Module 1

1. Define system software engineering.

- **Definition**: System software engineering is the process of applying engineering principles, tools, and methods to develop and maintain system-level software.
- **Purpose**: It deals with software like operating systems, compilers, assemblers, and device drivers that control hardware and support applications.
- **Objective**: Its main aim is to ensure efficiency, reliability, security, and smooth functioning of both hardware and user applications.

2. List any 4 attributes of good software.

- **Functionality**: The software must perform all the required tasks correctly and meet user needs.
- **Reliability**: It should consistently work without failures, ensuring stability during operation.
- **Efficiency**: Uses system resources like CPU, memory, and storage effectively without wastage.
- **Maintainability**: Should be easy to update, fix bugs, and improve when technology changes.

3. State the two types of software with example.

- **System Software**: Software that manages computer hardware and provides a base for applications.
  - *Example*: Windows, Linux, or device drivers.
- **Application Software**: Programs designed to perform specific tasks for the user.
  - *Example*: MS Word, WhatsApp, or Photoshop.
- **Purpose**: Together, both ensure the computer functions effectively for system control and user tasks.

4. Identify the key challenges faced by software engineers.

- **Complexity**: Handling large-scale and technically complex projects is difficult.
- **Changing Requirements**: Clients often change needs during development, causing delays.
- **Quality Assurance**: Maintaining high performance, security, and bug-free software is challenging.
- **Time & Cost**: Meeting deadlines and staying within the budget requires strict planning.

5. Recall the importance of system engineering in software development.

- **Definition**: System engineering integrates hardware, software, and processes into a complete working system.
- **Purpose**: Ensures that all parts of a project work together smoothly without conflict.
- **Importance**: Helps reduce risks, improves quality, supports large-scale projects, and ensures successful system integration.

6. Explain the characteristics of software.

- **Intangible**: Unlike hardware, software cannot be touched or physically seen.
- **Flexible**: It can be modified, updated, or improved as per user requirements.
- **Non-wearable**: Software does not wear out with use, but may fail due to bugs or errors.
- **Customizable**: Can be developed to suit specific needs, industries, or organizations.
- **Dependent**: Always requires hardware to execute its functions.

7. Discuss how software engineering helps in reducing cost.

- **Definition**: Software engineering is a systematic approach to building reliable software.
- **Process**: It uses proper planning, requirement analysis, design, testing, and documentation.
- **Advantage**: Detecting errors early reduces rework and prevents costly failures.
- **Result**: Saves time, labor, and resources, leading to reduced overall development cost.

8. Illustrate the difference between software and hardware.

- **Software**: A set of programs and instructions that direct a computer to perform tasks. It is intangible and stored in digital form.
- **Hardware**: The physical parts of a computer system such as CPU, RAM, keyboard, and mouse.
- **Key Difference**: Software tells hardware *what to do*, while hardware performs the actual actions.
- **Relation**: Both depend on each other; hardware is useless without software and vice versa.

9. Classify different types of software engineering cost.

- **Development Cost**: Cost of requirement gathering, design, coding, and testing.
- **Quality Assurance Cost**: Expenses on reviews, verification, validation, and testing tools.
- **Maintenance Cost**: Cost for fixing bugs, updating features, and adapting to new needs.
- **Management Cost**: Includes project planning, coordination, documentation, and team management.
- **Support Cost**: Cost of training, installation, and technical support for users.

10. Summarize the benefits of using a systematic software process.

- **Consistency**: Provides a clear step-by-step approach, ensuring proper development.
- **Quality**: Leads to more reliable, secure, and error-free software products.
- **Efficiency**: Reduces duplication of work and improves productivity of developers.
- **Predictability**: Makes it easier to estimate time, cost, and resources required.
- **Customer Satisfaction**: Delivers software that meets user needs within budget and deadlines.

Module 2

1. Explain the concept of software process model.

- A **software process model** is a structured framework that defines the steps to be followed in software development.
- It organizes activities like **requirement analysis, system design, coding, testing, deployment, and maintenance**.
- Different models such as **Waterfall, Iterative, Spiral, RAD, and Agile** are used depending on project type.
- It provides a **clear roadmap** so that developers and clients know how the software will be created.
- By following a process, chances of error reduce and efficiency improves.
- It ensures that work is systematic instead of random, saving both **time and cost**.
- Overall, it brings **discipline, order, and quality assurance** to software development.

2. Summarize the characteristics of waterfall model.

- The **Waterfall model** is a **linear and sequential** development process.
- It follows strict phases: **Requirement → Design → Implementation → Testing → Deployment → Maintenance**.
- Each phase must be completed before the next one begins, making it **rigid and structured**.
- It produces **clear documentation** at every stage for better understanding.
- The model is simple and easy to use, especially for beginners.
- Best suited for **small projects** where requirements are stable and well-defined.
- Limitation: Difficult to go back if any change is needed later, making it **costly to fix errors**.

3. Discuss the process of process iteration (Priority, Advantage, Disadvantage).

- **Process Iteration** means repeating stages of development to refine and improve the software.
- **Priority**: Important when requirements are unclear or changing, as iteration allows corrections.
- **Advantage**: Customers can see working versions early, give feedback, and improvements can be made quickly.
- It reduces risks because problems are handled step by step instead of all at once.
- Enhances software quality by adding features gradually and fixing issues earlier.
- **Disadvantage**: Too many iterations may lead to **higher cost and longer time**.
- It may also confuse planning because the final scope keeps changing with every cycle.

4. Identify the Benefits of incremental delivery.

- In **Incremental delivery**, software is built and delivered in small parts (increments).
- Users can use the **basic version early** while additional features are added later.
- This improves **customer satisfaction** as they see progress quickly.
- It allows flexibility because features can be added or changed with each increment.
- Bugs can be detected early in small parts instead of the full system.
- It reduces risk since failure of one increment doesn’t stop the entire project.
- Useful for projects where **time-to-market is important**.

5. Apply RAD model to a time critical small project (Context: College).

- **RAD (Rapid Application Development)** is focused on fast software creation using reusable tools and prototypes.
- In a college project, like **Student Attendance System**, RAD can deliver results quickly.
- Students and teachers can test prototypes early and give feedback for improvements.
- It allows making quick changes to adapt to college requirements.
- RAD uses pre-built modules and components, reducing development time.
- This makes it suitable for **small projects with strict deadlines** such as semester submissions.
- The final outcome is fast, efficient, and user-approved software within the time limit.

6. Discuss the property of RAD model.

- **Component-Based**: Uses reusable modules, templates, and tools to save time.
- **High Speed**: Designed for rapid development, delivering results quickly.
- **User-Centric**: Involves constant user feedback at every stage.
- **Prototyping**: Builds prototypes for early evaluation and refinement.
- **Flexibility**: Easy to modify design and features based on user needs.
- **Suitability**: Best for **small to medium projects with short deadlines**.
- **Goal**: Deliver quality software quickly with active customer involvement.

7. Illustrate the benefits of Agile model over waterfall model.

- **Flexibility**: Agile adapts easily to changing requirements, unlike rigid Waterfall.
- **Early Delivery**: Agile delivers working software in short cycles, Waterfall delivers only at the end.
- **Customer Involvement**: Agile involves clients throughout; Waterfall involves them mainly at the start and end.
- **Risk Reduction**: Issues are solved early in Agile; in Waterfall, problems may appear late.
- **Faster Feedback**: Agile collects regular feedback, ensuring user satisfaction.
- **Productivity**: Teams stay more motivated and efficient due to short iterations.
- Overall, Agile is **modern and dynamic**, while Waterfall is **old and rigid**.

8. Demonstrate component-based development in real application.

- **Component-Based Development (CBD)** uses reusable software parts (modules, APIs, plugins) to build applications.
- Each component is self-contained and can be plugged into different projects.
- Example: A college website can use **Google Maps API** for directions, a **payment gateway plugin** for fees, and a **login module** for students.
- Developers save time by reusing existing components instead of coding from scratch.
- It reduces cost because tested components are reliable and need less maintenance.
- It also improves **scalability**, as new modules can be added easily.
- CBD makes projects faster, flexible, and more secure.

9. Use prototyping for faster early feedback (Iterative model).

- **Prototyping** is the process of creating an early sample version of the software.
- Users can see a **working model** before the final product is completed.

- This helps them understand features better and provide early feedback.
- Developers can test ideas, correct errors, and make improvements quickly.
- Saves time and prevents costly changes at the end of development.
- Example: A **college library system prototype** can be shown to students and librarians for suggestions.
- Prototyping ensures the final software is **closer to user expectations**.

10. Choose the appropriate model for an unstable requirement (Spiral or Agile).

- For **unstable requirements**, the best models are **Agile** and **Spiral**.
- **Agile**: Provides flexibility by delivering in short sprints and adapting to changes.
- **Spiral**: Focuses on iterative cycles with strong risk analysis in each loop.
- Both allow continuous updates and modifications as requirements evolve.
- Agile is best for **fast-changing business apps**, while Spiral is good for **high-risk projects**.
- Outcome: Ensures the software matches user needs even if they change often.

11. Why Spiral is called 'Spiral'?

- The Spiral model is drawn like a **spiral diagram**, where each loop represents a development phase.
- Every cycle includes steps like **planning, risk analysis, development, and evaluation**.
- With each loop, the project grows bigger and more complete, like a spiral expanding outward.
- The spiral shape shows **continuous refinement** and repeated improvement.
- It is mainly focused on **risk analysis at each cycle**, which makes it unique.
- Hence, it is called Spiral because the process expands in a **spiral path**, not a straight line.

M-2 (5 Marks Each)

1. Explore the facilities and drawbacks of waterfall model.

- **Facilities/Advantages**:
  - Simple and easy to understand.
  - Clear structure with well-defined stages.
  - Good for projects with **stable requirements**.
  - Provides proper documentation for future reference.
  - Works well for **short-term, low-risk projects**.
- **Drawbacks/Disadvantages**:
  - Inflexible, changes are difficult once a phase is finished.
  - Errors found in later stages are **expensive to fix**.
  - Customer feedback comes very late.
  - Not suitable for complex or long-term projects.
  - Final product is delivered only at the end, so **no early working model**.

2. Compare iterative and incremental development model.

- **Iterative Model**: Builds software in cycles, where each cycle improves the previous version. Focus is on **refinement**.

- **Incremental Model**: Builds software in small parts, where each increment adds new features. Focus is on **delivery**.
- **Comparison**:
  - Iterative allows improvement of features repeatedly, while Incremental delivers usable parts step by step.
  - Iterative gives feedback on improvement, Incremental gives feedback on usability.
  - Both models are flexible, reduce risks, and are better than the rigid Waterfall model.
- Example: Iterative would keep refining a login system; Incremental would first deliver login, then profile, then chat, etc.

3. Apply the Spiral model in high risk project planning.

- The **Spiral model** is best for high-risk projects because it emphasizes risk analysis at every stage.
- In each cycle, developers identify risks, create alternatives, build prototypes, and evaluate them.
- Example: In a **banking system project**, risks like data security, fraud, and downtime are analyzed in every loop.
- Each iteration improves the system by reducing uncertainty and testing critical functions.
- This ensures that risks are minimized before moving to the next step.
- Thus, Spiral provides **safety, flexibility, and reliability** in projects where failure can be costly.

4. Demonstrate the use of Agile methodology in real-life project development.

- **Agile methodology** is widely used for modern, fast-changing projects.
- Example: In developing a **college mobile app**, Agile delivers features in sprints.
- First sprint can deliver attendance system → second sprint adds notices → third sprint adds online fee payment.
- Each sprint produces a working version that is tested by students and teachers.
- This allows feedback at every stage and continuous improvements.
- Agile ensures **quick delivery, customer satisfaction, and flexibility**, making it perfect for real-world projects.

Module - 3

1. Use an ER Diagram to Design a Hospital System

- Identify main entities: **Patient, Doctor, Nurse, Appointment, Department, Treatment**.
- Define relationships: Patient **books** Appointment, Doctor **handles** Treatment, Department **has** Doctors.
- Specify attributes: Patient(Name, Age, ID), Doctor(Name, Specialization), Appointment(Date, Time).
- Use **primary keys** for each entity (Patient_ID, Doctor_ID).
- Draw the ER diagram showing entities, attributes, and relationships.

2. Demonstrate Logical DFD for an ATM System

- Identify processes: **Authenticate User, Withdraw Cash, Deposit Cash, Check Balance**.
- Data stores: **Customer Account Database, Transaction Records**.
- External entity: **Customer** interacts with the ATM.
- Show logical flow: Customer sends request → Process → Update database → Provide output.
- Use **arrows for data flow** and label clearly.

3. Apply Functional and Non-Functional Requirements to a Banking System

- **Functional:** Transfer funds, open/close account, generate statements.
- **Non-functional:** Security (PIN, encryption), reliability, response time < 3 sec.
- Functional requirements describe **what system does**.
- Non-functional requirements describe **how system performs**.
- Both are documented in **SRS** for clarity.

4. Illustrate the Structure of a Software Requirements Specification (SRS)

- **Introduction:** Purpose, scope, definitions.
- **Overall Description:** System perspective, user needs, constraints.
- **Specific Requirements:** Functional & non-functional requirements.
- **Appendices:** References, glossary.
- Clear, structured SRS helps developers and stakeholders.

5. Formulate a Complete SRS for an Online Exam Portal

- **Introduction:** Purpose – conduct online exams, Scope – students, teachers, admin.
- **Functional Requirements:** Login, take exam, auto-grading, result generation.
- **Non-functional Requirements:** Security, fast response, 99% uptime.
- **Data Requirements:** Student info, exam questions, results.
- **Constraints:** Web-based, supports multiple browsers.

6. Design a Data Dictionary for a Student Management System

- Define entities: **Student, Course, Enrollment, Teacher**.
- Attributes: Student(ID, Name, Age), Course(Code, Name, Credits).
- Data type and size for each attribute (e.g., Name: String, 50 chars).
- Primary key for each entity (Student_ID, Course_Code).
- Relationships between entities documented clearly.

7. ER Diagram for a Library Management System

- Entities: **Book, Member, Issue, Author**.
- Relationships: Member **borrows** Book, Book **written by** Author, Issue **links** Member & Book.
- Attributes: Book(ID, Title, ISBN), Member(ID, Name), Issue(Date, ReturnDate).
- Primary keys: Book_ID, Member_ID, Issue_ID.

- Diagram shows entities, attributes, and relationships.

8. Structured Interviewing Technique for Requirement Elicitation

- Prepare **list of questions** covering all system aspects.
- Involve **all stakeholders**: users, managers, IT staff.
- Use **open-ended and scenario-based questions**.
- Document answers and clarify ambiguities.
- Review findings with stakeholders for confirmation.

9. Framework to Assess Technical Feasibility

- Check if **hardware, software, and network** resources are sufficient.
- Evaluate team skills for development.
- Assess technology compatibility with existing systems.
- Identify risks and possible mitigation strategies.
- Provide recommendation: feasible, partially feasible, or not feasible.

10. Data Dictionary Template for Inventory System

- **Entity:** Product, Supplier, Stock.
- **Attributes:** Product(ID, Name, Price, Quantity), Supplier(ID, Name).
- **Data type & size:** Name – String(50), Price – Decimal(8,2).
- **Primary key:** Product_ID, Supplier_ID.
- **Description:** Brief explanation for each attribute.

11. Method to Validate Software Requirements

- Conduct **stakeholder walkthroughs**: review each requirement with users.
- Use **checklists** to ensure all functional and non-functional points are covered.
- Identify inconsistencies, missing requirements, or ambiguities.
- Document corrections and get stakeholder approval.
- Repeat process until all requirements are validated.

12. Strategy to Gather Functional & Non-Functional Requirements for E-commerce

- Identify **stakeholders**: customers, admin, delivery team.
- Use **interviews, questionnaires, and observation** to collect requirements.
- Separate **functional requirements** (checkout, payment, product search).
- Separate **non-functional requirements** (security, performance, uptime).
- Validate requirements with stakeholders before documentation.

Basics of Requirements

1. Define Software Requirement Analysis

- Software Requirement Analysis is the process of **understanding, gathering, and defining the needs** of users and stakeholders.
- It involves **studying the system requirements** and documenting them clearly.
- Helps identify **functional, non-functional, and domain requirements**.
- Ensures developers understand what to build.
- Reduces errors, misunderstandings, and project risks.

2. Differentiate System and Software Requirements

Aspect	System Requirement	Software Requirement
Scope	Entire system including hardware and software	Only software features and behavior
Examples	Network bandwidth, processor speed	Login module, report generation
Dependency	Hardware and software	Only software
Focus	Overall system performance	Software functionality
Audience	System engineers, IT managers	Software developers, testers

3. Define Functional Requirements with Examples

- Functional requirements describe **what the system must do**.
- They focus on **tasks, features, and services** the system provides.
- Examples: User login, fund transfer, online purchase, report generation.
- They form the basis for **system design and testing**.
- Functional requirements must be **clear, complete, and verifiable**.

4. Define Non-Functional Requirements with Examples

- Non-functional requirements describe **how the system performs tasks**.
- They focus on **quality attributes** like reliability, usability, and performance.
- Examples: System response time < 2 sec, 99% uptime, secure login.
- Ensure **user satisfaction and system efficiency**.
- Important for system scalability, maintainability, and performance.

5. What are Domain Requirements?

- Domain requirements are **specific to the business or industry**.

- Derived from rules, standards, or regulations in a particular domain.
- Example: Banking regulations, healthcare privacy rules.
- Ensures system complies with **domain standards**.
- Helps reduce errors and legal issues.

6. What are User Requirements?

- User requirements describe **what the users expect from the system**.
- Written in **simple, non-technical language**.
- Example: “Students can view grades online.”
- Focuses on **user goals and interactions** with the system.
- Serves as a foundation for detailed system specifications.

Requirement Elicitation and Analysis

7. What is Requirement Elicitation?

- Requirement elicitation is the process of **collecting and understanding requirements from stakeholders**.
- Involves techniques like interviews, questionnaires, and observation.
- Helps uncover **explicit and implicit user needs**.
- Prevents miscommunication between developers and users.
- Forms the basis for **requirement analysis and SRS creation**.

8. List Common Techniques of Requirement Elicitation

- **Interviews:** Direct discussion with stakeholders.
- **Questionnaires/Surveys:** Collect feedback from multiple users.
- **Observation:** Study how users work with the current system.
- **Workshops/Brainstorming:** Collaborative requirement gathering.
- **Prototyping:** Build simple models to clarify requirements.

9. What are Use-Cases in Software Engineering?

- Use-cases describe **interactions between users and the system**.
- Show step-by-step **functional behavior for specific scenarios**.
- Helps capture **requirements clearly and completely**.
- Example: ATM withdrawal use-case: Authenticate → Select amount → Dispense cash.
- Useful for testing and validating functional requirements.

10. Define Scenarios in Requirement Analysis

- Scenarios are **specific situations describing how the system behaves**.
- Helps understand real-world usage of the system.
- Example: Insufficient balance during ATM withdrawal.
- Scenarios support **use-case development and requirement clarity**.
- Used for validating and refining requirements.

11. Explain the Role of Interviews in Requirement Elicitation

- Interviews collect **detailed requirements directly from stakeholders**.
- Clarifies expectations, priorities, and constraints.
- Helps detect **hidden or implicit requirements**.
- Can be structured or unstructured depending on needs.
- Supports collaboration between developers and users for accurate SRS.

Modeling Techniques

12. What is a Data Flow Diagram (DFD)?

- DFD is a **graphical representation of data movement** in a system.
- Shows processes, data stores, input/output, and external entities.
- Helps understand **system functionality and information flow**.
- Used during **requirement analysis and system design**.
- Simplifies complex processes into **visual diagrams** for clarity.

13. Differentiate Physical DFD and Logical DFD

Aspect	Physical DFD	Logical DFD
Focus	How system operates physically	What the system does logically
Components	Hardware, files, people	Processes, data flows, data stores
Level of Detail	Implementation-specific	Abstract, requirement-focused
Purpose	Shows real implementation	Shows functional requirement
Audience	System engineers, developers	Analysts, stakeholders

14. What is an Entity-Relationship Diagram (ERD)?

- ERD shows **entities, attributes, and relationships** in a system.
- Entities: objects or concepts (e.g., Student, Course).
- Relationships: interactions between entities (e.g., Enrolls).
- Used for **database modeling and requirement analysis**.
- Clarifies **data structure and system design**.

15. Define a Data Dictionary

- A data dictionary is a **central repository of all data definitions** in the system.
- Includes entities, attributes, data types, sizes, and relationships.
- Ensures consistency in **database design and documentation**.
- Helps developers, testers, and stakeholders understand data requirements.
- Supports requirement validation and system maintenance.

Requirement Validation and Specification

16. What is Requirement Validation?

- Requirement validation ensures **requirements are correct, complete, and feasible**.
- Detects ambiguities, conflicts, and missing requirements.
- Involves **stakeholders to verify requirements**.
- Prevents errors in development and testing.
- Ensures system meets user and business needs.

17. List Common Requirement Validation Techniques

- Reviews and walkthroughs with stakeholders.
- Prototyping and simulation for validation.
- Peer inspections and document reviews.
- Checklist-based verification.
- Test case analysis to ensure requirements are testable.

18. Define Requirement Specification

- Requirement specification is a **formal description of system requirements**.
- Documents functional, non-functional, and domain needs.
- Provides a reference for developers and testers.
- Ensures all stakeholders agree on requirements.
- Basis for creating SRS and system design.

19. Differentiate Requirement Specification and Design Specification

Aspect	Requirement Specification	Design Specification
Purpose	What the system must do	How the system will implement it
Focus	Functional and non-functional needs	Architecture, modules, algorithms
Audience	Users and stakeholders	Developers and designers
Detail Level	Abstract and high-level	Detailed and implementation-focused
Timing	Early in SDLC	After requirement specification

Software Requirement Specification (SRS)

20. Define SRS (Software Requirement Specification)

- SRS is a **formal document that describes all software requirements**.
- Includes functional, non-functional, and domain requirements.
- Serves as a **contract between stakeholders and developers**.
- Helps in planning, designing, and testing the system.
- Provides a reference for maintenance and updates.

21. List Main Sections of an SRS Document

- **Introduction:** Purpose, scope, definitions.
- **Overall Description:** System perspective, user characteristics, constraints.
- **Specific Requirements:** Functional, non-functional, and interface requirements.
- **Appendices:** References, glossary, supporting info.
- **Index/Table of Contents** for easy navigation.

22. What is Included in Functional Requirements in SRS

- Actions or services the system must provide.
- Example: User login, data entry, report generation.
- Defines system behavior in response to inputs.
- Used for designing and testing system functions.
- Should be **clear, complete, and verifiable**.

23. What is Included in Non-Functional Requirements in SRS

- Quality attributes of the system.
- Examples: Performance, reliability, security, usability.
- Defines **how the system performs** tasks.
- Helps in system optimization and user satisfaction.
- Critical for scalability and maintainability.

24. Purpose of Constraints and Assumptions in SRS

- **Constraints:** Limitations on system design (platform, tools, budget).
- **Assumptions:** Conditions assumed true during development (e.g., network availability).
- Guide development and testing strategies.
- Manage expectations of stakeholders.
- Reduce risks and ensure feasibility.

Feasibility

25. What is Feasibility Study in Software Engineering

- Feasibility study assesses **whether a project is practical and achievable**.
- Considers technical, operational, economic, and legal aspects.
- Helps decide whether to proceed with the project.
- Identifies potential risks, costs, and benefits.
- Prevents wasting resources on impractical solutions.

26. List Types of Feasibility

- **Technical:** Availability of technology, tools, and expertise.
- **Operational:** System usability and stakeholder acceptance.
- **Economic:** Cost-effectiveness and ROI analysis.
- **Legal:** Compliance with laws, regulations, and contracts.
- **Schedule Feasibility:** Can the project be completed on time.

27. Why is Feasibility Analysis Important

- Prevents investment in impractical projects.
- Identifies risks and constraints early.
- Helps in decision-making and project planning.
- Ensures proper allocation of resources.
- Increases chances of project success and reduces failures.