SOFWARE ENGINEERING NOTES BY FAIZ 🤫💁

SOURCES OF NOTES:

CHATGPT, GEEKS FOR GEEKS, TECH KNOWLEDGE

MODULE 1: Introduction to SE and Process model

Q.1 What is Software Process Framework ?

=>
A Software Process Framework is a structured approach that defines the steps, tasks, and activities involved in software development.

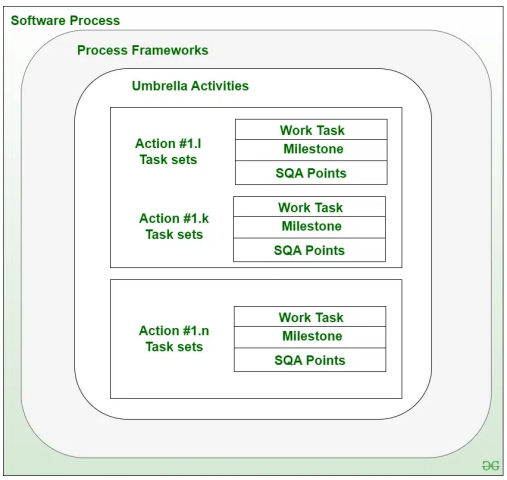
This framework serves as a foundation for software engineering, guiding the development team through various stages to ensure a systematic and efficient process.

A Software Process Framework helps in project planning, risk management, and quality assurance by detailing the chronological order of actions.

It includes task sets, umbrella activities, and process framework activities, all essential for a successful software development lifecycle.

Utilising a well-defined Software Process Framework enhances productivity, consistency, and the overall quality of the software product.

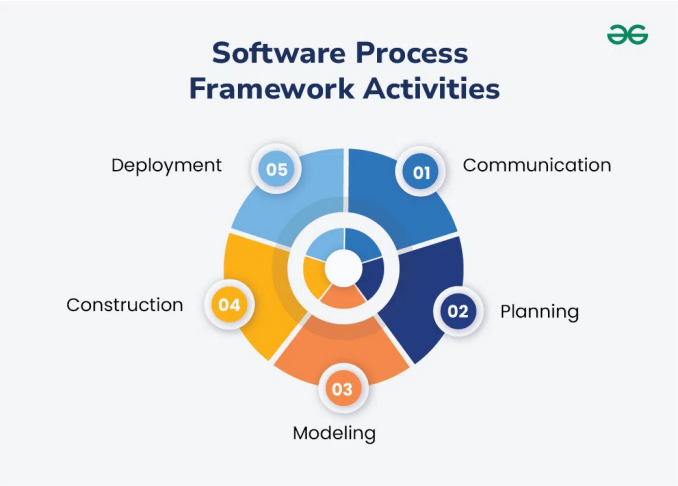
Software Process includes:
Tasks: They focus on a small, specific objective.
Action: It is a set of tasks that produce a major work product.
Activities: Activities are groups of related tasks and actions for a major objective.



The Software process framework is required for representing common process activities. Five framework activities are described in a process framework for software engineering.

1. Communication
2. planning
3. modeling
4. construction
5. and deployment.

These are all examples of framework activities. Each engineering action defined by a framework activity comprises a list of needed work outputs, project milestones, and software quality assurance (SQA) points.



1. Communication:
   Gathering requirements from customers and stakeholders to define the system's objectives.
   - Activities:
     - Requirement Gathering: Meetings, interviews, surveys.
     - Objective Setting: Defining system goals.
   - Explanation: Ensures all stakeholders are aligned on the system's goals.

2. Planning:
   Establishing a detailed work plan, identifying risks, resources, and setting a timeline.
   - Activities:
     - Work Plan, Risk Assessment, Resource Allocation, Schedule Definition.
   - Explanation: Organizes the project and prepares for potential challenges.

3. Modeling:
   Creating designs and models to define the system structure and functions.
   - Activities:
     - Requirement Analysis, System Design.
   - Explanation: Translates requirements into a visual blueprint for development.

4. Construction:
   Building and testing the software to ensure it meets requirements.
   - Activities:
     - Code Generation, Testing, Bug Fixing.
   - Explanation: The actual development and testing of the software.

5. Deployment:
   Delivering the product to users and gathering feedback for improvement.
   - Activities:
     - Product Release, Feedback Collection, Product Improvement.
   - Explanation: Ensures the product meets user expectations.

Q.2 EXPLAIN CMM ?

=>
The SEI capability maturity model is a process meta model developed by software engineering Institute(SEI).

It defines the process characteristic that should exist if an organisation want to establish a software process that is complete the period of the SEI capability.
Maturity model should always be adopted, it argues that software development:

- It must be taken seriously.
- It must be thoroughly.
- It must be controlled uniformly.
- It must be tracked accurately.
- It must be conducted professionally.
- It must focus on the needs of its customer.

The SEI capability maturity module represents two type of meta models:
as a continuous model
as a staged model.

As a Continuous model
It describes the process in two dimensions as shown in the figure. Each process area are assessed against specific goal and practises and is rated according to the following capability levels:

Level 0: Incomplete
the process area is either not performed or does not achieve all goals and objectives defined by SEI capability. Maturity model for level 1 capability.

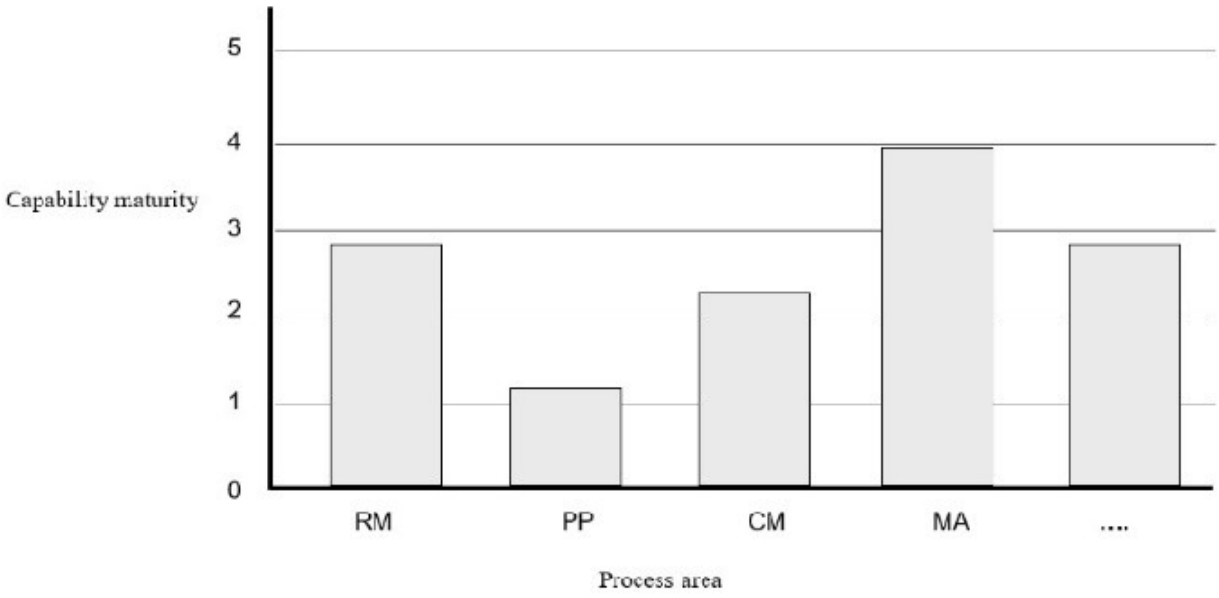
Level 1 :performed
all of the specific goals of the process area have been satisfied. Work task required to produce defined work, product and being conducted.

Level 2: managed
all level one criteria have been satisfied in Edison.
All work associated with the process area confirms to an organisationally defined policy.
All people doing the work have access to adequate resources to get job done.

Level 3: Defined
all level two criteria has been achieved.
All the process is tailored from organisation set of standard process according to organisation guideline.

Level 4 :Quantitatively managed
all level three criteria have been achieved.
Also, the process is control and improve using measurement and quantitative assessment.

Level 5: optimised
all level four criteria has been achieved.
All the process area is adopted all standards to meet the changing customer needs.
The SEI CMM defines each more process area in terms of specific goal and specific practises.



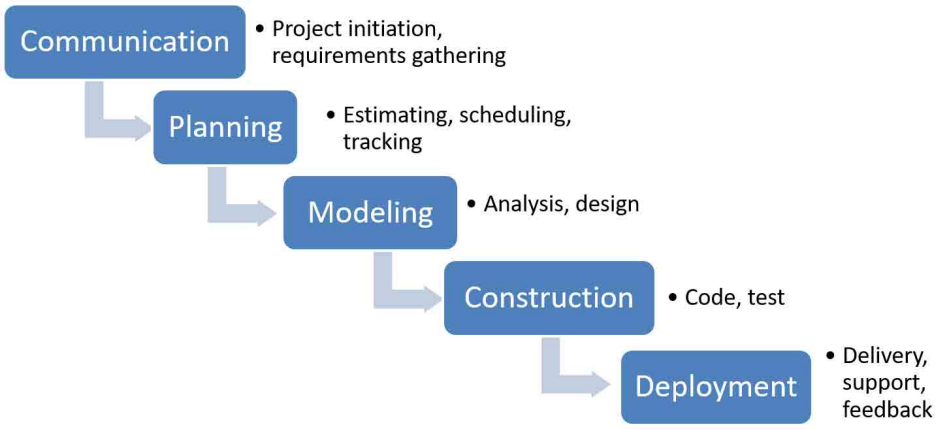
SEI CMM Process area Capability profile

Q.3 EXPLAIN WATERFALL MODEL?

=>
This model is also known as “Linear sequential Model” or “classic life cycle model”.

The **Waterfall Model** is a traditional software development methodology where the process is structured into sequential phases, each of which must be completed before moving on to the next.

It's called "Waterfall" because progress flows in one direction, similar to a waterfall. It’s ideal for projects with clear and fixed requirements.



1. Communication

Definition: This stage focuses on understanding the client’s requirements and defining the project's goals.
Goal: To ensure all stakeholders are aligned on what the software should do.
Activities:

- **Requirement Gathering:** Engaging with stakeholders to gather detailed system requirements.
- **Project Initialization:** Formalizing the project’s objectives and scope, ensuring a clear understanding of the client’s expectations.

2. Planning

Definition: In this stage, the project plan is created, outlining the timeline, resources, and risks involved in the project.
Goal: To ensure the project proceeds in an organized manner with clear expectations.
Activities:

- **Estimation:** Estimating the time, resources, and costs required for the project.
- **Scheduling:** Creating a project timeline with deadlines for each phase.
- **Tracking:** Establishing methods for tracking progress, ensuring that the project stays on course.

3. Modeling

Definition: This phase translates the requirements into a blueprint for the system, ensuring the team understands how to build the software.
Activities:

- **Analysis:** Breaking down the gathered requirements to define how the system should work.
- **Design:** Creating detailed architectural and component designs that serve as a guide for the development phase.

Goal: To create a structured design that guides the development team in coding the system.

4. Construction

Definition: The actual development of the software happens in this phase, followed by rigorous testing to ensure the system works as expected.
Activities:

- **Coding:** Developers write the software based on the design specifications.
- **Testing:** The system is tested for defects and to ensure that it meets the requirements outlined in the initial stages.

Goal: To build a bug-free, functional software product that meets all specified requirements.

5. Deployment

Definition: The software is delivered to the client, and feedback is collected for future improvements.
Activities:

- **Delivery:** The finished product is handed over to the client for use.
- **Support:** Ongoing maintenance and support are provided to address any issues that arise after deployment.
- **Feedback:** Users provide feedback on the software, which is considered for future updates or improvements.

Goal: To ensure the software is successfully delivered and satisfies the client's needs.

Q.4 SHORT NOTE ON AGILE METHODOLOGY.

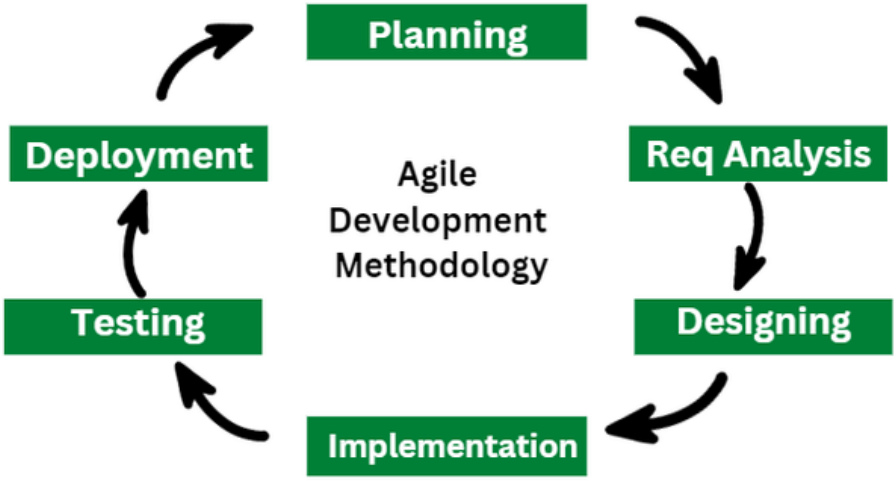
=>

• Agile is a project management and software development approach that aims to be more effective.

1)It focuses on delivering smaller pieces of work regularly instead of one big launch.
2)This allows teams to adapt to changes quickly and provide customer value faster.

• Agile methodologies are iterative and incremental, which means it's known for breaking a project into smaller parts and adjusting to changing requirements.

1)They prioritize flexibility, collaboration, and customer satisfaction.
2)Major companies like Facebook, Google, and Amazon use Agile because of its adaptability and customer-focused approach.



1. Requirement Gathering
In this stage, the project team identifies and documents the needs and expectations of various stakeholders, including clients, users, and subject matter experts.
It involves defining the project's scope, objectives, and requirements.
Establishing a budget and schedule.
Creating a project plan and allocating resources.

2. Design
Developing a high-level system architecture.
Creating detailed specifications, which include data structures, algorithms, and interfaces.
Planning for the software's user interface.

3. Development (Coding)
Writing the actual code for the software. Conducting unit testing to verify the functionality of

individual components.

4. Testing
This phase involves several types of testing:
Integration Testing: Ensuring that different components work together.
System Testing: Testing the entire system as a whole.
User Acceptance Testing: Confirming that the software meets user requirements.
Performance Testing: Assessing the system's speed, scalability, and stability.

5. Deployment
Deploying the software to a production environment.
Put the software into the real world where people can use it.
Make sure it works smoothly in the real world.
Providing training and support for end-users.

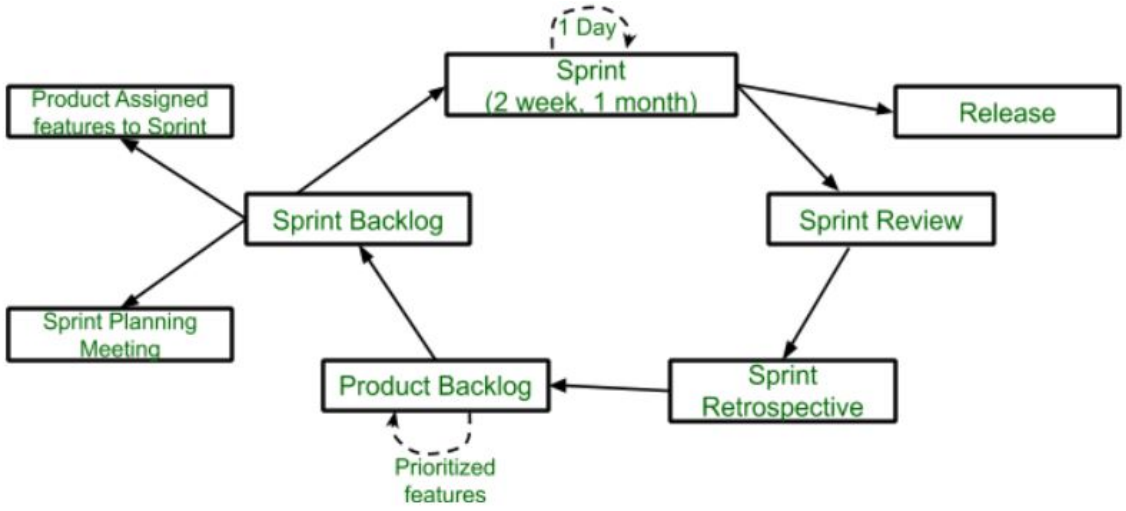
6. Review (Maintenance)
Addressing and resolving any issues that may arise after deployment.
Releasing updates and patches to enhance the software and address problems.

Q.5 SHORT NOTE ON SCRUM METHODOLOGY.

Scrum is a widely used Agile framework for managing complex software development projects.

It focuses on delivering products incrementally through iterative work cycles called sprints, typically lasting 2-4 weeks.

Scrum promotes flexibility, collaboration, and continuous feedback.



Key Elements of Scrum:

- Roles:
  - *Product Owner*: Manages the product backlog and prioritizes tasks based on customer needs.
  - *Scrum Master*: Facilitates the Scrum process, removes impediments, and ensures the team follows Scrum principles.
  - *Development Team*: A self-organizing, cross-functional group responsible for delivering the product increment.

- Artifacts:
  - *Product Backlog*: A prioritized list of features or tasks to be completed.
  - *Sprint Backlog*: A subset of the product backlog to be completed during a sprint.
  - Increment: The working product delivered at the end of each sprint.

- Events:
  - *Sprint Planning*: A meeting to define what will be done in the upcoming sprint.
  - *Daily Scrum*: A brief daily meeting to synchronize the team's progress.
  - *Sprint Review*: A meeting to showcase the work completed during the sprint.
  - *Sprint Retrospective*: A meeting to reflect on the sprint and identify improvements for the next.

Scrum encourages continuous feedback, adaptability to change, and regular delivery of functional software, making it ideal for dynamic and evolving projects.

Q.6 SHORT NOTE ON KANBAN METHODOLOGY.

=>

Kanban is an Agile methodology focused on visualizing and improving workflow efficiency in software development and other processes.

It emphasizes continuous delivery without the need for sprints or time-boxed iterations like Scrum.

Key Principles of Kanban:

1. **Visualizing Workflow:** A Kanban board is used to represent tasks as cards moving through various stages, such as "To Do," "In Progress," and "Done."
   This helps teams track progress and identify bottlenecks.

2. **Limiting Work in Progress (WIP):** Limits are set on the number of tasks in each stage to prevent overloading team members and ensure a steady flow of work.

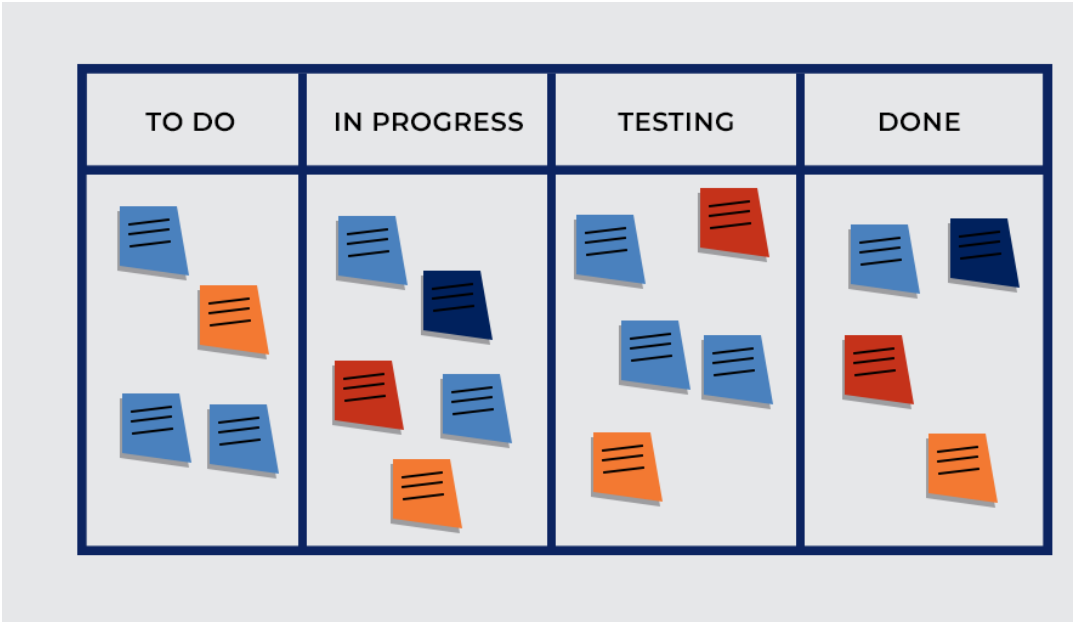
3. **Managing Flow:** Teams focus on optimizing the flow of tasks through the system, ensuring faster and smoother delivery.

4. **Continuous Improvement:** Teams regularly review and refine their processes for more efficiency.

Kanban is flexible, allowing tasks to be completed as they arise, making it ideal for teams that handle varying priorities and need to adapt quickly.

It supports continuous delivery and promotes workflow transparency.

Kanban is an Agile methodology focused on visualizing and improving workflow efficiency in software development and other processes. It emphasizes continuous delivery without the need for sprints or time-boxed iterations like Scrum.



Kanban Boards:

- A **visual tool** used to track work items across different stages, typically organized into columns like "To Do," "In Progress," and "Done."
- Helps teams monitor workflow, identify bottlenecks, and improve task management.

Kanban Cards:

- **Task representations** on the Kanban board, containing details like the task description, assignee, and deadline.
- Cards move through columns as the task progresses from one stage to another.

Benefits of Kanban:

1. **Improved Workflow Visibility:** Teams can easily track progress and identify delays or issues in real-time.
2. **Increased Efficiency:** Limiting Work in Progress (WIP) prevents task overload and ensures steady task completion.
3. **Flexibility:** Kanban allows for changes in priorities without needing to adjust iterations or sprints, making it highly adaptable.

Q.7 SHORT NOTE ON EXTREME PROGRAMMING(XP).

=>

The extreme programming is one of the most commonly used a child process models.

All the process models obey the principle of agility and manifesto of software development.

The XP uses the concept of object oriented programming. This approach is preferred development paradigm.

As in conventional approach, a developer focuses on the framework activities like planning, design, coding, and testing, the XP also has set of roots and practises.

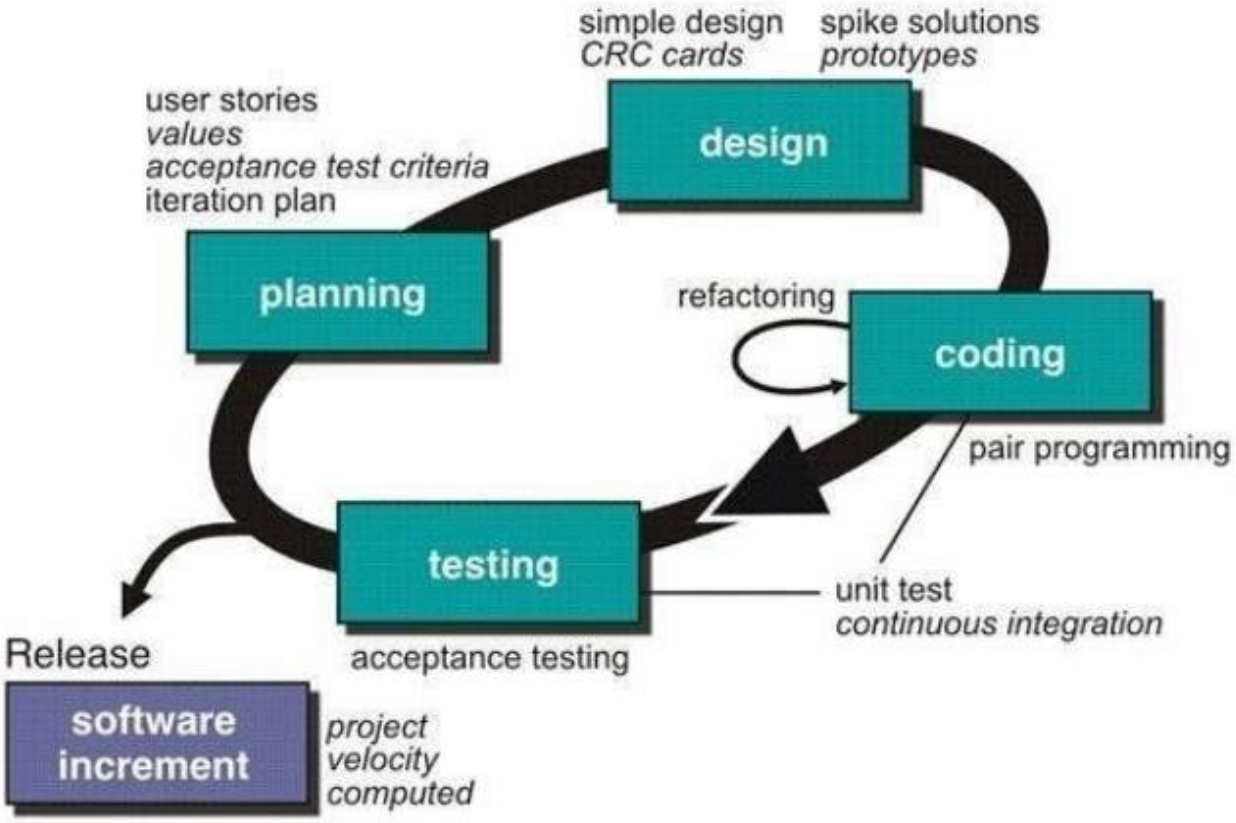


FIG7.1 EXTREME PROGRAMMING (XP) PROCESS

XP VALUES:

Following are set of five values at establish a foundation for all work performed in context with XP

1. Communication
2. Simplicity
3. Feedback
4. Courage
5. Respect

1) For any development process, there must be regular meeting of developer and the customer. There should be a proper communication for requirement, gathering, and discussion of the concepts.

2) The simple design can always be easily implemented in code.
3) The feedback is an important activity which leads to the discipline in the development process.

4) In every development project, there is always a pressure situation. The courage or the discipline will definitely make the task easy.

5) In addition to all the XP values, the agile should calculate respect among all the team members, between other stakeholders and with customer.

Q.8) SHORT NOTE ON SPIRAL MODEL.

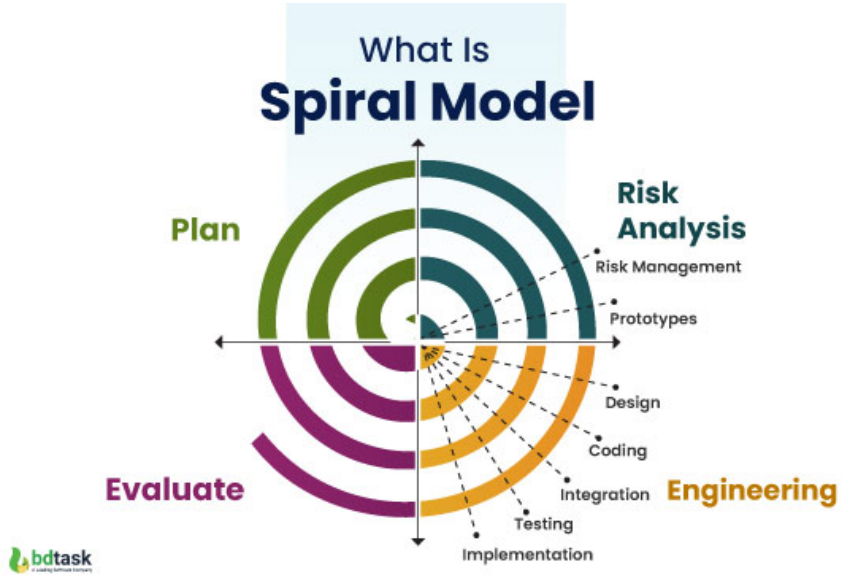
=>

The spiral model is combination of well-known waterfall model and iterative prototyping.
It yields rapid development of more complete version of software.

Using spiral model software is developed by series of evolutionary releases during the initial release.

It may be just paper work for Toto type, but during later release the version got more completed stages.

The spiral model is divided into a set of framework activities defined by software engineering team. Each framework activity represent one segment of spiral as shown in figure



1. COMMUNICATION

A software development process starts with communication between customer and developer

2. PLANNING

It includes complete estimation, example (Cost estimation of project) and scheduling (complete timeline chart for project development) and risk analysis

3. MODELLING

- It includes detail requirement, analysis and project design(algo, flowcharts etc).
- flowchart shows complete flow of program whereas algorithm is step-by-step solution of problem

4. CONSTRUCTION

It includes coding and testing steps:
1)Coding: design details are implemented using appropriate programming language
2)Testing: testing is carried out

5. DEPLOYMENT

It includes software delivery support and feedback from customer. If customer suggest some corrections or demand additional capabilities, then changes are required for such correction or enhancement.

Merits of Spiral Model:

1. **Risk Management:** Allows early identification and mitigation of risks in each iteration.
2. **Flexibility:** Adaptable to changes in requirements, providing room for project refinements.
3. **Customer Feedback:** Ensures continuous customer involvement and feedback, improving product quality.

Demerits of Spiral Model:

1. **Complexity:** Managing multiple iterations and risk assessments can be complex and resource-intensive.
2. **Costly:** The iterative nature and risk analysis make it expensive, especially for small projects.
3. **Requires Expertise:** It requires a highly skilled team to perform risk assessments and manage the process effectively.

MODULE 2: SOFTWARE REQUIREMENT ANALYSIS AND MODELING

Q.1) EXPLAIN REQUIREMENT MODEL

=>

The **Requirement Model** is a crucial phase in software development that focuses on identifying, documenting, and managing the needs and expectations of stakeholders for a system.

This model ensures that all stakeholders are aligned on the project's objectives and that the software meets the intended business goals.

It forms the foundation for the system design and subsequent phases of development.

Key Functions of the Requirement Model:

Inception:

- The initial phase where the project scope and objectives are defined. Stakeholders are identified, and high-level requirements are gathered to understand the project vision.

Elicitation:

- The process of gathering detailed requirements from stakeholders using techniques like interviews, surveys, and workshops. The goal is to capture user needs and expectations clearly.

Elaboration:

- Refining and expanding on the gathered requirements to create a detailed understanding of the system. Functional and non-functional requirements are fully developed during this phase.

Negotiation:

- Resolving conflicts between different stakeholder requirements, prioritizing needs, and ensuring that the project scope remains feasible given time and resource constraints.

Specification:

- Documenting the requirements in a clear and structured format, often in a **Software Requirements Specification (SRS)**. This document serves as a blueprint for developers and stakeholders.

Validation:

- Ensuring that the documented requirements accurately represent stakeholder needs and are feasible to implement. This may involve reviews, prototypes, and feedback sessions to confirm accuracy.

Requirement Management:

- Managing changes to requirements throughout the project lifecycle. It involves tracking changes, maintaining version control, and ensuring that the development aligns with updated requirements.

Q.2) EXPLAIN DATA FLOW DIAGRAM (upto 2 levels)

=>

DFD is also called as “Bubble Chart” as it consist of series of bubbles joined by lines.

A Data Flow Diagram (DFD) is a graphical representation used to visualize how data moves through a system, showing the flow of information between different processes, data stores, and external entities.

It’s commonly used during the system analysis phase to understand and model how data is processed within a system.

Key Components of a DFD:

1. Processes:
   - Represent tasks or functions where data is processed. Processes transform input data into output data.
   - Depicted as circles or rounded rectangles.

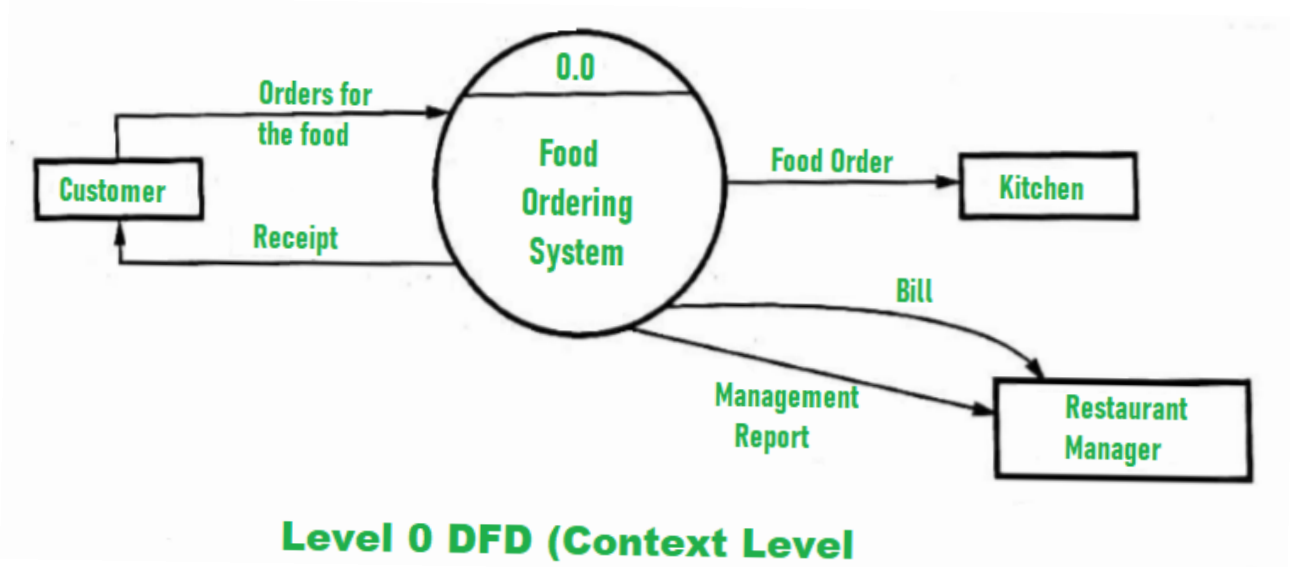
2. Data Flows:
   - Arrows that show the direction of data movement between components (processes, data stores, and external entities).
   - Indicate how data travels through the system.

3. Data Stores:
   - Represent where data is stored within the system, such as databases or files.
   - Depicted as open-ended rectangles.

4. External Entities (Sources or Sinks):
   - Represent people, organizations, or systems that interact with the system from outside.
   - Depicted as squares or rectangles.

Levels of DFD:

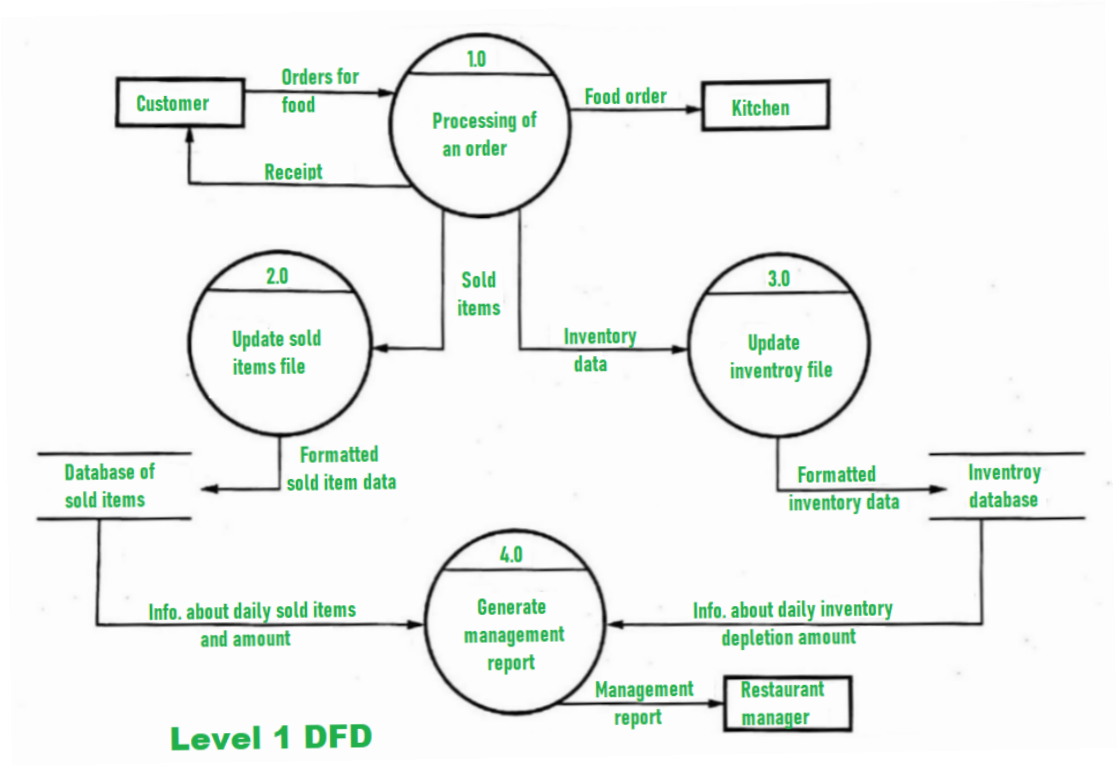
1. Context Level (Level 0):
   - A high-level overview showing the entire system as a single process, with its interactions with external entities.



Level 0 DFD (Context Level

2. Level 1 (Decomposition):

- Breaks down the single process from Level 0 into sub-processes, showing more detail about the internal data flows and processes.



3. Level 2 and beyond:

- Further decomposition of sub-processes into even more detailed DFDs, if needed.

Benefits of DFDs:

- Clarity: Provides a clear visual overview of how data moves through the system, making it easier to understand and analyze.

- Improves Communication: Helps developers, stakeholders, and business analysts communicate effectively about the system's data flow and processes.

- Identifies Gaps: Helps in identifying inefficiencies, missing processes, or bottlenecks in the system.

Example:

In a simple banking system, a DFD might show how a customer (external entity) sends a loan application (data flow) to the loan approval process (process), which accesses a customer database (data store) and returns a decision (data flow) back to the customer.

In summary, DFDs are essential tools for visualizing how data is processed and transferred through a system, aiding in system design and analysis.

Q.3) EXPLAIN SOFTWARE REQUIREMENT SPECIFICATION (SRS)

=>

Basically, SRS or software requirement specification is an official document or the statement of what the system developers implement.

It includes both system requirement and user requirement.

The SRS has a set of users like senior management of the organisation engineers responsible for developing the software.

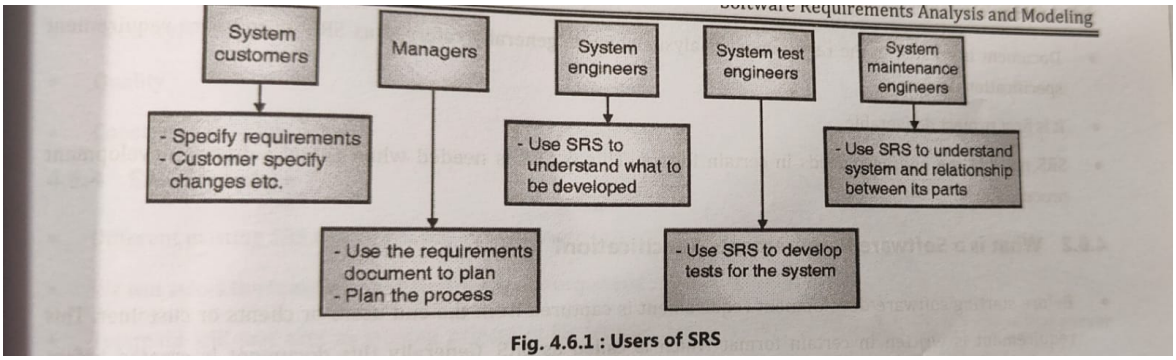


Fig. 4.6.1 : Users of SRS

Software Requirements Specification (SRS) is a comprehensive document that defines the expected behavior, functionalities, and constraints of a software system.

It serves as a formal contract between the stakeholders and the development team, ensuring that all parties are aligned on what the system will achieve.

- It signifies both developer and client understood what should be implemented in the software.
- All capabilities and functionalities are stored in SRS
- Requirement document need in every step of software development.
- Analyst uses this document for designing the software developer uses this document during the coding where software tester uses this document to check the functionalities of the software.
- All remaining project documents are depend upon requirement document because of what it is referred as parent of all document.
- It contains functional and non-functional requirements.

Good SRS have accomplishes following goals:

- It gives feedback to customer.
- The large problem can be divided into different smaller components.
- It acts as input to design which is part of design specification.
- It acts as product validation check.

Following things are part of SRS:

- Interfaces
- functional capabilities
- performance level
- data structures
- safety
- reliability
- security
- quality
- constraints and limitations

key characteristics of a good Software Requirements Specification (SRS):

1. **Complete:** Covers all functional and non-functional requirements needed for the system.
2. **Clear:** Uses simple and unambiguous language to avoid misunderstandings.
3. **Consistent:** SRS Should be consistent
4. **Testable:** Requirements can be tested to confirm they are met.
5. **Modifiable:** Organized in a way that allows easy updates and changes.
6. **Valid:** All requirement should be valid.

Q.4)DIFFERENCE BETWEEN FUNCTIONAL AND NON FUNCTIONAL REQUIREMENT

=>

Aspect	Functional Requirements	Non-Functional Requirements
Definition	Describes what the system should do, i.e., specific functionality or tasks.	Describes how the system should perform, i.e., system attributes or quality.
Purpose	Focuses on the behavior and features of the system.	Focuses on the performance, usability, and other quality attributes.
Scope	Defines the actions and operations of the system.	Defines constraints or conditions under which the system must operate.
Examples	User authentication, data input/output, transaction processing.	Scalability, security, response time, reliability, maintainability.
Measurement	Easy to measure in terms of outputs or results.	More difficult to measure, often assessed using benchmarks or SLAs.
Impact on Development	Drives the core design and functionality of the system.	Affects the architecture and overall performance of the system.
Focus on User Needs	Directly related to user and business requirements.	Focuses on user experience and system performance.
Documentation	Typically documented in use cases, functional specifications, etc.	Documented through performance criteria, technical specifications, etc.
Evaluation	Can be tested through functional testing (e.g., unit or integration tests).	Evaluated through performance testing, security testing, and usability testing.
Dependency	Determines what the system must do to meet user needs.	Depends on how well the system performs the required tasks.

MODULE 3: SOFTWARE ESTIMATION METRICS

Q.1) NOTE ON LOC

=>

Lines of Code (LOC) is a metric used in **software estimation** to measure the size of a software project by counting the number of lines in the source code.

It's one of the simplest and oldest ways to estimate the effort, time, and resources required to develop a software system.

Key Points:

1. **Definition:**
   LOC refers to the total number of lines written in a program, including executable instructions, declarations, and comments, though sometimes only the executable lines are counted.

2. **Purpose in Estimation:**
   LOC is used to estimate:

   - **Development effort:** The time and resources needed to write a certain number of lines.
   - **Cost:** More lines of code often indicate a larger, more complex system requiring more effort.
   - **Productivity:** Helps measure a developer's output by comparing LOC against time.

3. **Example:** If a project requires developing a simple payroll system, and similar past projects have taken approximately 50 lines of code per feature, an estimate might look like this:

   - **Number of Features:** 20
   - **Estimated LOC per Feature:** 50
   - **Total LOC Estimate:** 20 features × 50 LOC = 1,000 LOC

4. If historical data shows that 100 LOC typically takes 2 hours to develop, the estimated time for the payroll system would be:

   - **Development Time:** 1,000 LOC ÷ 100 LOC per 2 hours = 20 hours.

5. **Advantages:**

   - Simple and easy to measure.
   - Can be used with historical project data for effort estimation.

6. **Challenges:**

   - LOC doesn't measure complexity or code quality.
   - Developers can write more code than necessary, inflating the estimate.
   - It's language-dependent (different languages require different amounts of code to achieve the same functionality).

Conclusion:

LOC is a straightforward estimation metric but has limitations.

It is often combined with other metrics like Function Points or used with caution to ensure that complexity, efficiency, and quality are also considered.

An Example of LOC-Based Estimation (1)

Function	Estimated LOC
User interface and control facilities (UICF)	2,300
Two-dimensional geometric analysis (2DGA)	5,300
Three-dimensional geometric analysis (3DGA)	6,800
Database management (DBM)	3,350
Computer graphics display facilities (CGDF)	4,950
Peripheral control function (PCF)	2,100
Design analysis modules (DAM)	8,400
Estimated lines of code	*33,200*

18

Q.2) NOTE ON FUNCTIONAL POINT

=>

Function Points (FP) is a widely used metric in **software estimation** to measure the size and complexity of a software system based on its functionality rather than lines of code (LOC).

It focuses on what the software does from the user's perspective, making it a more reliable estimate of effort and resources required.

Key Points:

1. **Definition:**
   Function Points measure the functionality provided to the user, considering the inputs, outputs, data, and interactions in a system. It assesses the size of the system based on the complexity and number of functional components.

2. **Components of Function Points:**

   - **External Inputs (EI):** Data or control inputs received by the system (e.g., forms, buttons).
   - **External Outputs (EO):** Data sent out of the system (e.g., reports, error messages).
   - **External Inquiries (EQ):** User requests that trigger immediate responses without data modification (e.g., search results).
   - **Internal Logical Files (ILF):** Data maintained within the system (e.g., databases, files).
   - **External Interface Files (EIF):** Data used from external systems but not maintained by the system (e.g., external databases).

3. **Function Point Calculation:**
   Each component is assigned a weight (low, medium, or high) based on its complexity. The weighted total is summed to calculate the total **Function Points**.

4. **Example:** Consider an online bookstore system:

   - **External Inputs (EIs):** 10 user input forms (e.g., login, search book, add to cart).
   - **External Outputs (EOs):** 5 output reports (e.g., order confirmation, invoice).
   - **Internal Logical Files (ILFs):** 2 internal databases (e.g., user data, product catalog).
   - **External Interface Files (EIFs):** 1 external file (e.g., external shipping database).

5. Based on complexity:

   - Inputs: 10 (low complexity)
   - Outputs: 5 (medium complexity)
   - Internal Files: 2 (high complexity)
   - External Files: 1 (medium complexity)

6. By applying weights for each, a total Function Point count is determined. For example, this could total **40 FPs**.

7. **Advantages:**

   - Language-independent: Measures functionality, not code.
   - Better reflects system complexity and user needs.
   - Useful for early estimation when detailed design or coding hasn't started.

8. **Challenges:**

- Requires a good understanding of system functionality.
- More time-consuming to calculate compared to simpler metrics like LOC.
- May involve subjective judgments about complexity.

Conclusion:

Function Points provide a more meaningful and objective way to estimate effort in software development by focusing on user-required functionality rather than code size. They offer better insights into project scope and complexity, making them valuable for early estimation and resource planning.

Table 5.3.2 : Estimating information domain values

Information domain value	Opt.	Likely	Pess.	Est. count	Weight	FP count
Number of external inputs	20	24	30	24	4	97
Number of external outputs	12	15	22	16	5	78
Number of external inquiries	16	22	28	22	5	88
Number of internal logical files	4	4	5	4	10	42
Number of external interface files	2	2	3	2	7	15
Count total						320

Q.3)EXPLAIN COCOMO IN DETAIL

=>

The COCOMO (Constructive Cost Model) is a widely used software cost estimation model developed by Barry Boehm in the early 1980s.

It provides a structured framework for estimating the effort, time, and cost associated with software development projects based on various parameters, primarily the size of the software being developed.

COCOMO has evolved over the years, resulting in different versions, including COCOMO I and COCOMO II, which accommodate changes in software development practices.

Key Features of the COCOMO Model:

1. **Size Estimation**:
    - The COCOMO model primarily uses the estimated size of the software, usually measured in lines of code (LOC), as a key input for cost estimation. The model allows for different size measurement methods, including Function Points.
2. **Effort Estimation**:
   The basic effort estimation equation in COCOMO is:
   $\text{Effort(Person - Months)} = a \times (\text{KLOC})^b$
   Where:
   KLOC is the estimated size in thousands of lines of code.
   a and b are constants determined by the type of project (organic, semi-detached, or embedded).
3. **Project Types**:
    - COCOMO categorizes software projects into three types:
        - **Organic**: Small, simple projects developed by a small team with minimal complexity.
        - **Semi-Detached**: Medium-sized projects with a mix of experienced and inexperienced personnel, leading to moderate complexity.
        - **Embedded**: Complex projects requiring extensive hardware interaction, typically involving high risks and resources.
4. **Cost Drivers**:
    - COCOMO incorporates various cost drivers that can affect productivity, including:
        - Product complexity
        - Team capability
        - Development environment
        - Project schedule and constraints
    - These factors help refine the effort estimate based on specific project conditions.
5. **Scale Factors**:
    - The model includes scale factors that adjust the effort estimation based on project characteristics, such as product reliability and team cohesion. These factors allow for a more nuanced understanding of how different elements impact project costs.

COCOMO I vs. COCOMO II:

- **COCOMO I**: The original model that provides a basic framework for cost estimation. It focuses primarily on LOC and simple project types, making it less adaptable to modern software development practices.

- **COCOMO II**: An updated version that addresses the limitations of COCOMO I, incorporating modern development methodologies such as agile practices. COCOMO II offers more flexible size measurement options, supports iterative development, and provides a more detailed analysis of cost drivers and effort multipliers.

Benefits of the COCOMO Model:

- **Structured Approach**: COCOMO offers a systematic method for estimating software development costs, helping project managers make informed decisions.
- **Flexibility**: The model can be adapted to various project types and measurement techniques, enhancing its applicability.
- **Historical Data Usage**: COCOMO leverages historical data from similar projects to improve estimation accuracy.

Limitations:

- **Dependence on Historical Data**: Accurate estimates require relevant historical data, which may not always be available.
- **Assumptions and Simplifications**: The model relies on certain assumptions that may not hold true in every project scenario, potentially leading to inaccuracies.

Conclusion:

The COCOMO model is a foundational tool in software engineering for estimating project costs and resources. Its structured approach, coupled with its adaptability to different project types, makes it a valuable resource for software project managers. While COCOMO I laid the groundwork, COCOMO II has enhanced its relevance to modern software development practices, making it a key model for effective cost estimation in today's rapidly changing environment.

Q.5)PROJECT SCHEDULING AND TRACKING

=>

Project Scheduling and **Tracking** are essential components of project management that involve planning, executing, monitoring, and controlling the time and resources allocated to a project.

These processes ensure that a project is completed on time, within budget, and according to the specified requirements.

Project Scheduling

Definition: Project scheduling involves defining project activities, estimating the duration of each task, determining dependencies, and allocating resources to create a timeline for project completion.

Key Elements:

- **Task Breakdown**: Projects are divided into smaller, manageable tasks or activities. This breakdown is often represented in a Work Breakdown Structure (WBS).
- **Estimation**: Each task's duration is estimated using techniques such as expert judgment, historical data, or estimation algorithms (e.g., PERT, CPM).
- **Dependencies**: Identifying dependencies between tasks is very crucial in project scheduling. Tasks may be sequential, parallel, or dependent on the completion of other tasks.
- **Milestones**: Milestones are significant points in the project timeline that indicate critical progress or the completion of key deliverables.
- **Resource Allocation**: Scheduling involves assigning resources (human, financial, technical) to tasks to ensure they can be completed as planned.
- **Gantt Charts**: Visual tools like Gantt charts are commonly used to represent the project schedule, showing task durations, dependencies, and milestones.

Project Tracking

Definition: Project tracking is the process of monitoring project progress against the established schedule and budget.

It involves collecting data on task completion, performance metrics, and resource utilization to ensure that the project stays on course.

Key Elements:

- **Progress Measurement**: Tracking involves measuring the completion of tasks against the planned schedule.
  Techniques such as Earned Value Management (EVM) can help assess the value of work completed relative to the planned budget and schedule.

- **Reporting**: Regular reporting of project status to stakeholders is crucial. This includes updates on completed tasks, upcoming milestones, and any issues or risks encountered.
- **Change Management**: Projects often face changes in scope, resources, or timelines. Effective tracking allows project managers to identify deviations from the plan and implement change management processes to address them.
- **Risk Monitoring**: Tracking involves continuous monitoring of identified risks and emerging issues, enabling proactive responses to mitigate their impact on the project.
- **Adjustments**: Based on tracking results, project managers may need to adjust the project schedule or reallocate resources to keep the project on track.

Importance of Project Scheduling and Tracking

1. **Time Management**: Effective scheduling ensures that tasks are completed in a timely manner, preventing delays and ensuring that project deadlines are met.
2. **Resource Optimization**: Proper scheduling and tracking help optimize the use of resources, minimizing waste and ensuring that the right resources are allocated to the right tasks.
3. **Improved Communication**: Clear schedules and regular tracking updates facilitate communication among team members and stakeholders, ensuring everyone is aligned on project status and expectations.
4. **Risk Mitigation**: By continuously monitoring progress and potential issues, project managers can identify risks early and implement mitigation strategies, increasing the likelihood of project success.
5. **Performance Evaluation**: Tracking allows project managers to evaluate team performance, identify bottlenecks, and make data-driven decisions to enhance overall project efficiency.

Conclusion

Project scheduling and tracking are vital processes in project management that contribute to the successful completion of projects.

By effectively planning, monitoring, and adjusting project activities, managers can ensure that projects are delivered on time, within budget, and meet quality standards.

The use of tools and techniques for scheduling and tracking enhances project visibility, communication, and overall performance, ultimately leading to greater project success.

MODULE 4: SOFTWARE ESTIMATION METRICS

Q.1) SOFTWARE DESIGN PRINCIPLES

=>

Design means to draw or plan something to show the look, functions and working of it.

Software Design is also a process to plan or convert the software requirements into a step that are needed to be carried out to develop a software system.

Software design principles are guidelines and best practices that help developers create software that is efficient, maintainable, scalable, and easier to understand.

these principles serve as foundational concepts to guide the developer in building high-quality software.

Following are the key principles for software design:

Anticipation means it should let the user know it's next step or next move.

Communication let the user know the status of any activity initiated by him through some text message or through some image.

Consistency must be maintained throughout the software design.
For example, navigation, controls, icons, or menus must be consistent with respect to its colour shape throughout a design.

Efficiency- The design of the interface must optimise the end users efficiency.

Flexibility The most part of the interface design. The user may explore the application in some random fashion.
The system must be flexible enough to accommodate the needs of the user.

Focus- The software interface should stay focused on the user task at hand.

Readability- Information presented should be readable to all the users.

Learn ability- The application must be designed to minimise learning time, main work integrity through the system.

Latency reduction should minimise the waiting time for user and use that waiting for some internal operation to complete.
(Note: Flow chart/ diagram banao in points ka bc)

Following these design principles helps software engineers build systems that are easy to understand, extend, and maintain.

They also improve collaboration, as these principles encourage clean code and modular structures, making it easier for multiple developers to work on the same project.

Q.2) EXPLAIN COHESION AND COUPLING WITH ITS TYPES

=>

Cohesion and **Coupling** are fundamental concepts in software engineering that contribute to creating well-organized and maintainable code.

They help define the relationships within and between modules (or components) in a software system, aiming to make the system modular, robust, and easy to modify.

Cohesion

- **Definition**: Cohesion refers to the degree to which the elements inside a module (like a class or function) are related to each other and work together to achieve a single, well-defined task.
- **High Cohesion**: High cohesion is desirable as it indicates that a module has a specific and focused responsibility. A highly cohesive module is easier to understand, test, and maintain because all its components contribute to a single purpose.
    - Example: In a bank management system, a "CustomerAccount" class that only manages information related to customer accounts (e.g., account balance, transactions) is cohesive. All methods and data within the class relate specifically to the concept of a customer account.
- **Low Cohesion**: Low cohesion suggests that a module handles unrelated tasks or responsibilities, which can lead to code that is confusing and harder to maintain.
    - Example: If the "CustomerAccount" class also included methods for handling loan processing or report generation, it would lack cohesion. This mix of unrelated functions can increase complexity and make the module difficult to modify or debug.

Types of Cohesion

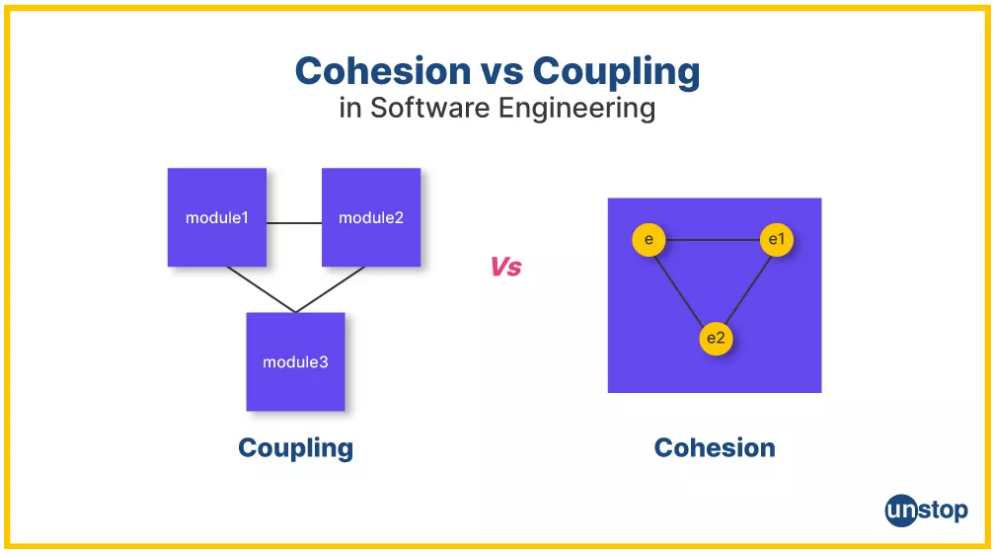
1. **Functional Cohesion**: All parts of the module contribute to a single, well-defined task (highest form of cohesion).
2. **Sequential Cohesion**: Elements are related by a sequence where output from one part serves as input to the next.
3. **Communicational Cohesion**: Components operate on the same data or contribute to similar tasks.
4. **Procedural Cohesion**: Components are grouped to follow a specific procedure, though not necessarily closely related.
5. **Temporal Cohesion**: Elements are related by timing, such as initializing tasks together.
6. **Logical Cohesion**: Elements perform similar tasks but are only loosely related.
7. **Coincidental Cohesion**: Elements are grouped arbitrarily without any meaningful relationship (lowest cohesion).

Coupling

- **Definition**: Coupling refers to the degree of interdependence between different modules. It measures how closely connected different modules or components are and how much they rely on each other to function.
- **Low Coupling**: Low (or loose) coupling is desirable, as it minimizes dependencies between modules. When modules are loosely coupled, changes in one module are less likely to impact others, making the system more modular and flexible.
    - Example: In the same bank management system, if "CustomerAccount" and "LoanProcessing" are separate classes and interact only through well-defined interfaces, they have low coupling. This makes it easier to modify or replace the "LoanProcessing" class without affecting the "CustomerAccount" class.
- **High Coupling**: High (or tight) coupling means modules are heavily dependent on each other, making the system more complex and difficult to modify.
    - Example: If "CustomerAccount" directly depends on specific methods or data from "LoanProcessing," any change in "LoanProcessing" might break "CustomerAccount," making maintenance difficult and introducing a higher risk of bugs.

Types of Coupling

1. **Data Coupling**: Modules interact by passing only necessary data.
2. **Stamp Coupling**: Modules share a data structure but use only a part of it.
3. **Control Coupling**: One module controls the behavior of another by passing control data.
4. **External Coupling**: Modules depend on external factors, such as shared global data, files.
5. **Common Coupling**: Multiple modules share global data, leading to high interdependence.
6. **Content Coupling**: One module directly modifies or relies on another module's internal workings (highest coupling).



High cohesion and low coupling together contribute to a modular, maintainable software design.

MODULE 5: SOFTWARE TESING

Q.1) SOFTWARE TESTING PROCESS

The **Software Testing Process** is a series of structured activities that ensure a software product meets specified requirements, functions correctly, and is free of defects.

This process is essential to maintaining software quality, as it identifies bugs, inconsistencies, and usability issues before the product reaches end users.

Key Stages in the Software Testing Process

1. **Requirement Analysis**
    - Testers review software requirements to understand what needs to be tested and ensure clarity and testability.
    - Test objectives, such as functionality, performance, and security, are identified in this stage.
2. **Test Planning**
    - A comprehensive test plan is created, outlining the scope, approach, resources, and schedule for testing activities.
    - This stage defines testing objectives, test criteria, and testing tools, and allocates roles and responsibilities.
3. **Test Case Design and Development**
    - Detailed test cases are designed based on the requirements to cover all aspects of the system.
    - Test cases include specific conditions, inputs, and expected outcomes to validate the system's functionality, usability, and reliability.
4. **Test Environment Setup**
    - The testing environment is configured to simulate the production environment, ensuring realistic testing.
    - Includes setting up hardware, software, network configurations, and required test data.
5. **Test Execution**
    - Testers execute test cases and record results, noting any deviations from expected outcomes.
    - Defects are reported and tracked, and the development team addresses these issues.
6. **Defect Tracking and Management**
    - Defects identified during test execution are documented, prioritized, and communicated to developers.
    - Retesting and regression testing are performed to verify that fixes work and new issues have not been introduced.

7. **Test Closure**
    - Once testing objectives are met, and major defects are resolved, test closure activities begin.
    - Test summary reports are created, evaluating the testing process, identifying areas for improvement, and documenting lessons learned.
    -

Types of Testing in the Process

- **Unit Testing**: Validates individual components or modules.
- **Integration Testing**: Checks interactions between modules.
- **System Testing**: Ensures the entire system meets requirements.
- **Acceptance Testing**: Confirms readiness for deployment by validating against end-user requirements.

Importance of the Software Testing Process

- **Quality Assurance**: Ensures software meets specified standards and user expectations.
- **Reliability and Performance**: Validates that the software performs consistently under expected conditions.
- **User Satisfaction**: Improves usability and overall experience, helping to build trust with users.
- **Cost Efficiency**: Identifies issues early, reducing the cost and effort of fixing defects in later stages.

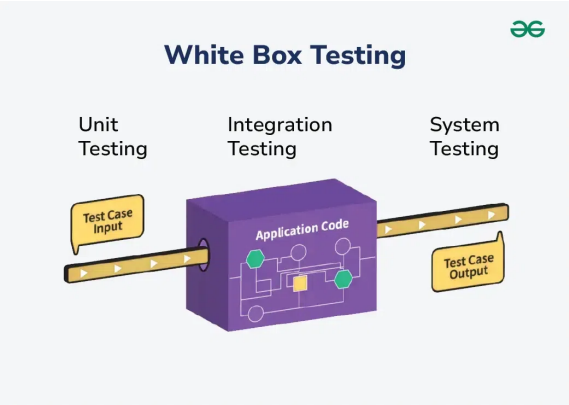
The software testing process is critical to delivering a high-quality, reliable product that meets user requirements and performs as expected.

Q.2) WHITE BOX TESTING AND BLACK BOX TESTING

White Box Testing and **Black Box Testing** are two primary software testing techniques used to identify defects and ensure software quality, each focusing on different aspects of the software.

White Box Testing

- **Definition**: Also known as structural, transparent, or glass-box testing, White Box Testing involves examining the internal structure, code, and logic of the software. Testers have access to the code and use this knowledge to design test cases that cover possible execution paths and uncover defects.
- **Approach**: Testers look at the code to validate flow, logic, data handling, and error handling. Common techniques include **statement coverage** (testing each line of code at least once), **branch coverage** (testing each possible path), and **path coverage** (ensuring all potential execution paths are tested).
- **Focus**: This method focuses on internal aspects like code quality, security, and performance, and ensures that all parts of the code execute as intended.
- **Example**: A tester writes test cases to check if all conditions in an "if-else" statement execute correctly.
- **Advantages**: Identifies hidden errors within code structure, optimizes performance, and improves code quality by allowing thorough code analysis.
- **Limitations**: Requires programming knowledge, which can make it time-consuming and complex, and it may overlook issues related to user experience or functionality.



White Box Testing Methods

1. **Basic Path Testing**: Ensures each path in the program executes at least once, covering possible code paths.
2. **Control Structure Testing**: Tests different control structures (like if-else, loops) to confirm all branches and conditions work as expected.

3. **Statement Coverage**: Checks that every line of code is executed at least once to detect unused code or missing statements.

4. **Branch Coverage**: Validates that all possible branches (true/false in conditions) in code are tested for correct behavior.

Black Box Testing

- **Definition**: Black Box Testing, or behavioral testing, evaluates the software's functionality without knowing the internal code structure.
  It focuses on testing software from the user's perspective, verifying that the application behaves as expected based on requirements.

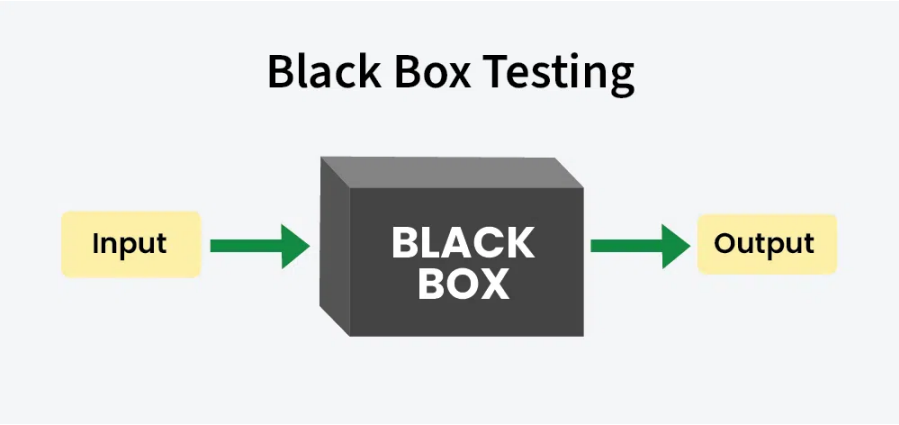
- **Approach**: Testers create test cases based on input-output conditions, expected behaviors, and functional requirements, without needing access to the code.
  Common techniques include **equivalence partitioning** (dividing input data into valid and invalid partitions), **boundary value analysis** (testing edge cases), and **decision table testing** (testing combinations of inputs).

- **Focus**: This method primarily checks functionality, usability, and how well the software meets specified requirements.

- **Example**: A tester enters various inputs into a login form to check if the login function works as expected.

- **Advantages**: Helps ensure that the software meets user expectations, is easier to perform without coding knowledge, and effectively catches functional issues.

- **Limitations**: Cannot uncover code-level errors or optimization issues and may not cover all paths, especially edge cases not identified in requirements.



Black Box Testing Methods

1. **Graph-Based Testing Method**: Uses graphs to model different states and transitions in the application to test workflows and interactions.

2. **Equivalence Partitioning**: Divides input data into valid and invalid partitions to reduce the number of test cases by testing one example from each partition.

3. **Boundary Value Analysis**: Focuses on testing the boundary values of input ranges, such as maximum, minimum, and just outside these limits, to catch edge case errors.

4. **Orthogonal Array Testing**: Uses systematic combinations of input data values to optimize test cases, ideal for applications with multiple inputs and settings.

Q.3) EXPLAIN BOUNDARY VALUE ANALYSIS IN DETAIL

Boundary Value Analysis (BVA) is a black box testing technique that focuses on testing the edges, or boundaries, of input ranges rather than the "normal" or "middle" values.

Since errors often occur at the boundary values of input, BVA helps to identify potential issues by examining conditions just inside, at, and just outside these limits.

Key Concepts in Boundary Value Analysis

1. **Boundary Conditions**: Boundaries refer to the minimum and maximum limits of input ranges. For example, if an application accepts ages between 18 and 60, then the boundaries are 18 (lower limit) and 60 (upper limit).

2. **Boundary Values**: These are the exact values at the boundaries and values close to them:

    - **Lower Boundary**: Minimum limit of the input range (e.g., 18).
    - **Upper Boundary**: Maximum limit of the input range (e.g., 60).
    - **Just Below the Lower Boundary**: Value slightly below the minimum (e.g., 17).
    - **Just Above the Lower Boundary**: Value just above the minimum (e.g., 19).
    - **Just Below the Upper Boundary**: Value just below the maximum (e.g., 59).
    - **Just Above the Upper Boundary**: Value slightly above the maximum (e.g., 61).

3. **Why Boundary Values are Important**: Software often encounters issues at boundaries due to potential errors in how the code handles edge cases. BVA increases the likelihood of identifying such defects, especially where boundary-related logic or comparisons might fail.

Boundary Value Analysis Example

Suppose we are testing a field that accepts numbers between 1 and 100:

- **Boundary Values**: 1 and 100 (exact limits).
- **Just Below the Lower Boundary**: 0.
- **Just Above the Lower Boundary**: 2.
- **Just Below the Upper Boundary**: 99.
- **Just Above the Upper Boundary**: 101.

For BVA, testers would create test cases for the values 0, 1, 2, 99, 100, and 101. Testing these values helps to ensure that the system correctly enforces the allowed input range.

Types of Boundary Value Analysis

1. **Single Boundary**: In cases where only a single input field or condition has boundary limits.
2. **Multiple Boundary**: When multiple fields have boundaries, BVA can combine boundary tests for each field (e.g., two fields, one with values 1-100 and another with values 200-300).

Advantages of Boundary Value Analysis

- **Efficiency**: Reduces the number of test cases needed by focusing only on boundaries, making it an efficient technique for uncovering defects.
- **Effective for Error Detection**: Commonly reveals boundary-related errors that might not be identified through random or middle-range input testing.

Limitations of Boundary Value Analysis

- **Limited to Numeric and Sequential Data**: Works best for ranges or ordered inputs; it is less effective for testing non-sequential data (e.g., categories, labels).
- **Does Not Cover All Scenarios**: Focusing only on boundaries may miss issues that occur within the input range.

Q.4) EXPLAIN ALPHA AND BETA TESTING WITH EXAMPLE

Alpha Testing and **Beta Testing** are two essential stages in the software testing process that focus on identifying defects before the software is released to the public.

Both types of testing have distinct purposes, environments, and participants.

Alpha Testing

- **Definition**: Alpha testing is an early testing phase conducted within the organization by the development team or a dedicated testing team. It aims to identify bugs and issues before the software is released to external users.

- **Environment**: Alpha testing is performed in a controlled environment, typically in the development or testing lab, where developers can quickly make fixes based on feedback.

- **Participants**: Primarily conducted by internal testers, including developers, quality assurance (QA) engineers, and selected end-users (usually from the organization).

- **Purpose**: The main goal is to catch as many bugs as possible, ensure that the software meets specified requirements, and evaluate its overall functionality and usability.

Example of Alpha Testing:
Imagine a software company developing a new accounting application. The development team completes the first version and then conducts alpha testing by having internal QA testers and a few employees from other departments use the application. They report bugs such as calculation errors, UI issues, and performance problems. The development team fixes these issues based on the feedback before moving on to the next phase.

Beta Testing

- **Definition**: Beta testing is the second phase of testing, where a version of the software is released to a limited number of external users (beta testers) outside the organization. This phase aims to gather feedback on the software's performance and usability in real-world conditions.

- **Environment**: Beta testing occurs in the user's environment, allowing the software to be tested under real-world conditions and usage scenarios.

- **Participants**: Conducted by external users who are not directly associated with the development team, such as customers, volunteers, or select clients.

- **Purpose**: The main goal is to identify any remaining issues that were not detected during alpha testing and to gather user feedback to improve the software before the final release.

Example of Beta Testing:
Continuing with the accounting application example, after addressing the issues found during alpha testing, the company releases a beta version to a group of selected customers. These customers use the application in their daily operations and provide feedback regarding usability, performance, and any additional bugs they encounter. This feedback helps the development team make final adjustments and improvements before the official launch.

Q.5) EXPLAIN REVERSE ENGINEERING.

The Reverse Engineering is the discipline of software engineering where the knowledge and design information is extracted from the source or it is reproduced.

The Reverse Engineering is a process where the system is analysed at higher level of extraction. It is also called as going backward through all the development cycles.

Following are some important purpose of reverse engineering:

- Security auditing
- Enabling additional features
- Used as learning tools
- Developing compatible product cheaper than that are currently available in the market

Following are the three important parameters to be considered for a reverse engineering process.

Abstraction level

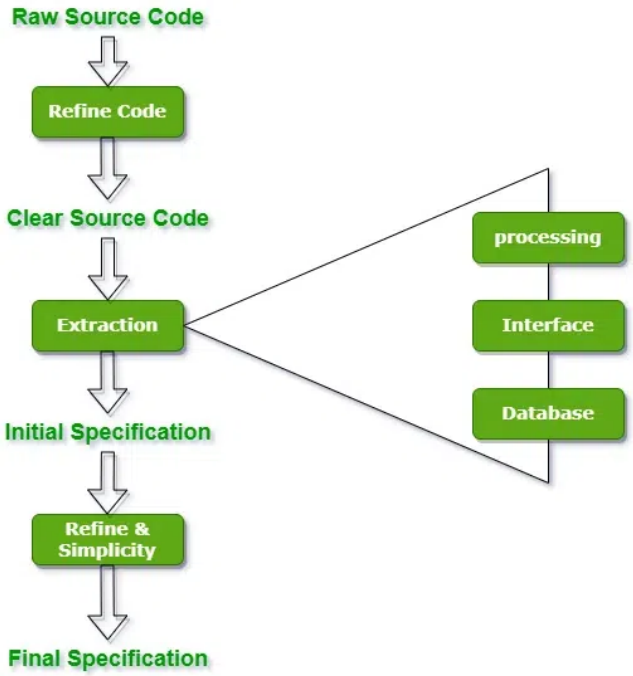
In the extraction level of reverse engineering process, the design information in a structure from the source code. It is the highest level in the engineering process.
It is always expected that exception level for energy level must be high.
When is higher it becomes easy for the developer to understand the program.

Completeness

Complete is nothing with the detail available through the abstraction level of reverse engineering process.
For example, from the given source, it is very easy to develop the complete procedural design.

Directionality

Directionality of reverse engineering process is a method of extracting information from the source port and making it label to software developer.
Software developers can use this information during the maintenance activity.
Following diagram exhibits the reverse engineering process.



The reverse engineering process include the following steps:

- The raw or dirty source code obtained from the abstraction level is taken as input.
- This code Is refined or restructured.
- From the clean code, the abstract is extracted.
- From this initial specification, the code is refined and simplified.
- Now we get the final specification.

Thus The final specification obtained from the reverse engineering process is used as final specification by developers.

Q.6) EXPLAIN CONTROL STRUCTURE TESTING .

Control structure testing is a key part of software testing focused on evaluating the various control structures within a program, including conditional branches, data flows, and loops. It ensures that all aspects of control flow work as intended, leading to more reliable and robust code. Let's dive into each type of control structure testing in detail:

1. Condition Testing

- **Purpose**: To ensure that all logical conditions in the program are evaluated correctly, covering all possible outcomes of conditional statements.
- **Focus**: Conditions are evaluated to check if they lead to the expected results. This type of testing is crucial for statements such as `if`, `else`, `switch`, and `case`.
- **Techniques in Condition Testing**:
    - **Simple Condition Coverage**: Tests each condition within a decision as true and false at least once.
    - **Multiple Condition Coverage**: Examines combinations of conditions within a compound decision.
    - **Condition/Decision Coverage**: Ensures that both individual conditions and overall decisions are covered.
    - **Modified Condition/Decision Coverage (MC/DC)**: Tests each condition within a decision independently to verify its effect on the outcome.
- **Example**: For the condition `if (A > 5 && B < 10)`, tests would cover scenarios like:
    - `A > 5` is true, `B < 10` is true.
    - `A > 5` is true, `B < 10` is false.
    - `A > 5` is false, `B < 10` is true.
    - `A > 5` is false, `B < 10` is false.

2. Data Flow Testing

- **Purpose**: To verify that data is used and updated correctly as it flows through the program. It helps in detecting issues like uninitialized variables, unused variables, and variables that are declared but not properly assigned.
- **Focus**: Data flow testing emphasizes the points where variables are defined, used, and modified.
- **Data Flow Terminology**:
    - **Def (D)**: When a variable is assigned a value (defined).
    - **Use (U)**: When a variable is read or used in a calculation (used).
    - **Kill (K)**: When the lifecycle of a variable ends, or it is redefined.
    -

- **Example**: Consider this pseudocode:

```
int x = 5; // Def
```

- `int y = x + 2; // Use of x`
- `x = y + 1; // Redefinition (Kill) of x`

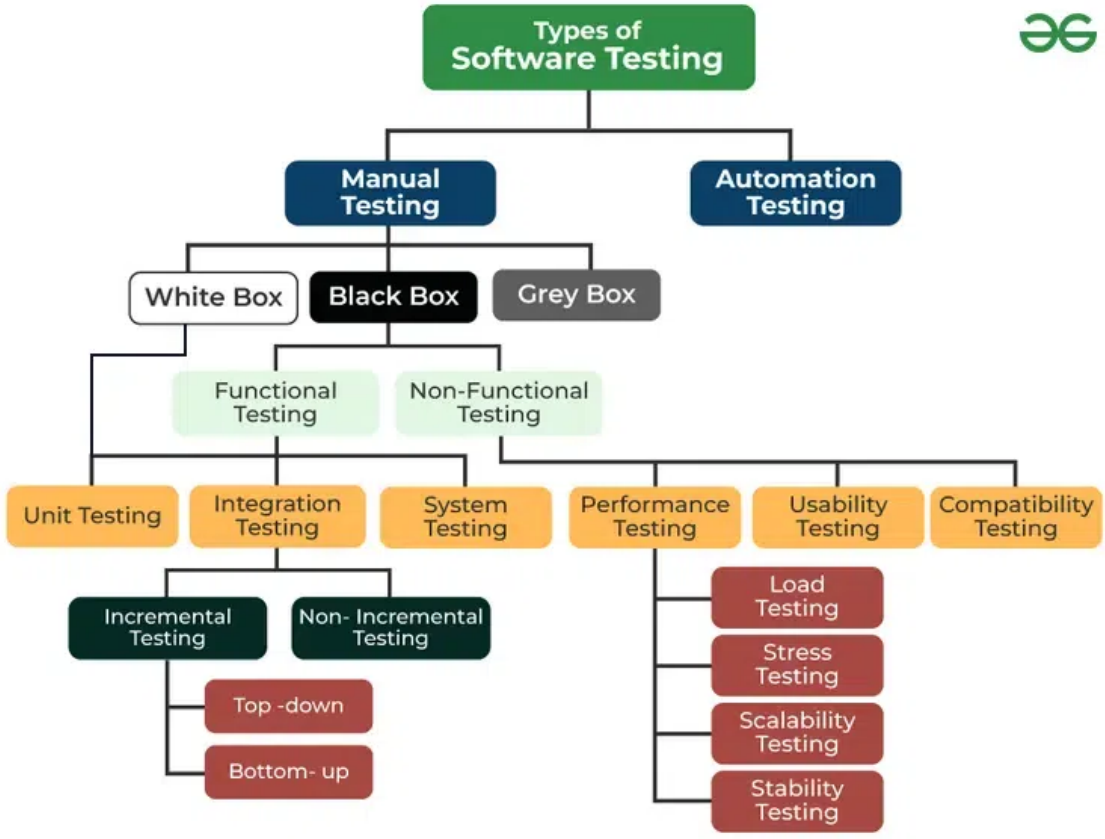
3. Loop Testing

- **Purpose**: To thoroughly test loops within the program, as loops can often lead to errors related to infinite looping, off-by-one errors, or incorrect initialization.
- **Focus**: Loop testing examines the behavior of loops under various conditions, including single iteration, multiple iterations, and boundary conditions.
- **Types of Loops**:
    - **Simple Loops**: Loops that have a single condition and a straightforward structure.
    - **Nested Loops**: Loops within loops, which are more complex and require more in-depth testing.
    - **Concatenated Loops**: Sequences of loops where the outcome of one loop affects the start of another.
- **Loop Testing Techniques**:
    - **Zero Iterations**: Tests how the loop behaves if it is skipped altogether (e.g., testing a `for` loop where the initial condition is not met).
    - **One Iteration**: Checks how the loop handles exactly one cycle, which can reveal errors in initialization or termination conditions.
    - **Multiple Iterations**: Tests loop functionality under typical conditions to ensure correct behavior over multiple cycles.
    - **Boundary Testing**: Tests loop behavior at its boundary conditions, such as maximum and minimum limits of the loop.
- **Example**: For a loop that runs based on `for (int i = 0; i < 10; i++)`, loop testing would involve:
    - Testing the loop with zero iterations (if `i` starts at a value greater than or equal to 10).
    - Testing with one iteration (if the loop condition changes slightly).
    - Testing with the typical range of iterations (1 to 9).
    - Testing the boundary by using `i = 9` to verify it stops correctly.

Q.7) WHAT IS SOFTWARE TESTING ?EXPLAIN TYPES OF SOFTWARE TESTING .

Software Testing is the process of evaluating and verifying that a software application meets the specified requirements and functions as expected.

It aims to identify bugs, ensure the software's quality, and enhance reliability before deployment.



Types of Software Testing:

- **Manual Testing**: Testing of software manually without the use of automation tools, requiring human intervention to execute test cases.
- **Automation Testing**: Testing that uses scripts and automation tools to perform tests without manual intervention, improving efficiency and speed.
- **White Box Testing**: Testing based on knowledge of the internal logic and code structure of the application, often performed by developers.
- **Black Box Testing**: Testing the functionality of the application without knowing the internal code structure, focusing on input-output and user interface behavior.
- **Grey Box Testing**: A combination of both white box and black box testing, where the tester has partial knowledge of the internal workings of the application.

- **Functional Testing**: Ensures that the software functions as expected according to the requirements.
- **Non-Functional Testing**: Focuses on non-functional aspects such as performance, usability, and reliability.
- **Unit Testing**: Tests individual components or units of code to ensure they work as intended.
- **Integration Testing**: Tests the interaction between different modules or components to ensure they work together.
- **System Testing**: Testing the complete and integrated software system to verify its compliance with specified requirements.
- **Performance Testing**: Tests how well the software performs under various conditions, including load and stress testing.
- **Usability Testing**: Focuses on evaluating how user-friendly and intuitive the software is for end users.
- **Compatibility Testing**: Ensures that the software works across different environments, platforms, and devices.
- **Incremental Testing**: Tests software incrementally, adding components progressively and testing their integration.
- **Non-Incremental Testing**: Tests the system all at once rather than in increments, typically done in one final phase.
- **Top-Down Testing**: Testing starts from the top of the system hierarchy and integrates modules progressively downwards.
- **Bottom-Up Testing**: Testing starts from the lower-level modules and integrates upwards until the whole system is tested.
- **Load Testing**: Tests the software's ability to handle a specified load of users or requests.
- **Stress Testing**: Tests the software under extreme conditions to see how it handles overloading or breaking points.
- **Scalability Testing**: Ensures the software can scale and handle increasing workloads.
- **Stability Testing**: Verifies that the software is stable under prolonged usage or stress.

Q.8) EXPLAIN RE ENGINEERING

Re-engineering is the process of modifying and updating an existing software system to improve its performance, functionality, and maintainability without altering its core functionality.

It typically involves analyzing the system's current state, making necessary changes, and improving the system's design to adapt to new requirements or technologies.

Key Activities in Re-engineering:

1. **Source Code Translation**: Converting the code from one programming language or platform to another to modernize the system.
2. **Restructuring**: Modifying the software's structure to improve readability, maintainability, or performance.
3. **Reverse Engineering**: Understanding the existing system's design and functionality by analyzing the codebase, often as a step toward modernization.
4. **Data Re-engineering**: Updating or converting data structures, databases, or file formats for improved data handling and efficiency.
5. **Refactoring**: Improving internal code quality without changing the software's external behavior, such as reducing complexity, removing redundancies, or optimizing performance.

Example:

A company may re-engineer its legacy inventory management system, originally built in COBOL, by converting it into a modern Java-based system to make it compatible with modern infrastructure and cloud services.

Benefits of Re-engineering:

- **Improved maintainability**: Makes the software easier to update and expand.
- **Cost-efficiency**: Reduces the need to build a new system from scratch.
- **Adaptation to new technology**: Allows the software to leverage modern hardware and software platforms.
- **Enhanced performance**: Optimizes performance by improving the system's architecture and design.

In short, re-engineering helps in evolving an aging system to meet current and future business needs while preserving its core functionality.

MODULE 6: SCM, QA & MANAGEMENT

Q.1)SQA & ITS TYPES

Software Quality Assurance (QA) is a systematic process that ensures the quality of software products throughout the development lifecycle. It involves various activities aimed at improving and ensuring the quality of software, including planning, monitoring, and controlling the software development process.

Key Objectives of Software Quality Assurance:

1. **Defect Prevention**: Identifying and addressing potential issues early in the development process.
2. **Process Improvement**: Continuously improving development processes to enhance software quality.
3. **Compliance**: Ensuring that the software meets specified requirements and standards.
4. **Customer Satisfaction**: Delivering a product that meets or exceeds user expectations.

Types of Software Quality Assurance:

Software QA can be categorized into several types based on the techniques and activities involved:

1. **Static QA**:
    - **Definition**: Involves reviewing and analyzing software artifacts without executing the code.
    - **Activities**:
        - Code reviews
        - Static code analysis
        - Inspections
        - Walkthroughs
    - **Benefits**: Early detection of defects, improved code quality, and reduced costs.

2. **Dynamic QA**:
    - **Definition**: Involves testing the software during its execution.
    - **Activities**:
        - Functional testing
        - Performance testing
        - Security testing
        - Usability testing
    - **Benefits**: Identifies defects that only occur during execution, ensuring the software behaves as expected under various conditions.

3. **Manual QA**:
    - **Definition**: Involves human testers executing test cases and assessing the software.
    - **Activities**:
        - Test case design
        - Test execution
        - Defect reporting
    - **Benefits**: Allows for exploratory testing and subjective evaluation of user experience.

4. **Automated QA**:
    - **Definition**: Involves using automated tools to execute tests and compare outcomes against expected results.
    - **Activities**:
        - Scripted test execution
        - Continuous integration testing
    - **Benefits**: Increased test coverage, faster feedback, and reduced manual effort.

5. **Quality Control (QC)**:
    - **Definition**: Focuses on identifying defects in the actual software products rather than processes.
    - **Activities**:
        - Testing (unit, integration, system, acceptance)
        - Defect tracking
    - **Benefits**: Ensures that the final product is free from defects and meets quality standards.

6. **Quality Assurance Metrics**:
    - **Definition**: Quantitative measurements used to assess the quality of the software and the QA process.
    - **Metrics**:
        - Defect density
        - Test coverage
        - Mean time to detect (MTTD)
        - Mean time to repair (MTTR)
    - **Benefits**: Helps in making data-driven decisions to improve quality.

Conclusion:

Software Quality Assurance is crucial for delivering high-quality software that meets user expectations and industry standards. By implementing various types of QA processes and techniques, organizations can enhance software reliability, performance, and maintainability, ultimately leading to greater customer satisfaction.

Q.2) NOTE ON FTR AND WALKTHROUGHS
=>

Formal Technical Review (FTR)

Definition: A Formal Technical Review (FTR) is a structured review process aimed at evaluating software products and associated documents to ensure quality, adherence to standards, and identification of defects.

FTRs are typically conducted early in the software development lifecycle, allowing teams to identify issues before they become costly to fix.

Key Characteristics:

- **Participants**: Involves a designated review team, including the author of the material being reviewed, peers, and sometimes stakeholders. The review team is usually selected based on expertise and relevance to the content being evaluated.
- **Preparation**: Reviewers prepare for the meeting by studying the material in advance, which includes design documents, source code, or test plans.
- **Documentation**: Formal documentation of the review process is maintained, including meeting minutes, issues identified, and action items.

Process:

1. **Planning**: Schedule the review meeting and select participants.
2. **Preparation**: Reviewers analyze the material prior to the meeting.
3. **Review Meeting**: The author presents the material, followed by a discussion where reviewers provide feedback and identify defects or issues.
4. **Follow-up**: The team documents findings and assigns action items to address identified issues.

Benefits:

- Early detection of defects reduces costs associated with fixing issues later in the development cycle.
- Enhances communication among team members and stakeholders.
- Promotes knowledge sharing and collective ownership of the product.

Walkthroughs

Definition: A walkthrough is a less formal review technique where a developer or author presents their work to a group for feedback.

The purpose of a walkthrough is to facilitate understanding of the content and gather input from peers.

Key Characteristics:

- **Participants**: Typically includes the author and a group of stakeholders, which may include developers, testers, and subject matter experts.
- **Less Structured**: Walkthroughs are generally more informal compared to FTRs, with less emphasis on strict procedures and documentation.

Process:

1. **Preparation**: The author prepares a presentation of the material, highlighting key aspects, decisions, and areas where feedback is needed.
2. **Presentation**: The author guides the group through the material, explaining the context and purpose.
3. **Discussion**: Participants ask questions and provide feedback, which may include suggestions for improvements or identifying potential issues.
4. **Documentation**: While not as formal as FTRs, important feedback may be recorded for future reference.

Benefits:

- Encourages collaboration and open communication among team members.
- Provides a platform for knowledge transfer and gaining diverse perspectives.
- Helps the author gain insights and suggestions that may enhance the work.

Conclusion

Both Formal Technical Reviews (FTR) and walkthroughs play crucial roles in the software development process by fostering collaboration, identifying defects, and ensuring that the final product meets quality standards.

While FTRs are structured and formal, walkthroughs are more informal and facilitate discussion and understanding among team members.

Utilizing these techniques can lead to improved software quality and greater team cohesion.

Q.3) NOTE ON VERSION AND CHANGE CONTROL
=>

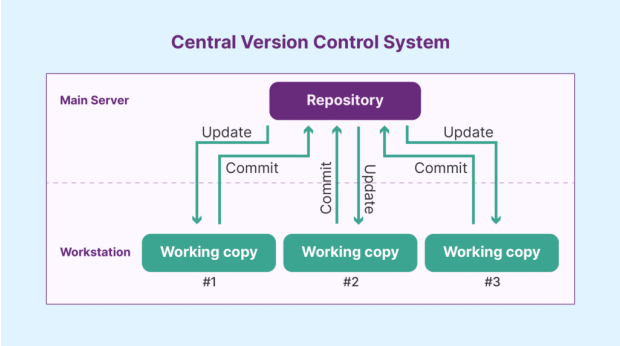
Version and Change Control

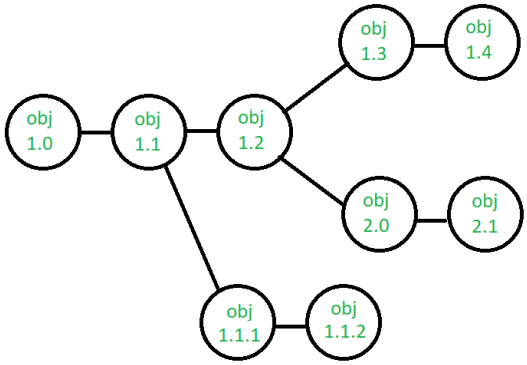
Definition: Version and change control is a systematic approach to managing changes to software and its associated documentation throughout the development lifecycle.

It ensures that all modifications are tracked, documented, and controlled, facilitating collaboration among team members and maintaining the integrity of the software product.

Key Concepts

1. **Version Control**:
    - **Definition**: Version control refers to the practice of tracking and managing changes to files, typically code, documentation, or other digital assets.
      It allows multiple versions of a document or software to coexist and provides a historical record of changes.





    - **Benefits**:
        - **Collaboration**: Enables multiple developers to work on the same project simultaneously without conflicts.
        - **History**: Maintains a complete history of changes, making it easy to revert to previous versions if necessary.
        - **Branching and Merging**: Supports the creation of branches for feature development or experimentation, which can later be merged into the main codebase.

2. **Change Control**:
    - **Definition**: Change control is a process used to ensure that all changes to a system or project are assessed, authorized, documented, and communicated.
      It aims to minimize disruptions and maintain the stability and quality of the software.

- **Change Request**: Any proposed change is documented in a change request form, which includes details such as the nature of the change, reason, impact assessment, and approval status.
- **Impact Analysis**: Evaluates the potential effects of the proposed change on the existing system, including risks, resource requirements, and timelines.
- **Approval Process**: Changes must be reviewed and approved by stakeholders or a change control board (CCB) before implementation.

Tools for Version and Change Control

1. **Version Control Systems (VCS)**:
   - Examples include Git, Subversion (SVN), Mercurial, and Perforce. These tools provide functionalities for tracking changes, branching, and merging code.
   - **Git**: A distributed version control system widely used for open-source and private software projects, enabling powerful branching and merging capabilities.
2. **Change Management Tools**:
   - Tools such as Jira, ServiceNow, and Trello facilitate change request documentation, tracking, and approval workflows.
   - **Issue Tracking**: Integrates version control with change management to provide visibility into how changes affect the codebase and project timelines.

Best Practices

1. **Establish a Clear Process**: Define and document the procedures for version control and change management, including roles and responsibilities.
2. **Consistent Naming Conventions**: Use clear and consistent naming conventions for version numbers (e.g., Semantic Versioning) to indicate the nature of changes (major, minor, patches).
3. **Regular Backups**: Ensure that the code repository and documentation are regularly backed up to prevent data loss.
4. **Communication**: Keep all team members informed about changes and their implications, fostering transparency and collaboration.
5. **Training**: Provide training to team members on version control and change management processes and tools to enhance their effectiveness.

Conclusion

Version and change control are essential components of software development that help maintain the quality and integrity of the product.

By systematically managing changes and maintaining version histories, teams can collaborate effectively, reduce errors, and ensure that the software meets its intended requirements.

Adopting best practices and utilizing appropriate tools can significantly enhance the overall development process.

Q.4) EXPLAIN SOFTWARE CONFIGURATION MANAGEMENT (SCM) & SCM PROCESS

=>

Software Configuration Management (SCM)

Definition: Software Configuration Management (SCM) is a systematic process that manages, tracks, and controls changes in software development projects. It ensures that the integrity and consistency of the software products are maintained throughout their lifecycle, including development, testing, deployment, and maintenance.

Objectives of SCM

1. **Version Control**: Track changes to software artifacts, such as source code, documentation, and configuration files, enabling the management of multiple versions of these artifacts.

2. **Change Control**: Manage changes in a structured manner, ensuring that all modifications are assessed, documented, and approved before implementation.

3. **Build Management**: Automate the process of compiling and assembling code into executable programs, ensuring that the build process is repeatable and consistent.

4. **Release Management**: Control the delivery of software products, ensuring that the correct versions of artifacts are released to users and stakeholders.

5. **Audit and Reporting**: Provide traceability of changes and versions for accountability and compliance purposes.

Key Components of SCM

1. **Configuration Identification**: Defining the items that need to be controlled and tracked, such as source code, libraries, documentation, and hardware.

2. **Configuration Control**: The process of reviewing and approving changes to configuration items, ensuring that changes are made systematically and that all stakeholders are informed.

3. **Configuration Status Accounting**: Keeping a record of the status of configuration items and their changes, providing a historical overview of the project's evolution.

4. **Configuration Audit**: Regularly reviewing configuration items and changes to ensure compliance with defined requirements and standards.

SCM Process

The Software Configuration Management process typically involves several key stages:

1. **Planning**:
    - Define the SCM policy and objectives, including scope, roles, and responsibilities.
    - Identify the tools and processes to be used for configuration management.
2. **Configuration Identification**:
    - Identify and document configuration items (CIs) that need to be managed. This includes software components, documentation, and related assets.
    - Establish version numbering schemes and naming conventions.
3. **Version Control**:
    - Implement a version control system (VCS) to track changes to CIs.
    - Use branching and merging strategies to manage concurrent development efforts and features.
4. **Change Control**:
    - Establish a change control board (CCB) or similar authority to review and approve change requests.
    - Implement a formal change request process, including impact analysis and documentation.
5. **Build Management**:

    - Automate the build process using build management tools (e.g., Jenkins, Maven, or Gradle) to compile and package software consistently.
    - Ensure that builds are reproducible and that artifacts are stored in a centralized repository.
6. **Release Management**:
    - Plan and execute software releases, ensuring that the correct versions of configuration items are delivered to users.
    - Document release notes and communicate changes to stakeholders.
7. **Status Accounting**:
    - Maintain records of configuration items, their versions, and change histories to facilitate tracking and reporting.
    - Provide regular status reports to stakeholders regarding the state of configuration items.
8. **Configuration Audit**:
    - Conduct regular audits to verify that configuration items comply with established standards and requirements.
    - Review the accuracy of configuration item records and ensure that all changes are properly documented.
9. **Continuous Improvement**:
    - Review the SCM process regularly to identify areas for improvement.
    - Gather feedback from team members and stakeholders to enhance the effectiveness of the SCM process.

Tools for SCM

Several tools are commonly used for Software Configuration Management, including:

- **Version Control Systems (VCS)**:
    - Examples: Git, Subversion (SVN), Mercurial, and Perforce.
    - Features include branching, merging, and tracking changes to code and documentation.
- **Build Automation Tools**:
    - Examples: Jenkins, Maven, Gradle, and Ant.
    - Automate the process of compiling, testing, and packaging software.
- **Change Management Tools**:
    - Examples: Jira, ServiceNow, and Trello.
    - Facilitate the documentation and approval of change requests.
- **Configuration Management Tools**:
    - Examples: Ansible, Puppet, and Chef.
    - Manage and automate the configuration of systems and environments.

Q.5) EXPLAIN SOFTWARE MAINTENANCE AND ITS TYPE

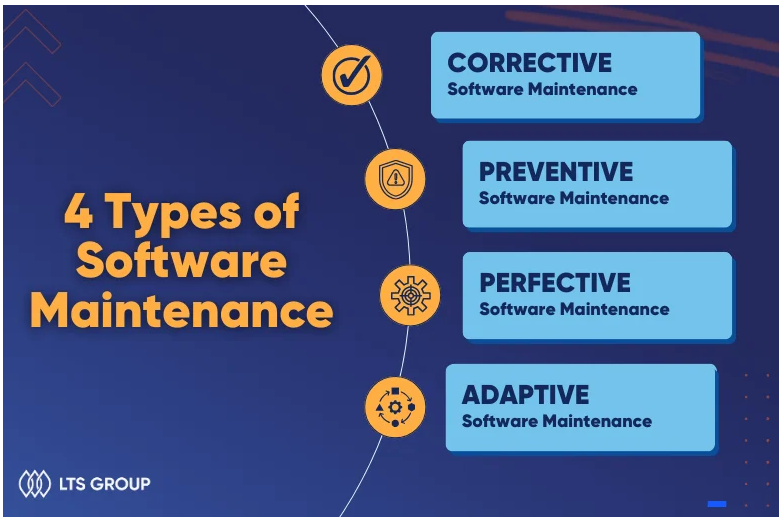
Software Maintenance

Definition: Software maintenance refers to the process of modifying and updating software applications after their initial deployment.

It encompasses all activities aimed at improving, fixing, or enhancing the software to ensure it continues to meet user needs and performs effectively over time.

Importance of Software Maintenance

- **Bug Fixes**: Addresses errors and defects that arise after the software is in use.
- **Performance Improvement**: Enhances the efficiency and speed of the software.
- **Adaptation to Changes**: Modifies software to keep it aligned with changing user requirements, operating environments, and technologies.
- **Extended Lifespan**: Helps prolong the useful life of the software through updates and improvements.



Types of Software Maintenance

1. **Corrective Maintenance**:
    - **Definition**: This type focuses on fixing defects or bugs in the software that are discovered after deployment.
    - **Characteristics**:
        - Reactive in nature, as it occurs in response to reported issues.
        - Aims to restore the software to its intended functionality.
    - **Example**: Patching a software application to resolve a critical security vulnerability.

2. **Adaptive Maintenance**:
    - **Definition**: This type involves modifying the software to accommodate changes in the environment in which it operates, such as updates to operating systems, hardware, or third-party software.
    - **Characteristics**:
        - Ensures continued compatibility and performance as external factors change.
        - Often involves reconfiguring or updating the software components.
    - **Example**: Updating an application to work with a new version of an operating system.
3. **Perfective Maintenance**:
    - **Definition**: This type focuses on enhancing and improving the software's functionality based on user feedback and evolving requirements.
    - **Characteristics**:
        - Proactive maintenance aimed at adding features or improving performance.
        - Involves refining existing features or introducing new ones to meet user expectations better.
    - **Example**: Adding new functionalities to a mobile app based on user requests.
4. **Preventive Maintenance**:
    - **Definition**: This type aims to prevent potential problems and defects before they occur through regular updates and checks.
    - **Characteristics**:
        - Focuses on reducing the risk of future issues by implementing improvements or updates.
        - Often includes regular code reviews, performance monitoring, and system checks.
    - **Example**: Conducting routine maintenance checks and optimizations to ensure system stability and performance.

Q.6)WHAT IS RISK? EXPLAIN TYPES OF RISK? EXPLAIN RMMM.

=>

What is Risk?

Risk in software development refers to the possibility that a project will not achieve its goals due to unforeseen events or conditions.

These events can lead to delays, cost overruns, or failure to meet the quality or performance requirements.

Risks are inherent in any software project due to factors like evolving requirements, technical challenges, and changes in business environments.

Risk is characterized by two key factors:

1. **Probability**: The likelihood that a certain event will occur.
2. **Impact**: The potential consequences or damage if the risk materializes.

Types of Risks

In software engineering, risks can be categorized into different types based on their nature:

1. Project Risks:

These are risks that affect the project plan and schedule. Project risks can lead to delays or increased costs.

- **Examples**:
  - Inaccurate time and cost estimates.
  - Lack of resources or skilled personnel.
  - Unforeseen technical challenges.
  - Changes in the project scope or goals.

2. Product Risks:

Product risks affect the quality and performance of the software product. These risks are usually associated with the technology being used or technical challenges that arise during development.

- **Examples**:
  - Unfamiliar or immature technology.
  - Integration issues with third-party systems or APIs.
  - Incompatibility with the hardware or software environment.
  - Failure to meet performance or scalability requirements.

3. Business Risks:

Business risks affect the viability or value of the project from a business perspective. These risks may result in the software not providing the intended benefits or failing to meet market needs.

- **Examples**:
  - Changing customer or market requirements.
  - Loss of a key stakeholder.
  - Inability to generate a return on investment (ROI).
  - Competitors releasing better products faster.

Risk Mitigation, Monitoring, and Management (RMMM)

The **Risk Mitigation, Monitoring, and Management (RMMM)** process helps software projects address risks proactively by taking actions to reduce, monitor, and manage potential risks before they cause serious problems.

1. Risk Mitigation:

Risk mitigation is the process of reducing the probability or impact of a risk.

This step involves identifying risks in advance and planning actions to either eliminate or reduce their likelihood or impact.

Common Risk Mitigation Strategies:

- **Avoidance**: Altering the project plan to avoid the risk entirely (e.g., choosing simpler, well-known technology to avoid technical risks).

- **Reduction**: Taking proactive steps to minimize the impact or likelihood of a risk (e.g., training team members to reduce the skill gap).

- **Transfer**: Transferring the risk to a third party (e.g., outsourcing part of the project to an external vendor).

- **Acceptance**: Acknowledging the risk and planning for contingencies if the risk occurs (e.g., accepting delays but planning backup timelines).

2. Risk Monitoring:

Risk monitoring involves continuously observing the project for potential risks and tracking the progress of mitigation strategies.

It ensures that the project team is aware of any new risks or changes in existing risks.

During monitoring, the project team regularly evaluates and updates the status of all identified risks, reviews mitigation plans, and adjusts the risk management strategy accordingly.

Key Steps in Risk Monitoring:

- Regular risk assessments throughout the project lifecycle.
- Ongoing review of mitigation strategies to see if they are effective.
- Tracking risk indicators, like delays or performance issues.
- Updating stakeholders about emerging risks.

3. Risk Management:

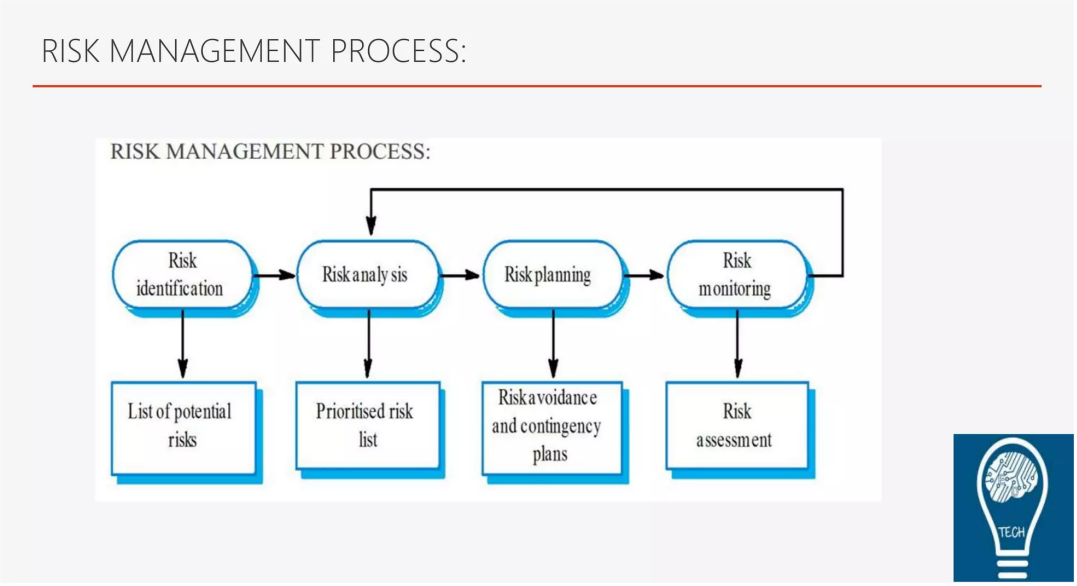
Risk management is the overall process of identifying, assessing, and controlling project risks.

It includes risk planning, assigning ownership, taking steps to mitigate risks, and ensuring that risk control measures are in place.

Effective risk management improves the likelihood of project success by systematically addressing potential threats.

The Risk Management Process:

1. **Risk Identification**: Identifying all possible risks that may impact the project.
2. **Risk Assessment**: Analyzing the likelihood and potential impact of each risk.
3. **Risk Prioritization**: Ranking risks based on their impact and likelihood to address the most critical ones first.
4. **Risk Mitigation Plan**: Developing actions to reduce or eliminate risks.
5. **Risk Monitoring**: Continuously tracking and reassessing risks.



Example of RMMM:

Imagine a project that involves integrating a payment gateway into an e-commerce platform.

A Product risk could be that the payment gateway provider's API may change during development.

The project team could mitigate this risk by keeping close communication with the API provider, monitoring their updates, and preparing for contingency plans, such as working with an alternative API.

By monitoring the risk and managing any changes, the project can avoid disruptions.

Importance of Risk Mitigation, Monitoring, and Management:

- **Improves Project Success Rate**: By identifying and addressing risks early, projects are more likely to succeed.
- **Reduces Uncertainty**: Proactively managing risks reduces unexpected issues and helps avoid last-minute crises.
- **Better Resource Management**: Effective risk management helps in allocating resources efficiently by focusing on areas where risks are high.
- **Ensures Stakeholder Confidence**: When risks are managed well, stakeholders have greater confidence in the project's ability to meet deadlines and objectives.

In summary, **Risk Mitigation, Monitoring, and Management** is a critical process in software development to identify, reduce, and manage risks throughout the project lifecycle, ensuring smoother delivery and reduced chances of failure.