# Life Cycle Phases in SDLC & Algorithms

## Life Cycle Phases in SDLC

### Planning

Purpose: Define project goals, scope, resources, and feasibility.

Activities:

- Identify business requirements.

- Create a project plan and timeline.

- Conduct a feasibility study (technical, economic, legal).

### Requirement Analysis

Purpose: Understand what the end user expects from the system.

Activities:

- Gather and document functional and non-functional requirements.

- Create a Software Requirements Specification (SRS) document.

### Design

Purpose: Develop the architecture and detailed design of the system.

Activities:

- High-level design (HLD): Defines overall system architecture.

- Low-level design (LLD): Detailed design of individual modules/components.

- Create wireframes, flowcharts, and prototypes.

### Implementation (Coding)

Purpose: Build the actual software by writing code.

Activities:

- Develop source code based on the design.

- Use coding standards and version control.

- Perform unit testing during development.

### Testing

Purpose: Ensure the software works as expected and is bug-free.

Activities:

- Perform various testing types: unit, integration, system, and acceptance testing.

- Report, fix, and re-test defects.

### Deployment

Purpose: Deliver the software to users or deploy it in a production environment.

Activities:

- Deploy the software on a live server or environment.

- Conduct user acceptance testing (UAT).

- Provide user training and support documentation.

### Maintenance

Purpose: Keep the software updated and running smoothly.

Activities:

- Monitor for bugs and performance issues.

- Provide regular updates and patches.

- Handle enhancements or feature additions.

## Algorithms Life Cycle Phases

Problem Definition: Clearly define the problem to be solved and its constraints.

Analysis: Analyze the inputs, expected outputs, and the relationship between them. Determine the problem's complexity and scope.

Algorithm Design: Develop a step-by-step procedure to solve the problem. Choose an approach (e.g., greedy, divide-and-conquer, dynamic programming).

Algorithm Representation: Represent the algorithm using pseudocode or flowcharts for clarity.

Implementation: Translate the algorithm into a programming language.

Testing and Debugging: Verify the algorithm for correctness using test cases. Debug errors and optimize for performance.

Analysis of Efficiency: Evaluate the algorithm’s time complexity (speed) and space complexity (memory usage).

Documentation and Refinement: Document the algorithm for future reference. Refine and optimize the algorithm if necessary.