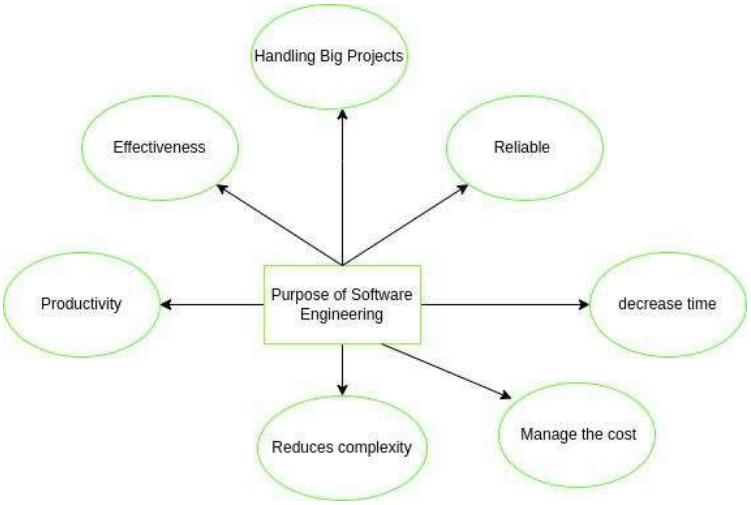
MODULE 1

What is Software Engineering? Why is it necessary

Software engineering is the process of designing, developing, testing, and maintaining software. It is needed because software is a complex and constantly evolving field that requires a structured approach to ensure that the end product is of high quality, reliable, and meets the needs of the users

Additionally, software engineering helps to manage the costs, risks and schedule of the software development process. It also provides a way to improve the software development process over time through testing and feedback.



What is Agile process Model. Different types of Agile

Agile is a software development and project management approach that emphasizes flexibility, collaboration, customer feedback, and continuous improvement.

Agile methodologies prioritize delivering small, incremental pieces of a project or product, which allows teams to adapt to changing requirements and customer needs.

There are several Agile methodologies, each with its own principles and practices. Here are some of the most popular Agile methodologies:

1. **Scrum:**

- Scrum is one of the most widely used Agile frameworks. It organizes work into fixed-length iterations called "sprints," typically lasting two to four weeks. During each sprint, a cross-functional team works on a prioritized set of features or user stories. Daily stand-up meetings (Scrum ceremonies) help the team stay aligned and address any impediments.

2. **Kanban:**

- Kanban is an Agile framework that visualizes work as a flow on a Kanban board. Teams use cards or tickets to represent work items, which move through columns representing different stages of the workflow. Kanban focuses on continuous delivery and minimizing work in progress (WIP) to improve efficiency.

3. **Extreme Programming (XP):**

- XP is an Agile methodology that emphasizes engineering practices such as pair programming, test-driven development (TDD), continuous integration, and frequent releases. It promotes close collaboration between developers and customers to deliver high-quality software.

4. **Lean Software Development:**

- Lean principles, adapted from manufacturing, emphasize eliminating waste, optimizing flow, and delivering value to the customer. Lean focuses on reducing lead time, improving efficiency, and enhancing overall product quality.

5. **Dynamic Systems Development Method (DSDM):**

- DSDM is an Agile methodology that provides a framework for rapid application development. It incorporates principles like timeboxing, prioritization, and frequent reviews to ensure that projects stay on track and deliver what the customer needs.

6. **Crystal:**

- Crystal methodologies, developed by Alistair Cockburn, come in various colors (e.g., Crystal Clear, Crystal Orange) and are tailored to different project sizes and complexities. They emphasize communication, simplicity, and team collaboration.

7. **Feature-Driven Development (FDD):**

- FDD is an Agile framework that focuses on feature modeling, domain object modeling, and regular progress reporting. It's particularly suitable for larger projects where breaking down features into manageable chunks is crucial.

9. **Adaptive Software Development (ASD):**

- ASD is an Agile methodology that encourages adaptation to changing circumstances. It emphasizes learning from experiences, continuous refinement, and collaboration between customers and developers.

Difference between Incremental Process models and Concurrent models.

Development Approach:	Incremental models emphasize breaking down the software development process into smaller, manageable parts or increments.	Concurrent models focus on executing multiple development activities concurrently. Instead of sequential phases, various phases of development, such as design, coding, testing, and integration, can happen simultaneously in different parts of the project.
Iterative vs. Parallel:	Incremental models are iterative in nature. Developers iterate through the same phases (requirements, design, coding, testing) for each increment until the entire system is complete.	Concurrent models allow for parallel execution of development activities. Multiple teams or individuals can work on different aspects of the project concurrently.
Risk Management:	Incremental development can help manage risks by allowing for early identification and mitigation of issues.	Concurrent development can be riskier, especially if there is insufficient coordination or communication between teams.
Flexibility:	Incremental models provide flexibility to adapt to changing requirements.	Concurrent models may be less flexible in accommodating changes once development activities have started in parallel.
Example:	Scrum and Kanban.	V-Model
Dependency Management:	Dependencies are typically managed within increments.	Managing dependencies between concurrent activities is a significant challenge.

What do you mean by Quality Function Deployment?

Quality Function Deployment (QFD) is a process and set of tools used to effectively define customer requirements and convert them into detailed engineering specifications and plans to produce the products that fulfill those requirements.

Goals

Understand Customer Needs.

Translate Customer Needs into Technical Requirements

Prioritize Requirements:

Key **components of the QFD** process include:

House of Quality (HOQ): The House of Quality is a central tool in QFD. It's a matrix that visually represents the relationships between customer requirements and the technical requirements needed to fulfill them. The HOQ helps teams prioritize and allocate resources to meet specific customer needs.

Customer Requirements (CRs): These are the needs and expectations of the customers, typically expressed in their own language and terms.

Technical Requirements (TRs): These are the specific features, characteristics, and attributes that the product or service must have to meet the customer requirements. TRs are often defined in engineering or technical terms.

Explain Functional and Non- Functional requirements with examples.

Functional Requirements:

Functional requirements describe what a system or software application is **supposed to do**. They define specific **functionalities, features, and actions** that the system must perform to meet user needs and business objectives

examples:

User Authentication: The system must allow users to create accounts, log in, and reset their passwords.

Email Notifications: The system must send an email confirmation to users after they make a purchase.

Non-Functional Requirements: Non-functional requirements, often referred to as **"quality attributes" or "system qualities,"** define the characteristics and constraints that apply to the system's operation and performance.

examples:

Performance: The system must load web pages within two seconds and handle 1,000 concurrent users without performance degradation.

Scalability: The system must be able to scale horizontally to accommodate increased user loads during peak traffic times.

Charcteristics of good srs

software requirements specification (SRS): A software requirements specification (SRS) is a document that describes what the software will do and how it will be expected to perform. It also describes the functionality the product needs to fulfill all stakeholders (business, users) needs.

1)Correctness: User review is used to ensure the correctness of requirements stated in the SRS. SRS is said to be correct if it covers all the requirements that are actually expected from the system.

2)Completeness: Completeness of SRS indicates every sense of completion including the numbering of all the pages, resolving the to be determined parts to as much extent as possible as well as covering all the functional and non-functional requirements properly.

3)Consistency: Requirements in SRS are said to be consistent if there are no conflicts between any set of requirements. Examples of conflict include differences in terminologies used at separate places,logical conflicts like time period of report generation, etc.

4)Unambiguousness: A SRS is said to be unambiguous if all the requirements stated have only 1 interpretation. Some of the ways to prevent unambiguousness include the use of modelling techniques like ER diagrams, proper reviews and buddy checks, etc.

5)Ranking for importance and stability: There should a criterion to classify the requirements as less or more important or more specifically as desirable or essential. An identifier mark can be used with every requirement to indicate its rank or stability.

6)Modifiability: SRS should be made as modifiable as possible and should be capable of easily accepting changes to the system to some extent.Modifications should be properly indexed and cross-referenced.

7)Verifiability: A SRS is verifiable if there exists a specific technique to quantifiably measure the extent to which every requirement is met by the system. For example, a requirement starting that the system must be user-friendly is not verifiable and listing such requirements should be avoided.

8)Traceability: One should be able to trace a requirement to design component and then to code segment in the program. Similarly, one should be able to trace a requirement to the corresponding test cases.

9)Design Independence: There should be an option to choose from multiple design alternatives for the final system. More specifically, the SRS should not include any implementation details.

10)Testability: A SRS should be written in such a way that it is easy to generate test cases and test plans from the document.

11)Understandable by the customer:An end user maybe an expert in his/her specific domain but might not be an expert in computer science. Hence, the use of formal notations and symbols should be avoided to as much extent as possible. The language should be kept easy and clear.

12)Right level of abstraction:If the SRS is written for the requirements phase, the details should be explained explicitly. Whereas, for a feasibility study, fewer details can be used. Hence, the level of abstraction varies according to the purpose of the SRS.

- **Correctness**
- **Completeness**
- **Consistency**
- **Modifibility**
- **Verifiability**
- **Tracebility**
- **Testability**
- **Design independence**
- **Understandable by the customer**
- **Unambiguousness**
- **Right level of abstraction**
- **Ranking for importance and stability**

What is requirement elicitation? Requirement engineering

Requirement Elicitation: Requirement elicitation is the initial phase of requirements engineering. It is the process of collecting, identifying, and gathering requirements from various stakeholders. The goal is to understand and document what the system or software should do, how it should behave, and what it should achieve in terms of functionality and performance.

Key activities in requirement elicitation include:

Stakeholder Communication: Engaging with stakeholders, such as end-users, customers, managers, and subject matter experts, to understand their needs and expectations.

Gathering Information: Collecting information through interviews, surveys, workshops, observations, and other techniques to uncover and document requirements.

Documentation: Capturing the elicited requirements in a structured format, such as requirement documents, user stories, use cases, or other modeling techniques.

Validation: Ensuring that the gathered requirements are complete, consistent, and accurate by involving stakeholders in the validation process.

Prioritization: Determining the relative importance of requirements and prioritizing them based on business goals, constraints, and stakeholder feedback.

Requirements Engineering: Requirements engineering is a broader process that encompasses all activities related to requirements throughout the software or system development lifecycle.

It encompasses seven distinct tasks:

inception, elicitation, elaboration, negotiation, specification, validation, and management. It is important to note that some of these tasks occur in parallel and all are adapted to the needs of the project.

What is Feasibility study?

A feasibility study in software engineering is a critical early phase of the software development life cycle (SDLC)

The primary goal of a feasibility study is to determine whether the project is worth pursuing and whether it can be completed successfully within the defined constraints.

includes the following key components:

1)Technical Feasibility: This aspect assesses whether the proposed project can be developed using existing technology and whether it can meet the required technical specifications. It involves evaluating the availability of the required hardware, software, and technical expertise. Technical feasibility also considers any potential risks or technical challenges that may arise during the project.

2)Economic Feasibility: Economic feasibility focuses on the financial aspects of the project. It involves estimating the project's costs and potential benefits. This includes not only the initial development costs but also ongoing maintenance, operational, and support costs. A cost-benefit analysis is often performed to compare the expected benefits against the projected expenses. If the expected benefits outweigh the costs, the project is considered economically feasible.

3)Operational Feasibility: Operational feasibility examines whether the proposed software solution can be integrated into the existing business processes and whether it aligns with the organization's goals and objectives. It also considers how the software will be maintained, supported, and operated after deployment.

4)Legal and Regulatory Feasibility: Legal and regulatory feasibility assesses whether the software project complies with relevant laws, regulations, and industry standards. It involves identifying any legal constraints, intellectual property issues, or data privacy and security requirements that must be addressed during development and operation.

5)Schedule Feasibility: Schedule feasibility evaluates whether the project can be completed within the desired timeframe. It considers the project's scope, complexity, available resources, and potential risks that could impact the schedule. A realistic project timeline is crucial to meeting stakeholder expectations.

6)Resource Feasibility: Resource feasibility looks at the availability of human resources, including skilled developers, project managers, and other team members, as well as physical resources like hardware and software licenses. Ensuring that the necessary resources are available is essential for successful project execution.

7)Market Feasibility (Optional): In some cases, especially for commercial software products, a market feasibility analysis may be conducted to assess whether there is a demand for the proposed software in the target market. This analysis may involve market research, competitor analysis, and customer surveys.

Based on the findings of the feasibility study, stakeholders can make an informed decision about whether to proceed with the project, modify its scope or objectives, or abandon it altogether.

It helps prevent investing time and resources in projects that are unlikely to succeed and ensures that the selected projects align with the organization's strategic goals.

What type of projects are best suited for Agile methodology and why?

Customer Collaboration: Projects that require close collaboration with customers or end-users benefit from Agile. Agile methodologies emphasize regular interactions with stakeholders to gather feedback, validate assumptions, and ensure that the delivered product aligns with customer expectations. This ongoing feedback loop is valuable for refining and improving the product.

Complex or Innovative Projects: Agile is well-suited for complex projects where solutions are not immediately apparent or require experimentation. It allows teams to break down complex problems into smaller, manageable increments (sprints in Scrum) and tackle them iteratively. Agile encourages experimentation and learning from both successes and failures.

Rapid Time-to-Market: Agile methodologies prioritize delivering valuable increments of a product quickly. This is ideal for projects where time-to-market is critical. By delivering usable features incrementally, Agile allows organizations to release a minimum viable product (MVP) sooner, which can be valuable for gaining a competitive edge or meeting market demands.

Cross-Functional Teams: Agile relies on cross-functional teams that include members with various skills and expertise. This is advantageous for projects that require collaboration between different departments or specialties. Cross-functional teams can work together to deliver end-to-end solutions efficiently.

What are projects best suited for water fall model, prototype model

=> Waterfall is suitable for projects where the requirements are stable and unlikely to change during development. Small to medium-sized projects with straightforward objectives and well-understood technology can benefit from this approach.

=> Prototype Model: The Prototype model, also known as the Iterative or Incremental model, is better suited for projects where the requirements are not well-defined or may change during development. It involves creating a preliminary version of the software to clarify requirements and gather feedback.

Elements or componenets of Use case diag, class diag, activity diag etc

Use Case Diagram:

A use case diagram in software engineering is a visual representation of the interactions between different actors (users or systems) and the various use cases (functional requirements) of a system. The key elements or components of a use case diagram include:

1. **Actor:** An actor is an external entity that interacts with the system. Actors can be individuals, other systems, or even hardware devices. In a use case diagram, **actors are represented as stick figures or icons.** Examples of actors might include "User," "Admin," or "Payment Gateway."

2. **Use Case**: A use case represents a specific functionality or feature that the system provides to its users. Use cases are depicted as ovals or ellipses and are connected to actors to show which actors are involved in or interact with each use case. Examples of use cases could be "Login," "Make a Reservation," or "Generate Invoice."

3**. Association (Line)**: Lines connecting actors to use cases represent associations and show that an actor interacts with a particular use case. Associations are typically labeled to indicate the nature of the interaction. For example, "User" <<uses>> "Make a Reservation" indicates that the "User" actor uses the "Make a Reservation" use case.

4. **System Boundary**: The system boundary is a box that encloses all the actors and use cases within the system. It defines the scope of the system under consideration.

Class Diagram:

A class diagram in software engineering is used to model the structure of a system by defining classes, their attributes, relationships between classes, and the operations or methods that can be performed on them. The key elements or components of a class diagram include:

1**. Class:** A class represents a blueprint for creating objects. It defines the properties (attributes) and behaviors (methods) that objects of the class will have. In a class diagram, classes are depicted as rectangles with three compartments: one for the class name, one for attributes, and one for methods.

2. **Attributes**: Attributes are the data members or variables that describe the state of a class. They are listed in the second compartment of the class rectangle and include the attribute name and data type. For example, "Car" class may have attributes like "make," "model," and "year."

3. **Methods**: Methods are the operations or functions that can be performed on objects of a class. They are listed in the third compartment of the class rectangle and include the method name, parameters, and return type. For example, "Car" class may have methods like "startEngine()" and "accelerate(speed)."

4**. Relationships**: Relationships between classes show how they are related or connected in the system. Common types of relationships in class diagrams include associations, aggregations, compositions, inheritance, and dependencies.

5. **Association:** An association represents a relationship between two or more classes. It can have multiplicity (indicating how many instances participate) and roles (naming the roles of each class in the relationship).

Activity Diagram:

An activity diagram in software engineering is used to model the flow of activities or processes within a system, showing the sequential and parallel flow of actions. The key elements or components of an activity diagram include:

1**. Activity**: An activity represents a specific action or task in the system. Activities are depicted as rounded rectangles and can include actions, decisions, forks, and joins.

2. **Action:** Actions are the basic units of work within an activity. They represent specific operations or computations that occur during the activity.

3. **Decision:** A decision node is used to represent a condition or a choice point in the process flow. It splits the flow into different paths based on a condition.

4. **Fork**: A fork node represents a point in the process where the flow splits into multiple parallel paths, allowing multiple activities to occur concurrently.

5. **Join:** A join node represents a point where multiple parallel flows converge into a single flow.

6. **Control Flow:** Control flow arrows or edges connect activities, decisions, forks, and joins to define the order and direction of the process flow.

7. **Start and End Nodes**: Every activity diagram begins with a start node and ends with an end node to indicate the start and termination of the process.

Explain briefly various steps involved in Requirement Engineering

Requirement Elicitation: This is the initial step where requirements are gathered from stakeholders. Techniques like interviews, surveys, workshops, and observations are used to understand their needs and expectations.

Requirement Analysis: Once gathered, requirements are analyzed to identify inconsistencies, conflicts, ambiguities, and missing information. The goal is to ensure that the requirements are clear, complete, and feasible.

Requirement Specification: The analyzed requirements are documented in a structured and organized manner. This includes creating documents like User Stories, Use Cases, Functional Requirements Specifications (FRS), and Non-Functional Requirements Specifications (NFRS).

Requirement Validation: The documented requirements are reviewed and validated by stakeholders, including end-users, domain experts, and developers. This step ensures that the requirements accurately reflect what the stakeholders want.

Requirement Management: Requirements change over time due to evolving needs or new insights. Requirement management involves tracking and controlling changes to ensure that the software stays aligned with the current requirements. This includes maintaining a traceability matrix to link requirements to design, implementation, and testing.

Requirement Prioritization: Not all requirements are equally important. Prioritization helps in determining which requirements should be addressed first. Techniques like MoSCoW (Must have , Should have, Could have, Won't have) or numerical ranking are used for prioritization.

Requirement Traceability: This involves establishing and maintaining links between requirements and other project artifacts, such as design documents, test cases, and code. Traceability helps ensure that each requirement is implemented correctly and that changes are appropriately reflected in related artifacts.

Requirements Documentation and Communication: Requirements need to be documented and communicated effectively to all stakeholders. Clear and concise documentation helps prevent misunderstandings and misinterpretations.

Requirements Evolution: Requirements engineering is not a one-time activity but an ongoing process. As the project progresses, requirements may evolve based on changing business needs or new insights.

already done.

Develop the SRS for Hospital Management System. Hospital Management System is a process of implementing all the activities of the hospital in a computerized automated way to fasten the performance. This system is to maintain the patient details, lab reports and to calculate the bill of the patient. You can also manually edit any patient details and issue bill receipt to patients within a few seconds. SRS for the hospital Management system should include the following:(a)Product perspective (b) Scope and objective (c) Functional requirements (d) Non-functional requirements

a). Product Perspective:

The Hospital Management System (HMS) is a comprehensive software solution designed to automate and streamline the various activities within a hospital. It will serve as a central platform to integrate and manage patient information, lab reports, and billing processes. The system will be developed as a standalone application and can also be integrated with other hospital systems if required.

b). Scope and Objective:

Scope:

The Hospital Management System will cover the following aspects of hospital operations:

Patient Registration: Capture and manage patient demographics, contact information, and medical history.

Lab Reports Management: Store and retrieve laboratory test reports and results.

Billing and Invoicing: Calculate and generate bills for patients based on their treatment and services received.

Appointment Scheduling: Schedule and manage appointments for patients and doctors.

Inventory Management: Track and manage hospital inventory, including medicines and medical equipment.

Staff Management: Manage information related to hospital staff, including doctors, nurses, and administrative personnel.

Report Generation: Generate various reports for administrative and decision-making purposes.

User Authentication and Authorization: Ensure secure access to the system with role-based permissions.

Objectives:

The primary objectives of the Hospital Management System are:

Improve efficiency and accuracy in patient information management.

Enhance the quality of patient care by providing easy access to medical records.

Expedite billing and invoicing processes, reducing errors.

Streamline appointment scheduling for both patients and healthcare providers.

Ensure the security and confidentiality of patient data.

Provide a user-friendly and intuitive interface for hospital staff.

c). Functional Requirements:

Patient Registration:

- Capture patient demographics (name, age, gender, address, etc.).
- Record medical history and allergies.
- Assign a unique patient ID.
- Update and edit patient information as needed.

Lab Reports Management:

- Upload and store lab test reports.
- Associate lab reports with patients.
- Allow authorized personnel to view and download reports.

Billing and Invoicing:

- Calculate bills based on treatment, services, and medicines.
- Generate itemized bills for patients.
- Record payments and maintain billing history.

Appointment Scheduling:

- Schedule appointments for patients with doctors.
- Manage appointment slots and availability.
- Send appointment reminders to patients.

Inventory Management:

- Track inventory levels of medicines and medical equipment.
- Alert when stock levels are low.
- Generate purchase orders for restocking.

Staff Management:

- Maintain records of doctors, nurses, and administrative staff.
- Assign roles and permissions to staff members.
- Manage staff schedules.

Report Generation:

- Generate reports such as patient demographics, billing summaries, and inventory status.
- Export reports in various formats (PDF, Excel, etc.).

User Authentication and Authorization:

- Implement secure login and authentication mechanisms.
- Define roles (admin, doctor, nurse, receptionist) with specific permissions.

d). Non - functional Requirements:

Performance: The system should respond to user requests within acceptable response times even under peak load conditions.

Security: Patient data should be stored securely and access should be restricted based on user roles.

Usability:

The system should have an intuitive user interface to minimize the learning curve for hospital staff.

Accessibility features should be considered for users with disabilities.

Reliability:

The system should be available 24/7 with minimal downtime for maintenance.

Regular backups of data should be performed to ensure data integrity.

Scalability: The system should be able to scale to accommodate a growing number of patients, doctors, and staff.

Agile , scrum and Kanban with examples, diagrams

Explain CMM and Key process Areas at each levels :

The Capability Maturity Model (CMM) is a framework that defines the maturity levels of an organization's processes in software engineering and software development.

The CMM framework helps organizations assess and improve their processes, with each level representing a higher degree of maturity and capability.

There are five levels in the CMM, each with its key process areas:

Level 1 - Initial:

At this level, processes are ad hoc, chaotic, and often unpredictable. Organizations have no defined process structure, and success depends on individual heroics rather than a consistent process. There are no key process areas defined at this level.

There are no key process areas defined at this level.

Level 2 - Repeatable:

Basic management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

- Project Planning: Developing plans to guide project execution.
- Project Monitoring and Control: Tracking progress and taking corrective actions as needed.

Level 3 - Defined:

The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization.

All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.

- Organization Process Definition: Documenting and improving processes to make them repeatable.
- Training Program: Providing training to staff to ensure competence.

Level 4 - Managed:

Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.

- Software Quality Management: Ensuring that products meet quality standards.
- Defect Prevention: Identifying and eliminating the root causes of defects.

Level 5 - Optimizing:

Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

Goal is to prevent the occurrence of defects

- Technology Change Management: Managing changes to technology tools and environments.
- Defect Reduction: Reducing the rate of defects through process improvement.

What are the 3P's of Project Management? What is Project metrice

People: People are at the core of every project, including project managers, team members, stakeholders, and end-users. Effective project management involves selecting the right individuals for the project team, defining their roles and responsibilities, and ensuring effective communication and collaboration among team members. Motivated and skilled people are key to achieving project objectives.

Process: A well-defined and structured project management process is essential for planning, executing, and controlling the project. Processes involve activities such as project initiation, planning, execution, monitoring and controlling, and project closure. A clear and standardized process helps ensure that the project is executed efficiently, with defined workflows and procedures to follow.

Product: The "product" in the context of project management refers to the deliverables or the end result of the project. This includes the actual output, whether it's a software application, a construction project, a marketing campaign, or any other project type. Project managers must ensure that the product aligns with the project's objectives and meets the requirements and expectations of stakeholders.

Project Metrics: Project metrics are quantitative measurements or indicators used to assess the progress, performance, and health of a project. They provide valuable data and insights to project managers, team members, and stakeholders for making informed decisions and managing the project effectively. Some common project metrics include:

Schedule Metrics: These metrics assess **project timelines and deadlines**. Examples include:

- **Project schedule variance:** The difference between the planned schedule and the actual progress.
- **Schedule performance index (SPI):** A ratio of actual work performed to the work planned.

Cost Metrics: These metrics **track project expenditures and budget adherence.** Examples include:

- **Cost variance:** The difference between the planned budget and the actual costs.
- **Cost performance index (CPI):** A ratio of the earned value to the actual cost incurred.

Effective project management relies on a combination of the 3P's (People, Process, Product) and the use of appropriate project metrics to guide decision-making and ensure project success.

What is coupling and cohesion?

Coupling:

- Coupling refers to the degree of interdependence or connection between different modules or components within a software system. It measures how closely one module relies on or interacts with another.
- Low coupling is generally desirable because it indicates that the components are relatively independent and changes to one component are less likely to impact others.
- High coupling can lead to a fragile and difficult-to-maintain system.

Types of coupling:

Low Coupling (Loose Coupling): Components have minimal dependencies on each other, making them easier to modify and maintain independently.

High Coupling (Tight Coupling): Components have significant dependencies, and changes in one component can have a cascading effect on others.

Cohesion:

- Cohesion refers to the degree to which the elements (e.g., functions, methods, or classes) within a module or component are related to a common purpose or functionality.
- High cohesion means that the elements within a module are tightly related and work together to achieve a specific goal.
- Low cohesion indicates that the elements within a module have diverse and unrelated purposes, making the module less focused and harder to understand and maintain.

Types of cohesion:

Functional Cohesion: Elements within a module are grouped together because they perform related functions or tasks. This is the highest level of cohesion and is considered ideal.

Sequential Cohesion: Elements are related because they are executed in a specific order.

Balancing coupling and cohesion is essential in software design. The goal is to minimize coupling between modules while maximizing cohesion within each module. This approach leads to modular, maintainable, and scalable software systems.

Explain with a suitable diagram SCRUM agile model.

Scrum is an Agile framework for iterative and incremental software development. It is based on a set of roles, events

Roles in Scrum:

Product Owner: The Product Owner is responsible for defining and prioritizing the product backlog, which is a prioritized list of features and requirements. They represent the customer's interests and ensure that the development team works on the most valuable items.

Scrum Master: The Scrum Master serves as a facilitator and coach for the Scrum team. They ensure that Scrum practices and principles are followed, remove impediments, and help the team work effectively.

Development Team: The Development Team consists of professionals who do the actual work of delivering a potentially shippable product increment in each sprint. They self-organize and cross-functionally collaborate to achieve their goals.

Scrum Events:

Sprint: A Sprint is a time-boxed iteration, typically lasting 2-4 weeks, where a potentially shippable product increment is developed. Sprints provide a predictable cadence for development work.

Sprint Planning: At the beginning of each Sprint, the team conducts Sprint Planning to select items from the product backlog and determine how they will be completed during the Sprint.

Daily Scrum (Daily Standup): The Daily Scrum is a daily 15-minute meeting where team members discuss what they worked on, what they plan to do next, and any impediments they are facing.

Sprint Review: At the end of each Sprint, the team holds a Sprint Review to demonstrate the completed work to stakeholders and gather feedback.

Sprint Retrospective: After the Sprint Review, the team holds a Sprint Retrospective to reflect on the Sprint and identify opportunities for improvement.

MODULE 3:

Function points:

Function Points (FP) are a unit of measurement used in software engineering to quantify the functionality provided by a software application.

Function Points are a size metric that focuses on the functionality offered by a software system, independent of the technology, programming language, or implementation details.

They are valuable for estimating project effort, measuring productivity, and comparing the size and complexity of different software applications.

External Inputs (EI): These are user interactions that provide data to the system. Examples include user input forms, file uploads, and API calls that pass data to the application.

External Outputs (EO): These are user interactions that result in data being presented by the system. Examples include generating reports, displaying search results, or returning data in response to an API request.

External Inquiries (EQ): These are user interactions that both retrieve and present data. For example, a user querying a database and displaying the results on the screen would be considered an External Inquiry.

Internal Logical Files (ILF): These represent internal data maintained by the application. ILFs include databases, data files, or any data store used within the software.

External Interface Files (EIF): These represent data referenced by the software but maintained by external applications or systems. EIFs include data files or databases not under the control of the application.

Function Points are calculated using **a specific formula**, which involves assigning complexity weights (Low, Average, or High) to each of the components mentioned above. The formula generally follows these steps:

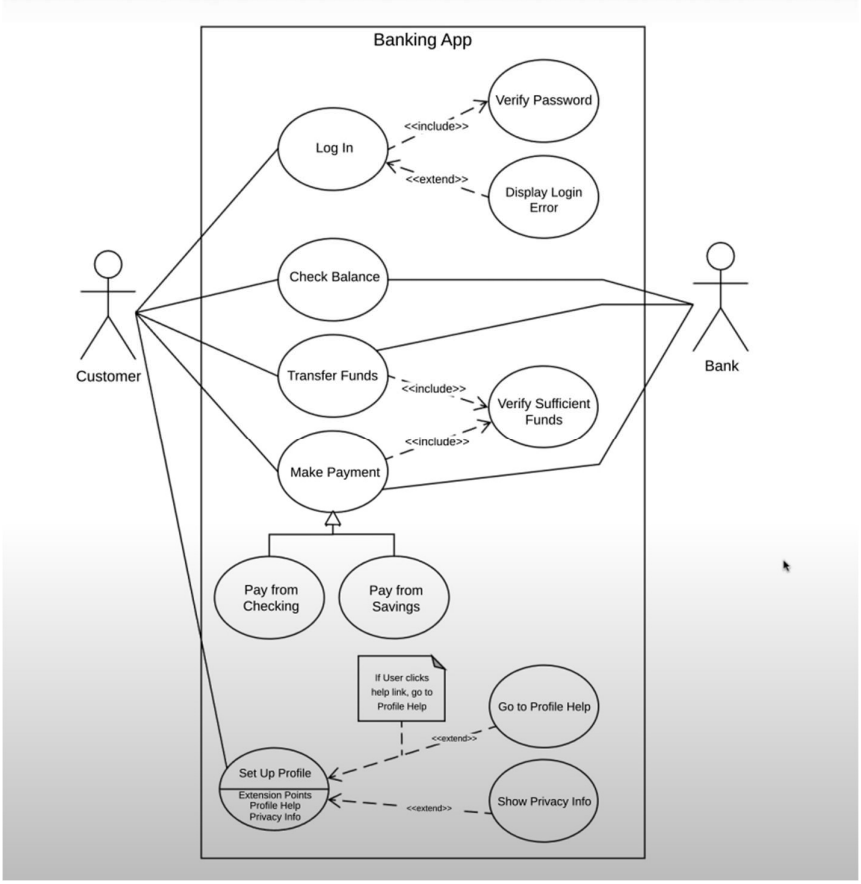
1. Count the number of each type of component (EI, EO, EQ, ILF, EIF) in the application.
2. Assign a complexity weight (Low, Average, or High) to each component based on criteria such as data complexity, processing complexity, and user interface complexity.
3. Multiply the count of each component by its assigned complexity weight.
4. Sum the weighted values for all components to calculate the total Function Points.
5. The resulting Function Point count provides a measure of the functional size of the software. This count can be used for various purposes, including:

Function Points focus on functionality and do not consider other factors like code quality, performance, or non-functional requirements.

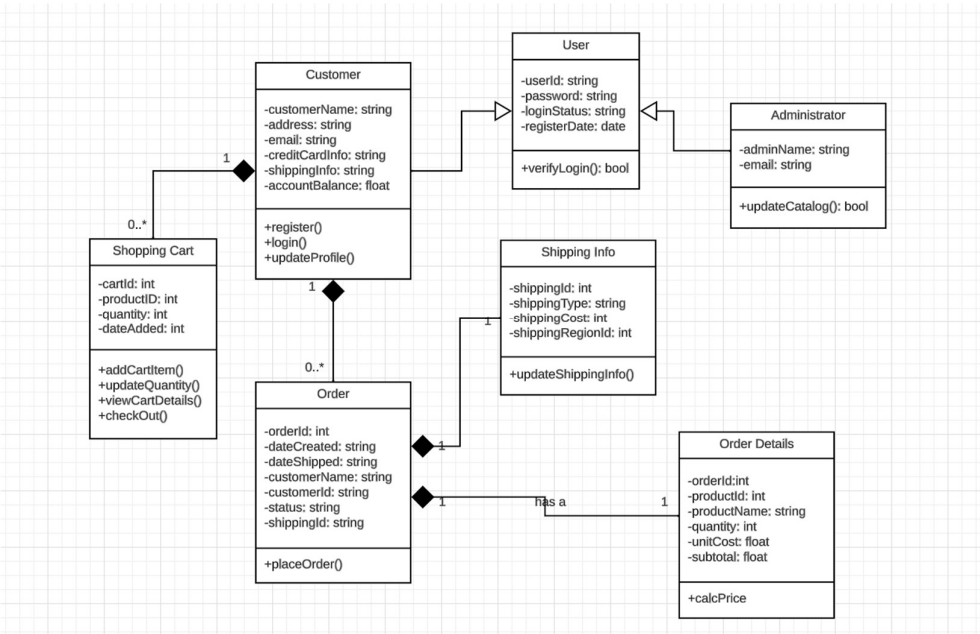
They are just one tool in the toolkit of software measurement and estimation and are often used in combination with other metrics and estimation techniques for a comprehensive assessment of software projects.

UML DIAGRAMS

USE CASE DIAGRAM :



CLASS DIAGRAM:



SEQUENCE DIAGRAM: