1. SDLC (Software Development Life Cycle)

Definition: SDLC is a systematic process that involves a series of phases to design, develop, test, and deploy software applications.

Phases of SDLC:

1. **Planning and Brainstorming:** Gather the team, set goals, and identify risks.
2. **Requirement Gathering and Analysis:**
   - Identify and document the user's needs and expectations.
   - Analyze the requirements to ensure feasibility and clarity.
3. **System Design:**
   - Create a high-level design (architectural design) to define the overall structure.
   - Develop detailed design specifications for modules and components.
4. **Implementation/Coding:**
   - Translate the design into actual code using programming languages.
   - Write clean, efficient, and well-documented code.
5. **Testing:**
   - Unit testing: Test individual units of code.
   - Integration testing: Test how different modules interact.
   - System testing: Test the entire system as a whole.
   - Acceptance testing: Verify the system meets user requirements.
6. **Deployment:**
   - Deploy the software to the production environment.
   - Configure the software for optimal performance.
7. **Maintenance:**
   - Fix bugs, add new features, and make improvements.
   - Provide ongoing support and maintenance.

Benefits of SDLC:

- **Improved Quality:** Systematic approach ensures high-quality software.
- **Reduced Costs:** Efficient planning and execution minimize costs.
- **Risk Management:** Identifies and mitigates risks early on.
- **Better Project Management:** Clear phases and milestones for better control.
- **Customer Satisfaction:** Meets user requirements and expectations.

Different SDLC Models:

- **Waterfall Model:** A linear, sequential approach.
- **Agile Model:** Iterative and incremental development.
- **Spiral Model:** Combines iterative development with risk analysis.
- **V-Model:** A variation of the waterfall model with a verification and validation phase for each development phase.

2. SRS (Software Requirements Specification) Document

An SRS document is a formal document that outlines the functional and non-functional requirements of a software system. It serves as a blueprint for the development team, ensuring

that the final product meets the user's needs.

Importance of SRS:

- **Clear Communication:** Provides a common understanding between stakeholders.
- **Requirements Traceability:** Tracks requirements throughout the development process.
- **Risk Mitigation:** Identifies potential risks and challenges early on.
- **Quality Assurance:** Ensures the software meets specified requirements.
- **Project Management:** Helps in planning, scheduling, and resource allocation.

Why We Need It:

- **Consistent Understanding:** Ensures everyone involved in the project has a shared understanding of the requirements.
- **Basis for Testing:** Provides a basis for creating test cases to verify the software's functionality.
- **Change Management:** Facilitates tracking changes and their impact on the project.
- **Legal Documentation:** Can be used as a legal reference in case of disputes.

3. Difference between Unit Testing, Integration Testing, and Acceptance Testing:

Feature	Unit Testing	Integration Testing	Acceptance Testing
Focus	Individual units of code	Interaction between modules	Entire system
Scope	Smallest testable parts	How modules work together	System's overall behavior
Testing Level	Developer	Developer/Tester	User/Customer
Goal	Verify unit functionality	Ensure modules work together	Verify system meets requirements

4. Classical Waterfall Model

The Waterfall Model is a linear, sequential approach to software development. It consists of the following phases:

1. **Requirement Gathering and Analysis**

2. **System Design**
3. **Implementation**
4. **Testing**
5. **Deployment**
6. **Maintenance**

Each phase is completed before moving on to the next, making it a rigid and inflexible model.

5. Project Planning

Project planning involves defining the scope, goals, and tasks of a software project. It includes:

- **Defining project objectives:** Clearly stating the project's purpose and goals.
- **Identifying tasks and dependencies:** Breaking down the project into smaller tasks and determining their relationships.
- **Estimating time and resources:** Allocating resources and timeframes for each task.
- **Creating a project schedule:** Developing a timeline for the project, including milestones and deadlines.
- **Risk management:** Identifying potential risks and developing mitigation strategies.
- **Communication plan:** Establishing communication channels and protocols.

6. Risks Associated with Software Development

- **Technical Risks:** Issues related to technology choices, compatibility, and performance.
- **Resource Risks:** Insufficient resources (people, budget, hardware, software).
- **Schedule Risks:** Delays in development or testing phases.
- **Quality Risks:** Defects, bugs, and security vulnerabilities.
- **Organizational Risks:** Lack of support, changes in priorities, or organizational restructuring.
- **Third-Party Risks:** Reliance on external vendors or services.