorities and dependencies.
    4. **Create a project plan**: Use tools like Gantt charts or PERT charts to visualize the schedule.

The goal is to ensure the project stays on track, and all tasks are completed within the available time and resources.

2. Project Tracking:

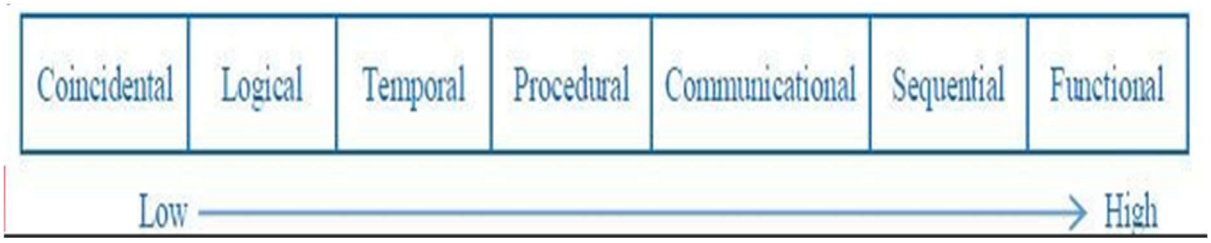
- **Project tracking** is the process of monitoring the progress of the project against the schedule. It helps ensure that the project is moving according to plan and allows for adjustments if delays or issues arise.
- **Steps in Project Tracking**:
    1. **Monitor task completion**: Regularly check if tasks are being completed on time.
    2. **Track resource usage**: Ensure that team members and resources are being used efficiently.
    3. **Identify risks**: Watch for potential problems or delays and address them early.
    4. **Update the schedule**: Adjust the timeline, if necessary, based on the project's progress.

4. Software Design

1. Explain Cohesion and Coupling. Explain different types of Cohesion and Coupling.

Cohesion

- The measure of how strongly the elements are related functionally inside a module is called cohesion. A good software design will have high cohesion.
- The elements inside a module can be instructions, groups of instructions, definition of data, call from another module etc.
- The aim is always for functions that are strongly related and the expectation is for everything inside the module to be in connection with one another.
- Cohesion is a measure of functional strength of a module.



Types of Cohesion

1. **Functional Cohesion**: All elements contribute to a single, well-defined task. This is the highest level of cohesion.
   Example: A function that calculates the area of different geometric shapes.
2. **Sequential Cohesion**: The output of one element serves as the input for another element.
   Example: A process where data is retrieved from a database, processed, and then displayed.
3. **Communicational Cohesion**: Elements operate on the same data or contribute to a common task but do not depend on each other in a sequence.
   Example: A function that calculates statistics like mean and median from a given dataset.
4. **Procedural Cohesion**: Elements are grouped because they always follow a specific sequence of execution.
   Example: A function that performs multiple steps like initializing, processing, and finalizing a transaction.
5. **Temporal Cohesion**: Elements are grouped by when they are executed, often in a specific timeframe.
   Example: Initialization functions that run at the start of a program.
6. **Logical Cohesion**: Elements are grouped because they are related logically, even if they operate independently.
   Example: A function that handles various types of input events (keyboard, mouse, etc.).
7. **Coincidental Cohesion**: The lowest level of cohesion, where elements are grouped arbitrarily without any meaningful relationship.
   Example: A utility module with unrelated functions.

Coupling

- Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling.
- The lower the coupling, the more modular a program is, which means that less code has to be changed when the program's functionality is altered later on.
- However, coupling cannot be completely eliminated; it can only be minimized

Types of Coupling

1. **Content Coupling**: One module directly changes another module's data.
   Example: Module A directly changing a variable in Module B.
2. **Common Coupling**: Multiple modules share the same global data, leading to unpredictability.
   Example: Several modules modifying a global variable.
3. **External Coupling**: Modules depend on externally defined data formats or protocols.
   Example: A module that requires a specific file format to function.
4. **Control Coupling**: One module controls the behaviour of another by passing control information.
   Example: A function that decides which algorithm to use based on a passed parameter.
5. **Stamp Coupling**: Modules share a data structure but only use parts of it.
   Example: Two functions using attributes from a complex object.
6. **Data Coupling**: Modules share data by passing parameters without affecting each other.
   Example: Two functions that exchange data through simple parameters.
7. **Message Coupling**: The lowest form of coupling, where modules communicate through messages or events.
   Example: Objects communicating through events in an event-driven system.

2. What are the software design principles. Explain in detail with examples.

(For 5 marks just write the points below, for 10 marks if asked to illustrate write with examples)

1. Avoid Tunnel Vision in the Design Process
2. Ensure Traceability to the Analysis Model
3. Reuse Existing Solutions and Avoid Reinventing the Wheel
4. Minimize Intellectual Distance Between Software and Real-World Problems
5. Promote Uniformity and Integration in Design
6. Design for Flexibility and Accommodate Change
7. Structure Design to Handle Errors Gracefully
8. Differentiate Between Design and Coding
9. Evaluate Design Quality Throughout the Process
10. Review Design to Minimize Conceptual Errors

1. Avoid Tunnel Vision

- **Principle**: It should not only focus on completing or achieving the aim but on other effects as well.
- **Example**:
    - If you're building an online store, instead of just using one type of database, compare different databases to see which one works best for your needs.

2. Traceability to the Analysis Model

- **Principle**: It should satisfy all the requirements that software requires to develop a high-quality product.
- **Example**:
    - If the requirement is a "Login feature," your design should include a login screen, a way to check the user's info, and store data about the users.

3. Reuse and Avoid Reinventing the Wheel

- **Principle**: It should not waste time or effort in creating things that already exist.
- **Example**:
    - If you need user authentication, instead of building it from scratch, use a library like **OAuth** that already does that.

4. Minimize Intellectual Distance

- **Principle**: Reduce the gap between real-world problems and software solutions.
- **Example**:
    - For a library system, use classes like Book, Member, and Loan Record, which are easy to understand because they’re real-world things.

5. Promote Uniformity and Integration

- **Principle**: It should mix or combine all parts of software i.e. subsystems into one system.
- **Example**:
    - If you are using the **MVC** pattern in one part of the project, use it everywhere to keep things consistent.

6. Design for Change

- **Principle**: The software should adjust to the change that is required to be done as per the user’s need.
- **Example**:
    - If you’re adding payment options to a website, instead of hardcoding just one type, create a structure where you can easily add more payment methods later.

7. Graceful Degradation

- **Principle**: It should work properly even if an error occurs during the execution
- **Example**:
    - If a payment fails, show a friendly message and try again, instead of crashing the whole system.

8. Design Is Not Coding and Coding is not Design

- **Principle**: Design means describing the logic and coding is a type of language, both should be differentiated
- **Example**:
    - Before writing code, sketch out how the system will work, using diagrams to plan interactions between different parts of the software.

9. Assess Quality During Design

- **Principle**: During the evaluation, the quality of the design needs to be checked and focused on.
- **Example**:
    - If your design includes a search function, check if it's efficient. If you're searching through sorted data, using **binary search** will be faster than just going through each item one by one.

10. Review to discover errors

- **Principle**: The overall evaluation should be done to check if there is any error present.
- **Example**:
    - If the requirement says to automatically reorder products when stock is low, make sure your design includes this feature so that nothing is missed.

3. What is User interface design? How does it help web technology and IT industry?

User Interface (UI) design is the process of creating interfaces in software or computerized devices with a focus on maximizing usability and the user experience. It involves designing all the elements that users interact with, including screens, buttons, icons, menus, and other visual elements. The goal is to create intuitive, aesthetically pleasing, and functional interfaces that allow users to interact effectively and efficiently with the software or system.

Key Components of User Interface Design

1. **Visual Design**: Involves the aesthetics of the interface, including colour schemes, typography, layout, and overall look and feel.
2. **Interaction Design**: Focuses on how users interact with the interface, including navigation, buttons, and user feedback.
3. **Usability**: Ensures that the interface is easy to use and understand, helping users achieve their goals with minimal frustration.

4. **Accessibility**: Designs interfaces that are usable for people with disabilities, ensuring that everyone can access and use the system.
5. **Consistency**: Uses consistent design patterns and elements throughout the application to make it more familiar and easier to navigate for users.

How User Interface Design Help Web Technology and the IT Industry

1. **Improved User Experience**: A well-designed user interface enhances the overall user experience by making software and applications easier to navigate and use, leading to higher user satisfaction.
2. **Increased Productivity**: Intuitive interfaces reduce the time users spend trying to figure out how to use the system, allowing them to accomplish tasks more efficiently.
3. **Higher Engagement**: A visually appealing and interactive interface can engage users more effectively, leading to longer usage times and better retention rates for web applications.
4. **Reduced Learning Curve**: Good UI design minimizes the amount of training or instruction users need, allowing them to start using the software quickly.
5. **Brand Identity**: UI design plays a crucial role in establishing a brand's identity. Consistent and attractive design elements can make a brand more recognizable and trustworthy.

4. Design user interface for Online shopping system.

5. Software Testing

1. What is software testing? Explain different types of software testing.

- **Testing** is the process of executing a program with the aim of finding errors.
- To make our software perform well it should be error-free.
- Testing is conducted at the phase level in software development life cycle or at module level in program code.
- Software testing comprises of Validation and Verification.
- **Validation** is process of examining whether or not the software satisfies the user requirements.
- **Verification** is the process of confirming if the software is meeting the business requirements and is developed adhering to the proper specifications and methodologies.

Types of Testing:

1. Unit testing:

- A unit is a single testable part of a software system and tested during the development phase of the application software.
- Unit testing involves the testing of each unit or an individual component of the software application.
- It is the first level of functional testing.
- The aim behind unit testing is to validate unit components with its performance.

Example: Testing a function that calculates the sum of two numbers to ensure it returns the correct result.

2. Integration testing:

- Once all the modules have been unit tested, integration testing is performed.
- Integration testing is the process of testing the interface between two software units or module.
- It's focus on determining the correctness of the interface.
- The purpose of the integration testing is to expose faults in the interaction between integrated units.

Example: Verifying that the login module integrates correctly with the database module to authenticate a user.

3. Validation testing:

- When integration testing ends, Validation testing begins.
- Validation Testing is a type of software testing used to ensure that the software product actually meets the specific requirements and expectations of the end-users or clients.
- It checks if the "right product" has been built, focusing on validating the software against its intended use.

Example:

If a client requires a payment gateway in an e-commerce website, validation testing would involve ensuring that:

The payment process works smoothly.

The system correctly handles transactions.

4. System testing:

- System Testing is a type of software testing where the entire integrated system is tested as a whole to verify that it meets specified requirements.
- It's conducted after integration testing and before acceptance testing, focusing on the complete and integrated software system to ensure that it works as intended.
- System testing validates both functional and non-functional aspects, such as performance, reliability, and security.

Example: Checking that users can log in, view account details, transfer funds, and make payments as expected.

2. Differentiate between White box testing & Black box testing.

Aspect	White Box Testing	Black Box Testing
Focus	Tests the internal code and logic.	Tests the software's functionality.
Knowledge Required	Requires understanding of the code.	No knowledge of the code is needed.
Test Basis	Based on the code structure.	Based on requirements and user needs.
Tester Role	Usually done by developers.	Usually done by testers or users.
Testing Methods	Looks at specific code paths (e.g., loops, conditions).	Focuses on functionality and outputs.
Type of Testing	Also called **Clear Box** or **Glass Box** testing.	Also called **Behavioral Testing**.
Focus on	Code errors, logic, and conditions.	User input, output, and interface behavior.
Use Cases	Used in unit testing and debugging.	Used in system and acceptance testing.
Advantages	Finds bugs in code early.	Ensures the software works as expected.
Disadvantages	Can't test user needs or UI.	Can't find code-level issues.
Example	Using **Control Flow Testing** or **Path Testing** to verify loops and conditions in the code work correctly.	Using **Equivalence Partitioning** or **Boundary Value Analysis** to test if a login form accepts correct credentials and rejects wrong ones.

3. Explain the software testing process.

1. **Requirement Analysis:** The testing team reviews requirements, specifications, and design documents to understand what needs to be tested. This ensures that all requirements are clear, testable, and aligned with the client's expectations.
2. **Test Planning:** A test plan is created, defining the testing strategy, scope, schedule, resources, and potential risks. This step sets the foundation for testing and helps in organizing the entire process.
3. **Test Case Development:** Test cases are designed based on requirements. Each test case includes inputs, expected results, and any preconditions needed. High-priority test cases are identified to ensure critical features are tested first.
4. **Test Environment Setup:** The testing environment is prepared with the necessary hardware, software, and configurations to mirror the production environment. Test data is also set up at this stage.
5. **Test Execution**: Test cases are run, and actual outcomes are compared with expected results. Any discrepancies or bugs are logged for further investigation.
6. **Defect Reporting and Tracking**: Identified defects are documented, prioritized, and reported to the development team. Each defect is tracked until it is resolved to ensure no issues remain unaddressed.
7. **Test Closure:** Once testing objectives are met, a summary report is prepared. The team reviews the testing process, captures lessons learned, and closes the testing phase.

4. Explain different techniques in White box testing.

White Box Testing is software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability and security.

In white box testing, code is visible to testers so it is also called Clear box testing, Open box testing, and Glass box testing.

White box testing techniques:

1) Statement Coverage

- Statement Coverage is a white box testing technique in which all the executable statements in the source code are executed at least once.
- It is used for calculation of the number of statements in source code which have been executed.
- The main purpose of Statement Coverage is to cover all the possible paths, lines and statements in source code.

$$\text{Statement Coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} \times 100$$

2) Decision Coverage

- Decision Coverage is a white box testing technique which reports the true or false outcomes of each Boolean expression of the source code.

- The goal of decision coverage testing is to cover and validate all the accessible source code by checking and ensuring that each branch of every possible decision point is executed at least once.
- In this coverage, expressions can sometimes get complicated. Therefore, it is very hard to achieve 100% coverage.

$$Decision\ Coverage = \frac{Number\ of\ Decision\ Outcomes\ Excercised}{Total\ Number\ of\ Decision\ Outcomes}$$

3) Branch Coverage

- Branch Coverage is a white box testing method in which every outcome from a code module(statement or loop) is tested.
- The purpose of branch coverage is to ensure that each decision condition from every branch is executed at least once.
- It helps to measure fractions of independent code segments and to find out sections having no branches.

$$Branch\ Coverage = \frac{Number\ of\ Executed\ Branches}{Total\ Number\ of\ Branches}$$

5. Explain different techniques in Black box testing.

Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths.

Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioural Testing

Black box testing techniques:

1) Boundary Value Analysis

- It is a widely used black box testing testing, which is also the basis for equivalence testing.
- Boundary value analysis tests the software with test cases with extreme values of test data.
- BVA is used to identify the flaws or errors that arise due to the limits of input data.
- For example: Taking inputs for a test case data for an age section should accept a valid data of anything between 1-100. According to BVP analysis, the software will be tested against four test data as -1, 1, 100, and 101 to check the system's response using the boundary values.
-

2) State Transition Testing

- This testing technique uses the inputs, outputs, and the state of the system during the testing phase.
- It checks the software against the sequence of transitions or events among the test data.

3) Decision Table Testing

- This approach creates test cases based on various possibilities.
- It considers multiple test cases in a decision table format where each condition is checked and fulfilled, to pass the test and provide accurate output. It is preferred in case of various input combinations and multiple possibilities.
- For example, A food delivery application will check various payment modes as input to place the order decision making based on the table.
    - Case 1: if the end-user has cash, the action will be taken.
    - Case 2: If the end-user doesn't have anything, then action will not be taken.

4) Graph-Based Testing

This technique of Black box testing involves a graph drawing that depicts the link between the causes (inputs) and the effects (output), which trigger the effects. This testing utilizes different combinations of output and inputs.

5) Error guessing technique

This method of designing test cases is about guessing the output and input to fix any errors that might be present in the system. It depends on the skills and judgment of the tester.

6. Differentiate between Alpha & Beta testing.

Parameter	Alpha Testing	Beta Testing
Performed by	Alpha testing is done by testers who are generally employees of the organization.	Beta testing is done by end users who are not employees of the organization.
Testing environment	Alpha testing is done in a lab environment.	Beta testing is done in a real-world environment.
Factors tested	Security testing and reliability are not tested in depth during alpha testing.	Reliability, security, and robustness are verified during beta testing.
Included testing type	Alpha testing includes white box and black box testing.	Beta testing includes black box testing.
Time required	Alpha testing requires a lot of time to perform because it includes both white box and black box testing.	Few weeks of time are required to perform beta testing.
Fixing of issues	Important issues can be corrected by developers without delay in alpha testing.	Issues detected in beta testing will be corrected in future versions of the software product.
Focus	Alpha testing focuses on giving assurance about the quality of the software product before beta testing.	Beta testing also focuses on quality but collects feedback from end users to ensure readiness for real-world use.

7. What is maintenance? Explain the different types of maintenance.

Software Maintenance

- **Purpose**: Part of the Software Development Life Cycle, it involves modifying software post-delivery to fix issues and improve performance.
- **Adaptability**: Software reflects real-world needs, so changes are required when those needs evolve.
- **Activities**: Includes error correction, feature removal, optimization, and enhancements.
- **Planning**: Effective maintenance needs a plan during development to manage costs, which can be 40-80% of the project budget.
- **Reasons for Modification**:
    - **Market Conditions**: New policies or regulations, like tax changes, may require updates.
    - **Client Requests**: Clients may request new features over time.
    - **Host Changes**: Software may need updates to work on new hardware or operating systems.
    - **Organizational Changes**: Business changes, like mergers or restructuring, may require software adjustments.

Types of Maintenance:

Corrective Maintenance:

- Focuses on fixing faults or defects found in daily operations, caused by errors in software design, logic, or coding.
- **Design errors** stem from incorrect, incomplete, or misunderstood changes.
- **Logical errors** arise from faulty logic, invalid tests, or incomplete data checks.
- Accounts for about 20% of all maintenance activities.

Adaptive Maintenance:

- Involves adjusting parts of the software to accommodate changes in the environment, like hardware or operating systems.
- Accounts for around 25% of maintenance activities.

Perfective Maintenance:

- Addresses new or changed user requirements, improving both functionality and performance, even without faults.
- Enhances code efficiency and adjusts features to meet evolving user needs.
- Makes up 50% of all maintenance, the largest portion.

Preventive Maintenance:

- Aims to prevent future errors by simplifying code, optimizing, and updating documentation to improve maintainability.
- Managed internally, without external requests, and accounts for 5% of maintenance activities.

8. Explain software re-engineering.

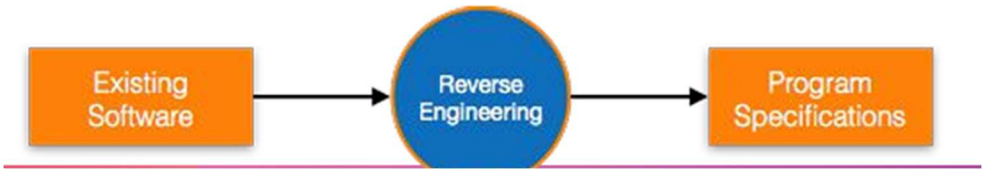
- When we need to update the software to keep it to the current market, without impacting its functionality, it is called **software re-engineering**. It is a thorough process where the design of software is changed and programs are re-written.
- Legacy software cannot keep tuning with the latest technology available in the market. As the hardware become obsolete, updating of software becomes a headache. Even if software grows old with time, its functionality does not.
- For example, initially Unix was developed in assembly language. When language C came into existence, Unix was re-engineered in C, because working in assembly language was difficult.
- Other than this, sometimes programmers notice that few parts of software need more maintenance than others and they also need re-engineering.

Re-engineering process:

- Decide what to re-engineer. Is it the whole software or a part of it?
- Perform Reverse Engineering, in order to obtain specifications of existing software.
- Restructure Program if required. For example, changing function-oriented programs into object-oriented programs.
- Re-structure data as required.
- Apply Forward engineering concepts in order to get re-engineered software.

9. Explain software reverse engineering.

1) It is a process to achieve system specification by thoroughly analysing, understanding the existing system.
2) This process can be seen as reverse SDLC model, i.e. we try to get higher abstraction level by analysing lower abstraction levels.
3) An existing system is previously implemented and designed, which we know nothing about.
4) Designers then do reverse engineering by looking at the code and try to get the design. With design in hand, they try to conclude the specifications. Thus, going in reverse from code to system specification



Purpose of Reverse engineering:

i. Security analysis

ii. Enabling additional features
iii. Used as learning tools

Process of reverse engineering in the following steps:

1. **Refine Code**: Start with the raw source code and clean it to remove unnecessary parts, making it easier to read (resulting in clear source code).
2. **Extract Abstractions**: Analyse the clear source code to identify key parts like processing logic, user interface, and database structure.
3. **Create Initial Specification**: Combine these abstractions to develop an initial high-level description of the software's design.
4. **Refine and Simplify**: Improve the initial specification to make it clearer and more precise.
5. **Final Specification**: The result is a refined, well-organized description of the software, making it easier to understand and modify.

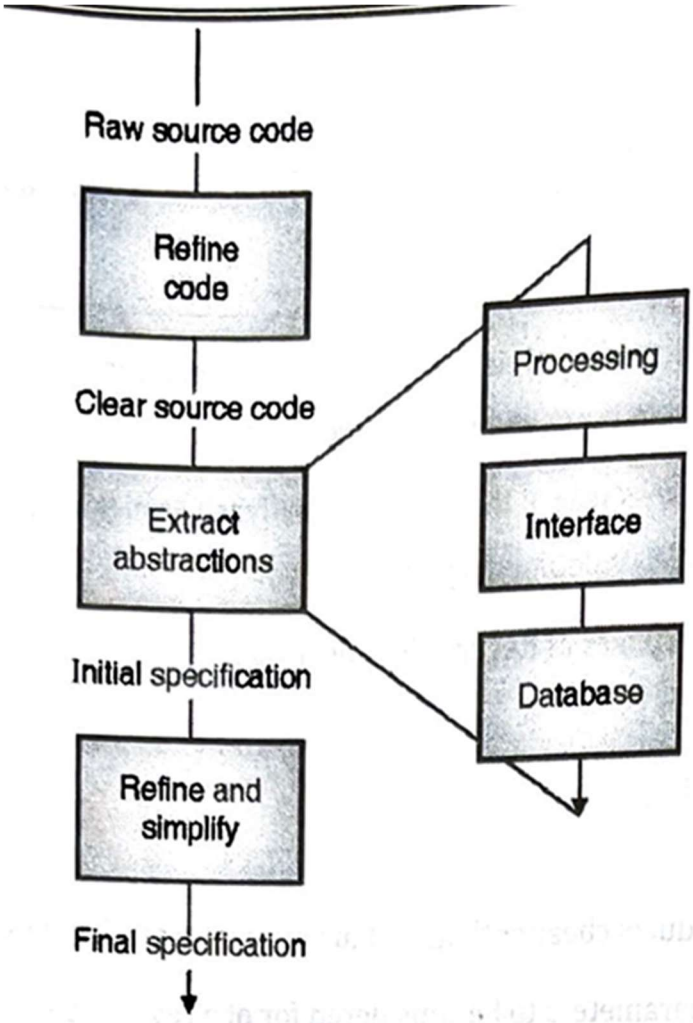


Fig. 10.3.1 : A Reverse Engineering Process

10. Design the test cases for Medical Management Application.

1. Login and Authentication

- **Test Case**: Verify login with valid credentials.
  - **Expected Result**: User successfully logs into the system.
- **Test Case**: Verify login with invalid credentials.
  - **Expected Result**: User sees an error message and cannot log in.

2. Patient Registration

- **Test Case**: Verify new patient registration with all mandatory fields.
  - **Expected Result**: New patient is successfully registered and added to the system.
- **Test Case**: Verify that duplicate patient entries are not allowed.
  - **Expected Result**: System prevents duplicate entries with a warning.

3. Appointment Scheduling

- **Test Case**: Verify appointment creation for an existing patient.
  - **Expected Result**: Appointment is successfully scheduled and reflected on the patient's profile.
- **Test Case**: Verify appointment cancellation.
  - **Expected Result**: Appointment is marked as cancelled and removed from the schedule.

4. Prescription Management

- **Test Case**: Verify prescription creation for a patient.
  - **Expected Result**: Prescription is saved and linked to the patient's record.
- **Test Case**: Verify editing an existing prescription.
  - **Expected Result**: Prescription is updated successfully.

5. Billing and Payment

- **Test Case**: Verify bill generation for a patient's consultation or treatment.
  - **Expected Result**: Correct bill is generated with accurate pricing.
- **Test Case**: Verify payment processing.
  - **Expected Result**: Payment is processed, and status is updated in the system.

6. Software Configuration Management, Quality Assurance and Maintenance

1. Define risk. What are the different categories of risk.

A **risk** is a potential problem that could negatively impact the project's success, such as delaying delivery, increasing costs, or reducing the quality of the product. Risks can be uncertain events or conditions that, if they occur, may affect the project's objectives.

1) **Schedule Risks**:
   - Risks that could delay the project timeline, often resulting in missed deadlines and potentially cascading delays.
2) **Budget Risks**:
   - Risks related to project costs, where the project may exceed its allocated budget.
3) **Operational Risks**:
   - Risks associated with the day-to-day operations of the project, including resource management and process issues.
4) **Technical Risks**:
   - Risks tied to the technology and tools used in the project, which can impact performance, functionality, or security.
5) **Other Unavoidable Risks (Programmatic Risks)**:
   - Also called programmatic risks, these are risks that stem from external factors and are often out of the project team's control.

2. Explain RMMM plan with suitable example.

RMMM stands for Risk Mitigation, Monitoring and Management. It is a framework that helps organizations assess and improve their risk management practices over time.

Risk Mitigation (Avoidance): This means taking steps to prevent employee turnover by fixing issues we can control, like improving pay and working conditions. We also prepare for when people do leave by sharing information among team members, setting up clear documentation guidelines, and having backup staff ready for key roles.

Risk Monitoring (Tracking): Here, project managers keep an eye on how the team feels, how well they're working together, and if there are any problems with pay or job opportunities. This helps spot potential issues early.

Risk Management (Dealing): If employees leave despite our efforts, we use our backups and make sure everything is documented and shared. The project manager might need to rearrange resources and schedules to focus on parts of the project that are fully staffed, helping new team members catch up quickly.

RMMM Plan Example:

Project Name: Online Learning Platform

1. Monitoring:

- Check team morale monthly through surveys.
- Review budget spending weekly to spot any issues early.

2. Management:

- Assign a team leader to oversee progress and address problems.
- Hold regular meetings to discuss any risks and updates.

3. Mitigation:

- If team morale drops, plan team-building activities.
- If the budget is tight, look for cheaper marketing options or partnerships.

3. Explain Software Quality Assurance(SQA) in detail.

Software Quality Assurance (SQA) is a systematic process that ensures software products meet specified requirements and standards throughout their development lifecycle. SQA encompasses a range of activities designed to improve and verify the quality of software.

Goals of SQA:

- Ensure software follows set standards and processes.
- Identify and reduce risks early in development.
- Improve the development process to minimize defects.

Roles and Responsibilities in SQA:

i. The software engineers who do technical work.
ii. An SQA group that has responsibility for quality assurance planning, record keeping, analysis, and reporting.

SQA Activities:

1. **Prepare an SQA Plan:**
    - Develop the plan during project planning, involving all stakeholders.
    - The plan outlines quality assurance activities for the software team and SQA group.
    - It specifies calculations, audits, applicable standards, error reporting methods, documents to be produced, and feedback levels.
2. **Participate in Software Process Development:**
    - The software team chooses a process for the project.
    - The SQA group reviews this process to ensure it aligns with organizational policies and standards (like ISO-9001).

3. **Review Software Engineering Activities:**
    - The SQA group checks compliance with the defined software process.
    - They report and track any deviations and verify corrections are made.
4. **Audit Software Work Products:**
    - Review selected work products for compliance with the software process.
    - Document deviations, ensure corrections, and report results to the project manager.
5. **Document and Handle Deviations:**
    - Record any deviations in the project method, process, or standards.
    - Ensure deviations are managed according to documented procedures.
6. **Track Noncompliance:**
    - Report noncompliance issues to senior management.
    - Track these issues until they are resolved.

4. Explain FTR.

Formal Technical Review (FTR) is an activity performed by software engineers to give assurance of software quality.

Objectives of Formal Technical Review (FTR)

1. **Error Detection:** Uncover errors in logic, functionality, and implementation in any representation of the software.
2. **Requirements Verification:** Ensure that the software meets the specified requirements.
3. **Standards Compliance:** Confirm that the software is developed according to predefined standards.
4. **Uniformity Review:** Assess the consistency in the software development process to promote uniformity.
5. **Project Manageability:** Help make the project easier to manage by identifying potential issues early.

Steps in Formal Technical Review (FTR)

1. **Review Meeting:**
    - Involve 3 to 5 participants.
    - Preparation should take no more than 2 hours for each person.
    - Keep the meeting duration under 2 hours, focusing on a specific part of the software.
    - At the end of the meeting, attendees decide to:
        - Accept the product without modifications.

        - Reject the product due to serious errors (requiring another review after corrections).
        - Accept the product provisionally (minor errors noted, no further review needed).
    - All attendees sign off to indicate their participation and agreement with the findings.
2. **Review Reporting and Record Keeping:**
    - Record all raised issues during the FTR.
    - Consolidate issues into a review list and prepare a summary report.
    - The report should answer:
        - What was reviewed?
        - Who reviewed it?
        - What were the findings and conclusions?

5. Compare FTR and Walkthrough.

Parameter	FTR (Formal Technical Review)	Walkthrough
Concept	Formal review process to ensure software quality	Informal review meeting, not a formal process
Performer	Conducted by software engineers and other reviewers	Usually initiated by the author of the code
Process	Work product inspected for defects by reviewers other than the creator	Creator presents product, participants provide comments
Work Product	Key deliverables from software development phases	Primarily used to examine source code with a line-by-line review
Way to Perform	Done as a peer review, typically without management involvement	Useful for reviewing high-level documents (e.g., requirements, architecture)
Objectives	Find defects, ensure standards are met	Share understanding, give feedback
Documentation	Findings are documented with formal reports	Minimal documentation, often just informal notes
Frequency	Conducted at key stages in development	Frequent, especially for code and early drafts
Outcome	Report with issues and follow-up actions	Feedback for creator; fixes often at their discretion

6. Explain Walkthrough

i. Walkthrough is a method of conducting informal group/individual review.
ii. In a walkthrough, author describes and explain work product in a informal meeting to his peers or supervisor to get feedback.
iii. Here, validity of the proposed solution for work product is checked.
iv. It is cheaper to make changes when design is on the paper rather than at time of conversion.
v. Walkthrough is a static method of quality assurance.
vi. Walkthrough are informal meetings but with purpose

7. Explain Software configuration Management (SCM) process.

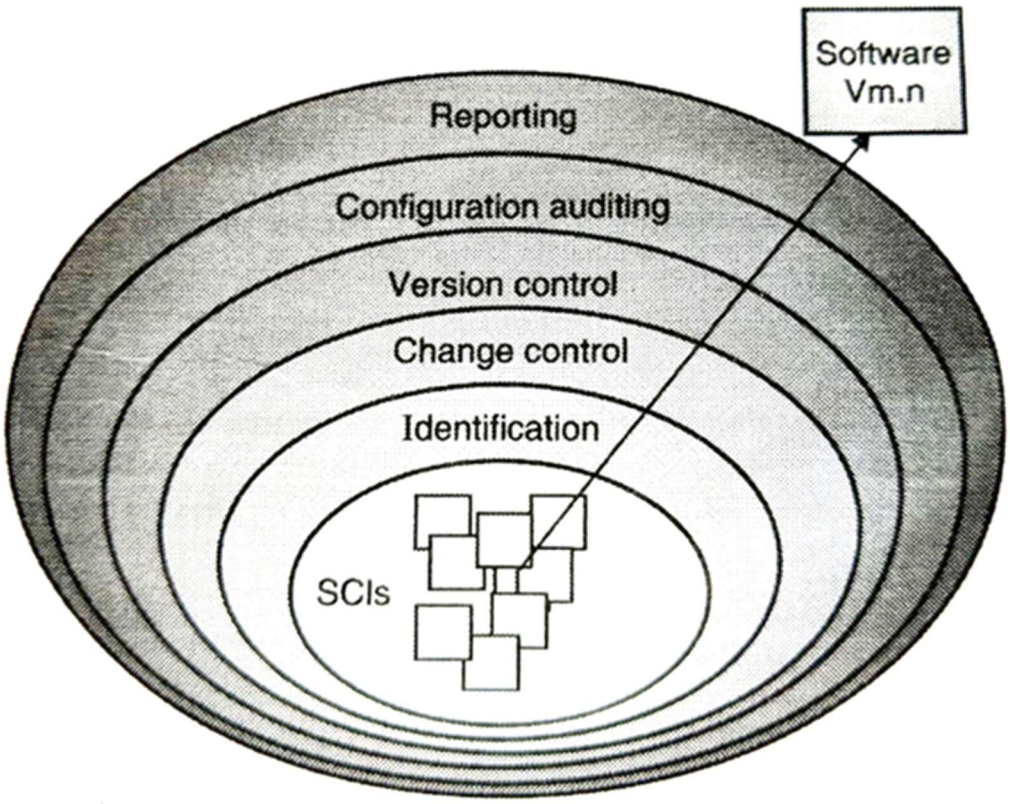
In Software Engineering, Software Configuration Management(SCM) is a process to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle. The primary goal is to increase productivity with minimal mistakes.

SCM PROCESS:

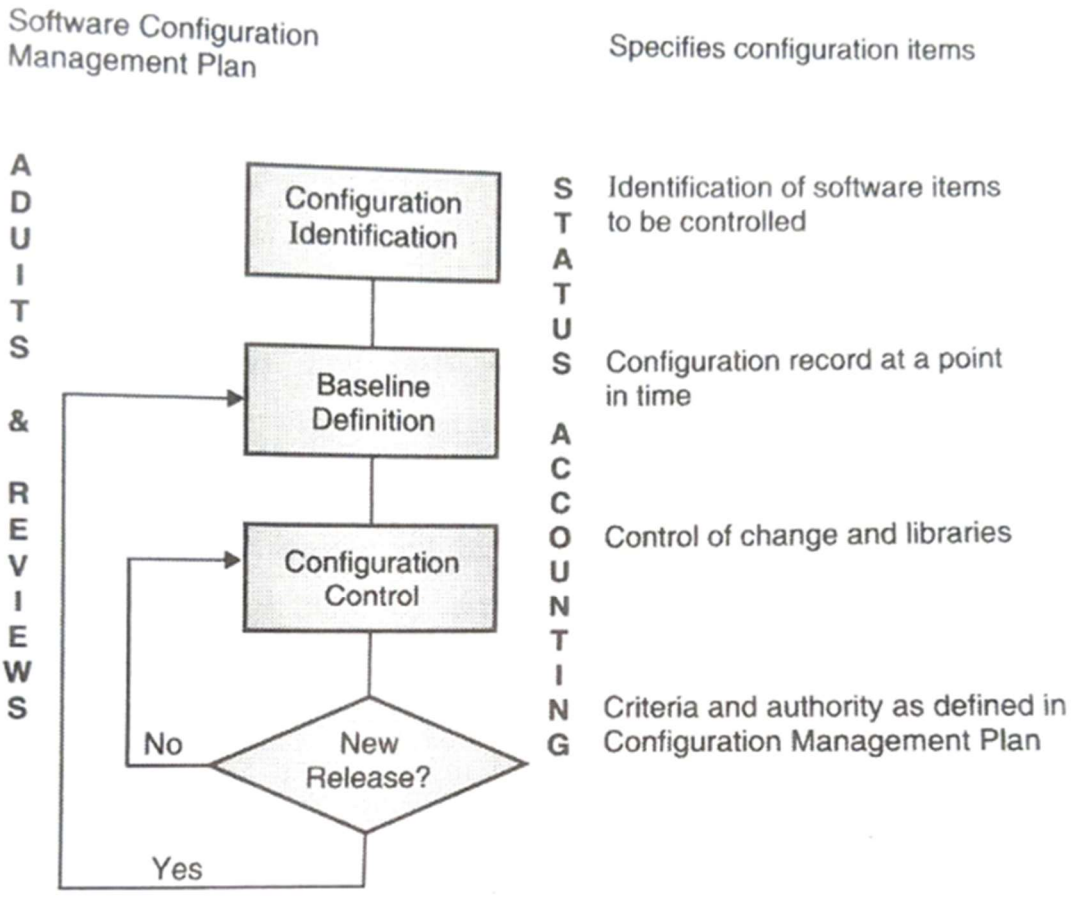
The SCM process defines a sequence of tasks which have 4 most important objectives:

1) To identify all of the respective items which jointly define the software configuration,
2) To manage changes applied to these items,
3) To ease the building of different versions of an application, and
4) To make it sure that there is consistency in software quality as the configuration evolves over time.

Tasks of SCM:



High level view of SCM:



1. **SCM Plan**
    - Defines guidelines and procedures for managing configuration items, change control, and audits.
2. **Configuration Identification**
    - Identifies all items to be controlled (e.g., code files, documents) and assigns unique identifiers for tracking.
3. **Baseline Definition**
    - Establishes a stable version (baseline) of the software, serving as a reference for future changes.
4. **Configuration Control**
    - Manages and documents authorized changes to configuration items, ensuring consistency.
5. **Status Accounting**
    - Tracks the current status and history of each configuration item for transparency and traceability.
6. **Audits and Reviews**
    - Verifies that configuration items and changes meet quality standards and comply with the SCM plan.

7. **Release Decision (New Release?)**
    - Assesses if the software meets all criteria for a new release; if so, it proceeds to release, if not, it returns for revisions.

8. Explain change control and version control. How is change control different than version control.

CHANGE CONTROL

Changes occur at all phases of the software lifecycle due to evolving requirements or identified deficiencies. Change control is a systematic process used in software engineering to manage changes to a project throughout its lifecycle

- **Testing**: Testing may reveal defects that necessitate updates to code, design, or requirements.
- **Version Management**: Changes must be made to the correct code version, with testing to ensure performance and integrity, and documentation must be updated accordingly.
- **Change Request Process**: A structured mechanism is needed to handle change requests (e.g., discrepancies, failure reports, new features) from various sources during development and support.
- **Flexibility**: No single procedure fits all change management scenarios; adaptability is essential.
- **Processing Activities**:
    - Define the information needed for change approval.
    - Identify the review process and information routing.
    - Control the libraries involved in change processing.
    - Develop procedures for implementing changes in code, documentation, and the final product.

VERSION CONTROL

Version control is a systematic process that integrates procedures and tools to manage different versions of configuration objects generated throughout the software development lifecycle.

Key capabilities of a version control system include:

- A project database (repository) which holds all the appropriate configuration objects.
- A version management capability which holds all the relevant versions of a configuration object.
- A make facility which helps to get all respective configuration objects and build a particular version of the software

Issue Tracking: Many version control systems include features for tracking issues and bugs associated with configuration objects.

Change Sets: The ability to create named change sets that specify the changes needed to build a particular version of the software, applied to a baseline configuration.

System Modelling: Employs a model that includes:

1. A template with a component hierarchy and build order.
2. Construction rules.
3. Verification rules.

Recent automated approaches to version control vary in sophistication, focusing on attributes that facilitate the building of specific software versions.

Difference between Change & Version control:

Aspect	Change Control	Version Control
Purpose	Manages *what* and *why* changes happen in a project.	Manages *how* changes to files and code are stored.
Focus	Decides if changes should happen.	Tracks and organizes all file versions.
Scope	Applies to project requirements, design, and plans.	Applies only to files and code.
Process	Involves approvals, impact analysis, and documentation updates.	Involves saving, branching, and merging code versions.