SDLC Phases Explained

fully explain these 7 phases of software development life cycle:

1. planning and nalysis:
-deadline goal
-stake holders
-priority framework
-documentation
-budget planning

2. Feasibility analysis:
-technical assesment
-cost examination
-reach analysis
-requirement translation(important)
- use case development
-Documentation

3.System design:
-blueprint creation
-architecture design
-UI design
-Visualization
-Modular planning(important)
4.Development
-code with best practices
-unit testing
5. Testing
6.Deployment
7.Maintainance

Here's a **fully detailed explanation** of the **7 phases of the Software Development Life Cycle (SDLC)** based on the points you've listed under each phase:

---

1. Planning and Analysis

This is the **foundation stage** of the SDLC, where the goal is to **understand the problem, define scope, set objectives**, and plan resources and time effectively.

Key Elements:

- **Deadline Goal:**
  Time estimation is critical. A realistic deadline is set based on complexity, available resources, and expected outcomes. Milestones may also be defined.

- **Stakeholders:**
  These include clients, end-users, developers, testers, managers, and sponsors. Each group provides input, expectations, and constraints for the project.
- **Priority Framework:**
  Features and tasks are ranked by urgency and importance using techniques like MoSCoW (Must have, Should have, Could have, Won't have) or risk/value analysis.
- **Documentation:**
  A **Software Requirement Specification (SRS)** document is prepared that outlines all functional and non-functional requirements, goals, constraints, and expectations.
- **Budget Planning:**
  Financial estimates are made for development tools, salaries, testing, deployment, and maintenance. Helps ensure project stays economically viable.

---

2. Feasibility Analysis

Before building the system, it is necessary to check **whether it is possible and worthwhile** to develop the software.

Key Elements:

- **Technical Assessment:**
  Determines if the team has the required **skills, tools, technology, and infrastructure** to develop the software.
- **Cost Examination:**
  Calculates estimated development, testing, deployment, and maintenance costs. Return on Investment (ROI) is also evaluated.
- **Reach Analysis:**
  Determines the **impact and scalability** of the software. Who will use it? How many users can it support? Will it support future expansion?
- **Requirement Translation (Important):**
  Converts **business-level language into technical requirements**. For example, "the user should get instant feedback" becomes "the system should respond within 500ms".
- **Use Case Development:**
  Specific user actions are mapped out in **use cases**, which describe how users will interact with the system. Helps in understanding user flow.
- **Documentation:**
  The findings are documented in a **Feasibility Study Report**, which is reviewed by decision-makers before giving the green light.

---

3. System Design

This phase focuses on **how the software will be built**, based on the requirements and feasibility findings.

Key Elements:

- **Blueprint Creation:**
  A high-level **design blueprint** is made that describes system structure, data flow, modules, and interactions.
- **Architecture Design:**
  Focuses on choosing the right **design architecture** (e.g., MVC, Microservices, Layered Architecture), defining components and how they interact.
- **UI Design:**
  User Interface mockups and prototypes are created to visualize what the application will look and feel like. Includes layout, colors, fonts, and responsiveness.
- **Visualization:**
  Includes wireframes, ER diagrams, class diagrams, sequence diagrams, and flowcharts to visualize the logic, data, and process.
- **Modular Planning (Important):**
  The system is **broken into manageable, independent modules** or components. Each module is designed to handle a specific task (like authentication, payment, reporting). Helps in scalability, parallel development, and debugging.

---

4. Development

This is the phase where **actual coding** happens based on the design document.

Key Elements:

- **Code with Best Practices:**
  Developers follow **coding standards, naming conventions, SOLID principles, DRY (Don't Repeat Yourself), KISS (Keep it Simple), and proper documentation**.
- **Unit Testing:**
  Developers write unit tests to **verify that individual units/modules** work as expected. Often automated, using tools like JUnit (Java), PyTest (Python), etc.

---

5. Testing

The system is tested thoroughly before deployment to ensure **quality, performance, and reliability**.

Key Types of Testing:

- **Integration Testing:**
  Checks how different modules work together.
- **System Testing:**
  Verifies the system as a whole.
- **Acceptance Testing:**
  Ensures software meets business requirements (UAT – User Acceptance Testing).

- **Performance Testing:**
  Tests responsiveness, scalability, and stability under load.
- **Security Testing:**
  Identifies vulnerabilities.
- **Bug Reporting:**
  Testers document defects in a bug-tracking system, and developers fix them in iterations.

---

6. Deployment

This is the process of **making the software live** and available to users.

Deployment Stages:

- **Staging Environment:**
  First deployed to a staging server that mimics production, for final round of testing.
- **Production Deployment:**
  Application is pushed to live servers and users start using the system.
- **Deployment Strategies:**
  Can include **phased rollout, blue-green deployment, or canary releases** to minimize risks.
- **Post-deployment Monitoring:**
  Monitoring tools are used to check server health, errors, performance, etc.

---

7. Maintenance

After deployment, the system enters a **maintenance phase** where it is updated, optimized, and fixed based on user feedback or issues.

Maintenance Types:

- **Corrective Maintenance:**
  Fixing bugs that weren't discovered during earlier testing.
- **Adaptive Maintenance:**
  Making the software work in a new or changed environment (e.g., new OS or hardware).
- **Perfective Maintenance:**
  Enhancing performance or adding new features based on user demand.
- **Preventive Maintenance:**
  Refactoring code, cleaning deprecated functions, improving security before problems occur.

Summary Diagram (Text Format):

```
markdown

1. Planning & Analysis
   ↓
```

```
2. Feasibility Analysis
   ↓
3. System Design
   ↓
4. Development
   ↓
5. Testing
   ↓
6. Deployment
   ↓
7. Maintenance
```

Would you like a diagram/infographic or a printable PDF of this explanation?