Software Requirement

1. Discuss the significance and use of requirement engineering. What are the problems in the formulation of requirements?

Requirement Engineering (RE) is a fundamental phase in the software development life cycle. It involves the systematic process of **identifying, documenting, analysing, and managing the requirements** of a software system from various stakeholders.

Significance:

1. **Foundation for Development:** RE provides a **clear and agreed-upon foundation** for designing, developing, and validating software systems.
2. **Stakeholder Communication:** It ensures that the expectations and needs of users, clients, and other stakeholders are well understood and agreed upon.
3. **Risk Reduction:** Good requirements help **reduce misunderstandings**, scope creep, and project failures.
4. **Improved Quality:** Clearly defined and validated requirements contribute to **better quality software** and user satisfaction.
5. **Cost Efficiency:** Identifying errors or ambiguities in requirements early saves time and cost, as fixing defects in later stages is more expensive.

As per the PDF, RE involves steps like **feasibility study, requirements elicitation, analysis, specification, and validation**, which together ensure the correctness and completeness of the software requirements.

Problems in the Formulation of Requirements

Despite its importance, requirement engineering faces several challenges:

1. **Ambiguity and Incompleteness:** Requirements are often **not clearly or fully defined**, leading to different interpretations by stakeholders and developers.
2. **Changing Requirements:** Stakeholders may **change their needs** during the project, making it difficult to maintain stability.
3. **Communication Gaps:** Miscommunication between users and developers can lead to **incorrect or misunderstood requirements**.
4. **Conflicting Requirements:** Different stakeholders might have **incompatible needs**, making it hard to arrive at a consistent set of requirements.
5. **Poor Elicitation Techniques:** If the right methods (like interviews, questionnaires, or observations) aren't used, **key requirements may be missed**.
6. **Lack of Domain Knowledge:** Analysts may lack **understanding of the application domain**, leading to incorrect assumptions.

2. What are crucial process steps of requirement engineering? Discuss with the help of a diagram.

1. Feasibility Study

- Assesses whether the software project is **technically, operationally, and economically feasible**.
- Helps in deciding **whether to proceed** with the project.

2. Requirement Elicitation

- Also known as requirement gathering.
- Involves **collecting requirements** from stakeholders through techniques like **interviews, questionnaires, brainstorming, and observation**.

3. Requirement Analysis

- Involves examining the gathered requirements to ensure they are **complete, clear, consistent, and non-conflicting**.
- Focuses on **understanding the requirements deeply** and resolving any ambiguities.

4. Requirement Specification

- The requirements are documented in a **Software Requirements Specification (SRS)**.
- The SRS document serves as a formal agreement between stakeholders and developers.

5. Requirement Validation

- Ensures that the documented requirements truly reflect the **customer's needs and expectations**.

- Involves **reviews, prototyping, and test case generation**.

6. Requirement Management

- Deals with **tracking and managing requirement changes** during the software lifecycle.
- Involves **version control, traceability**, and impact analysis

3. Explain the importance of requirements. How many types of requirements are possible and why ?

Why Requirements are Important:

1. **Clear Understanding of Needs:**
    - Requirements capture **user needs and expectations**, ensuring developers build the right product.
2. **Basis for Planning:**
    - They help in **estimating cost, time, and resources** required for the project.
3. **Improved Quality:**
    - Well-defined requirements lead to **better design, accurate coding, and effective testing**.
4. **Reduced Rework:**
    - Early detection of issues through validated requirements helps avoid **costly corrections later**.
5. **Better Communication:**
    - Act as a **common reference** between stakeholders, developers, testers, and project managers.
6. **Support for Validation and Verification:**
    - Enable **systematic testing** to check whether the final software meets its intended purpose.

According to the PPT, errors in requirements are a **major cause of project failure**, making it essential to spend adequate time and effort on requirement engineering.

Types of Requirements and Why They Are Needed

Requirements can be broadly classified into three main types:

1. Functional Requirements

- **Definition:** Describe **what the system should do** — its features, functions, and behavior.
- **Examples:**
    - User login/logout
    - Generating reports
    - Performing calculations
- **Why Needed:** They define the **core functionality** expected by users.

---

2. Non-Functional Requirements (NFRs)

- **Definition:** Define **how the system should behave**, focusing on **quality attributes**.
- **Examples:**
    - Performance (e.g., response time < 2 sec)
    - Security (e.g., encrypted data transmission)
    - Usability and scalability
- **Why Needed:** They ensure the system is **reliable, secure, and user-friendly**, not just functionally correct.

---

3. Domain Requirements

- **Definition:** Requirements specific to the **application domain** or industry.
- **Examples:**
    - Medical software must comply with HIPAA.
    - Banking apps need secure transaction protocols.
- **Why Needed:** They ensure the software complies with **industry regulations and norms**.

4. What do you understand with the term “requirements elicitation” ? Discuss any two techniques in detail.

Requirement elicitation is the process of **gathering requirements** from stakeholders such as clients, users, and subject matter experts. It is the **first and one of the most crucial activities** in the requirement engineering process.

As defined in the PPT, requirement elicitation is the process of “understanding and collecting requirements from stakeholders through various communication and observation techniques.”

- Ensures that developers understand **what users truly need**.
- Helps avoid **missing or misunderstood requirements**.
- Forms the **foundation for analysis, specification, and validation** phases.

1. Interviews

- **Description:**
    1. Conduct face-to-face or virtual discussions with stakeholders.
    2. Can be structured, semi-structured, or unstructured.
    3. Helps gather qualitative and in-depth information.
- **Advantages:**
    1. Builds strong communication with stakeholders.
    2. Allows clarification of ambiguous requirements.
    3. Helps uncover hidden needs or constraints.
- **Limitations:**
    1. Time-consuming and may require multiple sessions.
    2. Requires skilled interviewer to avoid bias.
    3. Difficult to schedule with all key stakeholders.

2. Observation

- **Description:**
    1. Involves watching users perform actual tasks.
    2. Can be passive (silent watching) or active (with interaction).
    3. Helps capture real workflows and user behavior.
- **Advantages:**
    1. Reveals non-verbal and informal task flows.
    2. Useful when users cannot articulate their needs.
    3. Identifies inefficiencies and workarounds in current systems.
- **Limitations:**
    1. User behavior may change due to being observed.
    2. May not capture rare but critical use cases.
    3. Time-intensive and may require multiple observations.

3. Questionnaires/Surveys

- **Description:**
    1. Use pre-defined questions sent to stakeholders.
    2. Suitable for collecting quantitative data.
    3. Can be online or paper-based.
- **Advantages:**
    1. Reaches large and geographically dispersed users.
    2. Cost-effective with quick turnaround.
    3. Easy to analyze and summarize results.
- **Limitations:**
    1. Limited depth of understanding.
    2. No way to clarify unclear responses.
    3. Risk of low response rate or incomplete answers.

4. Brainstorming

- **Description:**
    1. Group session where participants freely share ideas.
    2. Encourages spontaneous thinking without criticism.
    3. Typically led by a facilitator.
- **Advantages:**
    1. Generates a wide range of creative ideas quickly.
    2. Encourages collaboration and involvement.
    3. Useful in early stages to explore possible directions.
- **Limitations:**
    1. May lack structure or prioritization.
    2. Dominant personalities can influence outcomes.
    3. Risk of diverging from the actual problem scope

5. Explain the use case approach of requirements elicitation. What are use-case guidelines ?

What is a Use Case?

A **use case** is a description of how users (called **actors**) interact with a system to achieve a specific goal.
It represents a **functional requirement** in a **user-centric way**.

In requirement elicitation, the use case approach focuses on identifying all possible ways a system will be used by various actors to fulfill business objectives.

Purpose of Use Case in Requirements Elicitation:

1. **Captures user interactions clearly** in terms of goals and system behavior.
2. Helps in understanding the **system boundary**, users, and expected outcomes.
3. Facilitates communication between **technical and non-technical stakeholders**.

Components of a Use Case:

- **Actor:** External entity interacting with the system (e.g., user, another system).
- **Use Case Name:** Descriptive title (e.g., "Place Order").
- **Precondition:** Conditions that must be true before use case starts.
- **Main Flow:** Normal sequence of interactions between actor and system.

- **Alternate Flow:** Variations or exceptions to the main scenario.
- **Postcondition:** Result after successful completion.

Use Case Guidelines (Best Practices):

1. **Identify All Actors:**
    - Include primary (e.g., user) and secondary (e.g., payment gateway) actors.
2. **Define Each Use Case Clearly:**
    - Use **simple and consistent language**.
    - One use case should address **one specific user goal**.
3. **Use Diagrams and Text Together:**
    - Combine **use case diagrams** (for visual overview) and **use case descriptions** (for detailed behavior).
4. **Include Preconditions and Postconditions:**
    - Helps ensure **logical flow** and **state changes** in the system.
5. **Include Alternate Flows:**
    - Consider **what can go wrong** or **optional paths** to make requirements more complete.
6. **Review with Stakeholders:**
    - Ensure all use cases are **understood, validated, and approved** by users.

6. Consider the problem of library management system and design the following: (i) Problem statement (ii) Use case diagram (iii) Use cases.

The Library Management System (LMS) is a software application designed to manage and automate the day-to-day operations of a library. It aims to streamline processes such as book borrowing, returning, tracking, and managing member information. The system should allow both library staff and patrons to interact efficiently with the library resources.

Key features include:

- **Member management:** Registration, login, and tracking of borrowing history.
- **Book management:** Addition, removal, and searching of books.
- **Issue and return management:** Checking books out to members and accepting returns.
- **Late fee management:** Charging fines for overdue books.
- **Search functionality:** Allowing patrons and staff to search for books by title, author, or genre.
- **Book availability tracking:** Displaying the current status (available/borrowed) of books.
- **Admin management:** Admin users should be able to manage books, users, and handle system settings.

(ii) Use Case Diagram

A use case diagram for the Library Management System might look like this:

- **Actors:**
    - **Admin**: Manages the system settings, books, and user accounts.
    - **Librarian**: Handles book issuing and returning, manages overdue fees.
    - **Member**: Registers, borrows, returns books, and searches for books.

(iii) Use Cases

1. **Admin Use Cases:**
    - **Add Book:** Admin can add new books to the library system by specifying details like title, author, genre, and availability.
    - **Remove Book:** Admin can remove books that are no longer available or needed in the library.
    - **Manage Users:** Admin can register, update, or delete user accounts, as well as assign roles to staff or patrons.
    - **System Configuration:** Admin can modify settings related to fines, book limits, and other system parameters.
2. **Librarian Use Cases:**

- **Issue Book:** Librarian can issue books to members by checking the availability of books and registering the transaction in the system.
- **Return Book:** Librarian receives returned books, updates the availability status, and calculates any overdue fees.
- **Track Overdue Books:** Librarian monitors overdue books and charges fees based on the duration of delay.

3. **Member Use Cases:**
   - **Register Account:** Member can create an account in the system with personal details.
   - **Borrow Book:** Member can borrow available books from the library.
   - **Return Book:** Member can return borrowed books before or after the due date.
   - **Search Books:** Member can search for books by title, author, or genre to check availability.
   - **Pay Late Fee:** If applicable, members can pay any overdue fines generated from returning books late.
4. **System Use Cases:**
   - **Search Books:** The system provides functionality for searching books in the catalog.
   - **Track Availability:** The system maintains real-time data about book availability.
   - **Generate Reports:** The system generates reports on overdue books, total fines collected, and the number of books borrowed over a period.

7. (a) Functional vs Non-functional Requirements

Aspect	Functional Requirements	Non-functional Requirements
Definition	Describe **what** the system should do (specific behaviors or functions).	Describe **how** the system should perform or operate (qualities, constraints).
Focus	System **features**, tasks, and interactions.	System **performance**, usability, reliability, etc.
Examples	- The system shall allow users to log in. - The system shall allow members to borrow books. - Admin can add or remove books.	- The system should load search results within 2 seconds. - The system must be available 24/7. - The interface should be user-friendly.
Type of Requirement	Typically derived from **user needs or business processes**.	Often derived from **quality standards or stakeholder expectations**.
Verification	Can be tested directly with test cases.	May require performance testing or usability analysis.

(b) User Requirements vs System Requirements

Aspect	User Requirements	System Requirements
Definition	High-level requirements written in natural language describing what users expect the system to do.	Detailed and technical specifications describing how the system will fulfill user requirements.
Audience	Intended for **end-users** and **stakeholders**.	Intended for **developers**, **testers**, and **technical teams**.
Level of Detail	General and **non-technical**.	Detailed and **technical**.
Format	Often described as user stories or functional summaries.	Includes functional and non-functional specs, models, diagrams, etc.
Example	"A member should be able to search for books by title."	"The system shall allow keyword-based search in the 'Books' table using SQL LIKE queries with a response time under 3 seconds."

8. Difference between Use Case and ER- Diagram.

Aspect	Use Case Diagram	ER (Entity-Relationship) Diagram
Purpose	To model **functional behavior** of a system	To model **data and relationships** between entities
Focus	Focuses on **what the system does** (interactions)	Focuses on **what data the system stores** and how it's related
Represents	Actors, use cases, and their interactions	Entities, attributes, and relationships
Used In	**Requirement analysis and system behavior**	**Database design and data modeling**
Components	Actor, Use Case, System Boundary, Associations	Entity, Attribute, Relationship, Cardinality
Example Elements	User → Login, Register, Search Book	Member (Entity) has Name, ID; Borrow (Relationship) with Book
Diagram Type	Part of **UML (Unified Modeling Language)**	Part of **Data modeling** (often in DB design)

QFT (Quality Function Deployment)

Definition:
Quality Function Deployment (QFT) is a customer-driven planning tool used to translate customer needs (also called "voice of the customer") into engineering characteristics for a product or service.

Purpose:
To ensure the product or system design meets customer expectations right from the start.

Key Tool:

House of Quality – A matrix that maps customer requirements to technical specifications.

Steps Involved:

1. Identify customer needs (WHATs).
2. Determine how the company will meet those needs (HOWs).
3. Relate the WHATs to the HOWs.
4. Set priorities and target values.

Benefits:

- Aligns development with customer expectations.
- Enhances communication between departments.
- Improves product quality and customer satisfaction.

FAST (Function Analysis System Technique)

Definition:
FAST is a technique used to analyze and organize the functions of a product, system, or process logically.

Purpose:
To understand **what a product or system does**, not how it does it — especially useful in value engineering and problem-solving.

Key Idea:

- Uses **"How" and "Why" questions** to build a function hierarchy.
- Main question: "**How does this function happen?**" and "**Why is this function needed?**"

Diagram Used:

- **FAST Diagram** – A flow chart showing logical relationships between functions.

Benefits:

- Helps break down complex problems.
- Supports innovation and cost-effective design.
- Useful in both product design and process improvement.

Software Design

1. **What is design? Describe the difference between conceptual design and technical design.**

Design in software engineering refers to the process of transforming user requirements into a blueprint for constructing a software system. It involves defining the system's architecture, components, interfaces, and data to satisfy specified requirement

Aspect	**Conceptual Design**	**Technical Design**
Purpose	**Defines *what* the system should do from a user's perspective.**	**Specifies *how* the system will fulfill the requirements.**
Focus Areas	**User interactions, data flow, system behavior, and high-level structure.**	**System architecture, hardware and software specifications, communication interfaces, and detailed component design.**

Key Questions	**Where will the data come from ?** **What will happen to data in the system?** **How will the system look to users?** **What choices will be offered to users?** **What is the timings of events?** **How will the reports & screens look like?**	**Hardware configuration** **Software needs** **Communication interfaces** **I/O of the system** **Software architecture** **Network architecture** **Any other thing that translates the requirements in to a solution to the customer's problem.**
Audience	**Stakeholders, clients, and end-users.**	**Developers, system architects, and technical teams.**
Language Used	**Non-technical, user-friendly language.**	**Technical terminology and specifications.**
Outcome	**A model or prototype illustrating the system's functionality and user interface.**	**Detailed design documents guiding the implementation phase.**

2. **What is the difference between a flow chart and a structure chart?**

A flowchart and a structure chart are both visual tools used in software development, but they serve different purposes and convey distinct aspects of a system.

A flowchart is a diagrammatic representation of an algorithm or process. It illustrates the sequence of operations to be performed to solve a problem or complete a task. Flowcharts use standardized symbols connected by arrows to depict the flow of control from one step to another.

Key Characteristics:

- **Purpose: To visualize the step-by-step flow of control in a process or program.**
- **Focus: Emphasizes the sequence of operations and decision-making paths.**
- **Symbols: Utilizes simple symbols like rectangles (processes), diamonds (decisions), ovals (start/end), and arrows (flow direction).**
- **Usage: Commonly used in algorithm design, process documentation, and troubleshooting.**

Example: **A flowchart can represent the logic of a login process, showing steps like inputting credentials, verifying them, and granting or denying access.**

A structure chart is a hierarchical diagram that depicts the organization of a system's modules and their relationships. It breaks down a system into smaller components (modules), illustrating how they interact and the flow of data between them.

Key Characteristics:

- **Purpose: To represent the software's modular structure and the relationships between modules.**
- **Focus: Emphasizes the hierarchy and interaction of modules rather than the sequence of operations.**
- **Symbols: Uses boxes to represent modules and arrows to indicate control or data flow between them.**
- **Usage: Utilized during the design phase to plan and organize the system's architecture.**

Example: **A structure chart for a payroll system might show modules like "Calculate Pay," "Deduct Taxes," and "Generate Payslip," indicating how these modules interact.**

3. **Discuss the differences between object oriented and function oriented design.**

Aspect	**Object-Oriented Design (OOD)**	**Function-Oriented Design (FOD)**
Primary Focus	**Objects encapsulating data and behavior**	**Functions and the sequence of operations**
Approach	**Bottom-up**	**Top-down**
Modularity	**Based on objects/classes**	**Based on functions/modules**
Data Handling	**Encapsulated within objects**	**Shared across functions**
Reusability	**Achieved through inheritance and polymorphism**	**Achieved through function reuse**
Design Tools	**UML diagrams, class hierarchies**	**Data flow diagrams, structure charts**

Best Suited For	**Complex, scalable, and real-world modeling applications**	**Simple, linear, and computation-heavy applications**