# Design and Prototyping in SDLC + Algorithms

## Design and Prototyping in SDLC

### 1. Design in SDLC

The design phase focuses on planning and structuring the software system. It provides a detailed blueprint that outlines how the software will meet the specified requirements.

#### Objectives:

- Define the architecture of the system.

- Identify components and their interactions.

- Ensure scalability, reliability, and maintainability.

#### Key Activities:

- High-Level Design (HLD):

- Defines the overall architecture, including modules, data flow, and external interfaces.

- Often represented with diagrams like system architecture diagrams and context diagrams.

- Low-Level Design (LLD):

- Focuses on detailed specifications of individual components or modules.

- Includes algorithms, data structures, and logic flows.

#### Outputs:

- Design documents, including HLDs and LLDs.

- Diagrams like UML (Unified Modeling Language) diagrams: class diagrams, sequence diagrams, etc.

### 2. Prototyping in SDLC

Prototyping involves creating a preliminary version of the software to validate requirements, design choices, and usability before full-scale development begins.

#### Types of Prototypes:

- Throwaway Prototype: Used to quickly validate ideas, then discarded.

- Evolutionary Prototype: Gradually refined into the final product.

- Low-Fidelity Prototype: Simple, often non-interactive mockups focusing on functionality.

- High-Fidelity Prototype: Detailed, interactive, and closely resembles the final product.

#### Objectives:

- Validate requirements and user needs.

- Reduce risks by identifying issues early.

- Gather feedback from stakeholders.

#### Tools and Techniques:

- Tools: Figma, Adobe XD, Sketch for UI; programming languages for functional prototypes.

- Techniques: Wireframing, mockups, and iterative design.

## Algorithms in Design

### 1. Role of Algorithms in Design:

- Define logic for specific functionalities.

- Optimize processes for efficiency.

- Solve computational or data-related problems.

### 2. Algorithm Design Techniques:

- Divide and Conquer:

- Breaks down a problem into smaller sub-problems, solves them independently, and combines their solutions (e.g., merge sort, quick sort).

- Dynamic Programming:

- Solves problems by breaking them into overlapping sub-problems, storing results to avoid redundant computations (e.g., Fibonacci sequence, knapsack problem).

- Greedy Algorithms:

- Makes the locally optimal choice at each step to achieve a global solution (e.g., Dijkstra’s algorithm for shortest paths).

- Backtracking:

- Explores all possibilities by building solutions incrementally and abandoning paths that fail to satisfy constraints (e.g., solving a maze, N-Queens problem).

### 3. Algorithm Prototyping:

- Algorithms are often tested in prototype systems to verify their correctness and performance.

- Tools like Python, MATLAB, or pseudocode can be used for quick implementation and testing.

## Connection Between Design, Prototyping, and Algorithms

- Design lays the foundation for algorithm selection and application.

- Prototyping allows early testing of algorithms in a simulated environment to ensure they meet performance and functionality requirements.

- Together, they ensure the software's logic and user experience are optimized before development begins.