**SDLC** stands for Software Development Life Cycle.

**Phases of SDLC**

1. **Requirement Analysis**
   * Gather requirements from stakeholders (clients, users, business analysts).
   * Define *what* the software should do.
   * Deliverable: *Software Requirement Specification (SRS)*.
2. **System Design**
   * Create architecture and design based on requirements.
   * Includes database design, UI design, and system diagrams.
   * Deliverable: *Design Document*.
3. **Implementation (Coding/Development)**
   * Developers write code according to design specifications.
   * Follow coding standards and guidelines.
   * Deliverable: *Source Code*.
4. **Testing**
   * Verify and validate the software.
   * Types: unit testing, integration testing, system testing, user acceptance testing (UAT).
   * Deliverable: *Test Reports*.
5. **Deployment**
   * Release the software to the user environment (production).
   * May be rolled out in phases or full release.
   * Deliverable: *Deployed Application*.
6. **Maintenance**
   * Fix bugs, update features, and handle performance issues.
   * Deliverable: *Patch Updates / Maintenance Reports*.

### ****Popular SDLC Models****

* **Waterfall Model** – A linear, sequential process where each phase must finish before the next begins. Best for small projects with well-defined requirements.
* **Agile Model** – Iterative and incremental approach with short sprints, allowing rapid feedback and flexibility. Ideal for dynamic, evolving projects.
* **Spiral Model** – Risk-driven model that combines iterative development with strong risk analysis. Suitable for large, high-risk, mission-critical projects.
* **V-Model** – Extension of Waterfall where each development phase has a corresponding testing phase. Ensures early defect detection and high reliability.
* **Big Bang Model** – Minimal planning; development starts immediately with coding. Works for small projects or proof-of-concepts but very risky for large ones.

## 🏗 **Monolithic Architecture**

* **Definition**: Entire application is built as a single, tightly coupled unit.
* **How it works**: All features (UI, business logic, database operations) live in the same codebase and are deployed together as one package (e.g., .war, .jar, or single executable).

## 🔗 **Microservices Architecture**

* **Definition**: Application is divided into multiple loosely coupled, independently deployable services.
* **How it works**: Each microservice is responsible for one specific business function (e.g., user service, order service, payment service) and communicates via APIs (REST, gRPC, or messaging queues like Kafka/RabbitMQ).

| **Aspect** | **Monolithic** | **Microservices** |
| --- | --- | --- |
| **Deployment** | One big package | Each service deployed independently |
| **Scalability** | Scale whole app | Scale individual services |
| **Technology** | Same tech stack everywhere | Different stacks for different services |
| **Fault Impact** | One bug may crash whole system | Isolated — only one service fails |
| **Best For** | Small apps, startups, MVPs | Large, complex, evolving systems |

DevOps = Development + Operations.

 **Automate** the software delivery process.

 **Improve collaboration** between developers and operations.

 **Deliver software faster, more reliably, and with higher quality.**

**Automation** means making tasks happen **automatically by software/tools** instead of doing them **manually** by people.