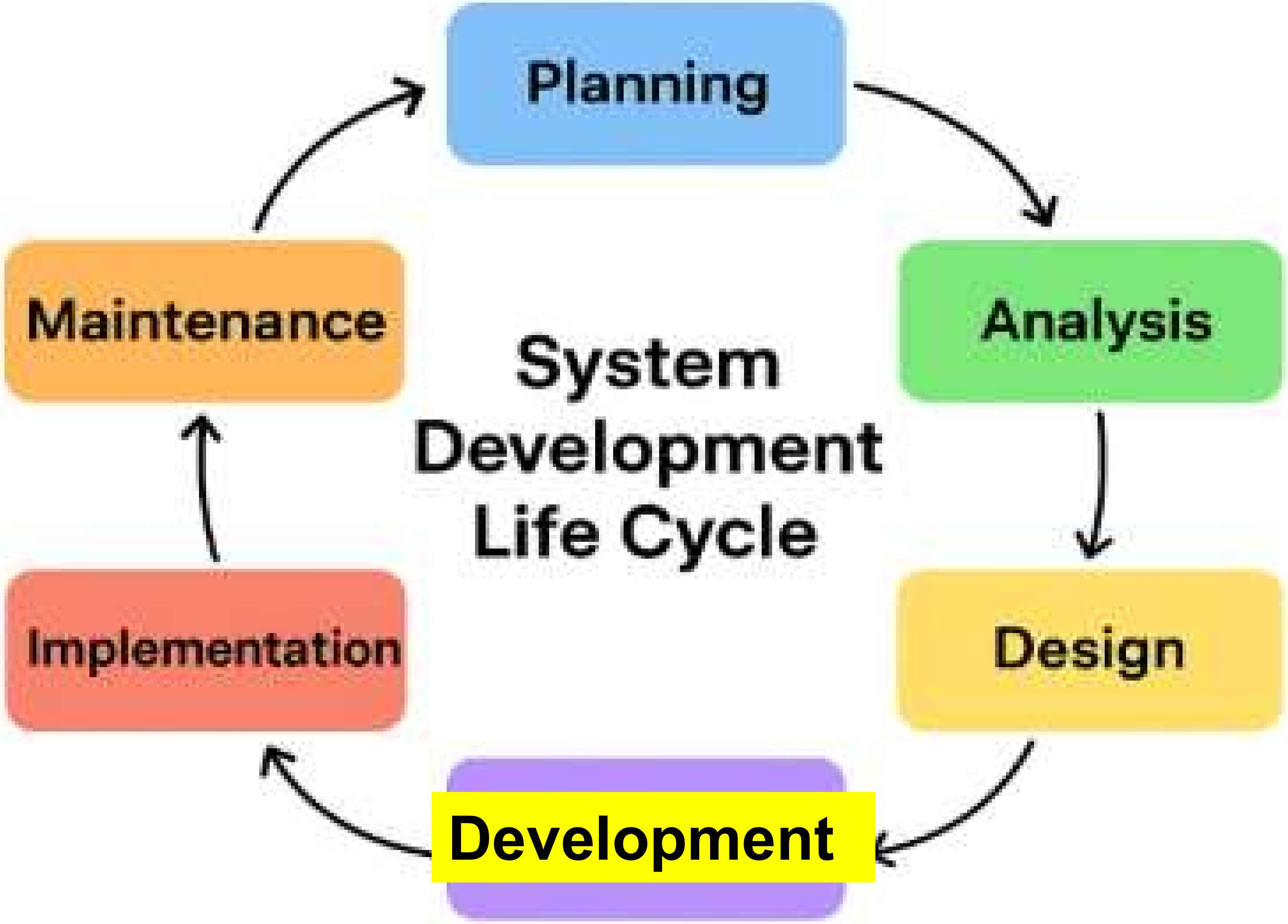
Chapter 3_1

Systems Development Life Cycle (SDLC)

- SAD typically follows the Systems Development Life Cycle (SDLC),
- The Systems Development Life Cycle (SDLC) is a structured framework used to develop, implement, and maintain information systems.
- Systems are built systematically to meet organizational and user needs.

- **Below is an introduction to the sex key SDLC phases in orders:**

1. **Planning**
2. **Analysis**
3. **Design**
4. **Development**
5. **Implementation and**
6. **Maintenance**


Planning
Analysis
Design
Development
Implementation
Maintenance
System Development Life Cycle

1. Planning Phase

In the Planning phase, the main goal is to prepare everything needed before starting the actual development:

Purpose:

- To define the project at a high level and decide whether it is worth يستحق starting.
- Identify the problem or need the new system should solve
- What Happens in the Planning Phase..as:

Define project scope - Conduct feasibility studies including - Create a project plan including

a) Define project scope

1. Set objectives (what the system must achieve)
2. Defining boundaries of the project (what the project will include and not include),
3. What people, tools, or materials are required.

b) Conduct feasibility studies including

- Technical feasibility – do we have the technology?
- Economic feasibility – is it cost-effective?
- Operational feasibility – will users accept it?
- Schedule feasibility – can it be done on time?

c) Create a project plan including:

- Timeline
- Budget
- Resources (people, tools, technologies)
  - ✓ Showing when each task be done,
  - ✓ Who will do it,
  - ✓ Estimate costs for all elements (labor, materials, overheads) and
  - ✓ Allocate a budget.

In short

- **Planning phase sets the foundation for the whole project to ensure it’s realistic, organized, and achievable.**
- **If the planning is wrong, the rest of the project will fail. This phase ensures the project is realistic, justified, and properly defined.**

Short Summary

Phase	Purpose	Key Activities
Planning	Decide if the project is viable and define its scope	Feasibility study, objectives, project plan

2. Analysis Phase

Purpose

- To deeply understand the requirements of the system and document what the system must do.
- What Happens in the Analysis Phase?.......
  - – Gather detailed requirements through
  - – Analyze current (existing) systems
  - – Model processes

a) Gather detailed requirements through

1. Stakeholder collaboration
2. Engage with stakeholders (e.g., end-users, managers, IT teams) via interviews, workshops, surveys, and
3. Focus groups to collect comprehensive, accurate, and
4. Prioritized requirements.

b) Analyze current (existing) systems

1. Strengths, weaknesses
2. Identify functional requirements

   What features the system must provide

3. Identify non-functional requirements

   Performance, security, reliability, usability

c) Model processes and data to clarify system needs: Use diagrams and models such as:

- Data Flow Diagrams (DFDs)
- Entity–Relationship Diagrams (ERDs)
- Use case diagrams

Why It Matters:

If you don’t understand the requirements, you will build the wrong system. The analysis phase ensures designers and programmers know exactly what to build.

Short Summary

Phase	Purpose	Key Activities
Analysis	Understand and document requirements	Requirement gathering, modeling, requirement specification

3. Design Phase (Security-focused)

Design Phase (Security-focused)

- “How the system will work”
- It focuses on
  - ✓ Defining the system architecture,
  - ✓ Data flows,
  - ✓ Interface layouts,
  - ✓ Data structures, and
  - ✓ Security controls.

- **It translates the requirements gathered in the Analysis phase into a blueprint for building the system.**

Key points:

1. Converts what the system must do into, how it will do it.
2. Produce system models such as ER diagrams, data flow diagrams, interface, and system architecture.
3. Defines hardware and software requirements.

4. **Ensures the system design aligns with user needs, performance, and security requirements.**
5. **Create detailed specifications for system components architecture:**

   ✓**Interfaces**,

   ✓**Databases**.

Interfaces

- **APIs**

Application Programming Interfaces,

- **UI**

User Interface wireframes,

API:

is a set of rules and tools that allows different software applications to communicate with each other.

UI is the part of a software, app, or website that a user sees and interacts with. It includes everything you click, type, or view like buttons, menus, text, images, and layouts.

Service Contracts

- Called a service agreement, professional services agreement, or consulting agreement)
- Legal agreements that define how a service will be provided, by whom, to whom, for how much, and under what terms.

- Legally binding between two parties:
    1. The service provider (the person or company that will perform the work)
    2. The client/customer (the person or company that will receive and pay for the work)

Database design

- Schema with tables, relationships, indexes, normalization.
- Use standards like UML (Unified Modeling Language), or sequence diagrams for clarity. In short: UML the standard way to draw blueprints of software systems.

Most common UML diagram types:

Class Diagram – shows classes, attributes, methods, and relationships (most used).

Use Case Diagram – shows actors and system functionality.

Sequence Diagram – shows interaction between objects over time.

Activity Diagram – flowchart-like, shows workflow/business processes.

State Machine Diagram – shows states and transitions of an object.

Component Diagram – shows physical structure of code components.

Deployment Diagram – shows hardware and software deployment.

6. **Develop prototypes and get stakeholder feedback. Build**

- ✓ **low-fidelity (sketches) or**
- ✓ **high-fidelity (interactive models using tools prototypes.**
- ✓ **Conduct reviews,**
- ✓ **usability testing, and**

- ✓ **feedback sessions to validate design assumptions and refine usability before development.**

7. Plan for

- Security,
- Scalability, and
- Performance.

Security: Define

Authentication, authorization, and encryption = Login = "Prove it's you"

Authorization = Permissions = "What can you access?“

Encryption = Locks data with a key

***Scalability*:**

Design for horizontal/vertical scaling, load balancing, caching, and auto-scaling policies.

Load Balancing = Distributing incoming traffic across multiple servers to ensure no single server gets overwhelmed.

Auto-Scaling = The system automatically adds or removes servers

***Performance*:**

- **Set response time targets,**
- **Optimize queries,**
- **Plan for CDNs (Content Delivery Network) and**
- **Monitoring (logging).**

8. Key security activities:

- ✓ Threat Modeling:
- ✓ Security Requirements Definition
- ✓ Secure Architecture Design:

Threat Modeling:

- Threat modeling is a proactive security practice used during system design to identify, analyze, and prioritize potential threats before writing a single line of code.

- It helps developers and analysts understand how an attacker might compromise the system and what security controls should be added to prevent these attacks.

Security Requirements Definition:

- translating an organization’s business goals and security needs into specific, (e.g., authentication method, encryption standards, input validation rules).
- This ensures that security is built into the system from the beginning rather than added later.

- it includes:

✓ Authentication requirements

Define how users prove their identity, such as Username + password

✓ Encryption standards

Specify how data must be protected

✓ Input validation rules

Define how the system must handle and sanitize input to prevent attacks,

Why it's important:

- Ensures the system complies with legal, organizational, and industry security standards.
- Provides developers with clear guidelines.
- Prevents vulnerabilities early in the SDLC.

In summary:

Security Requirements Definition converts security concerns into specific rules that guide system design and implementation, covering areas like authentication, encryption, and validation to ensure the system is secure by design.

Goal: design Phase Create a clear plan that guides developers during implementation

4. Development Phase

Implementation/Coding

(Building the system)

The actual system is built in this phase.

1. Programmers write code according to the design specifications.
2. Components/Modules are developed and
3. Unit testing often begins here.
4. Code integrated.
5. Code Reviews
6. Bug Fixing

Purpose:

- Con)vert the detailed design documents into actual (executable code) and
- Build a working software system).
- Programmers finally write the real production code

Key Activities in the Development Phase

1. **Writing Code**

   – Developers write clean,

   – Maintainable, and

   – Efficient code based on

     ✓ High-Level Design (HLD) and

     ✓ Low-Level Design (LLD).

   – Follow coding standards, naming conventions, and best practices.

HLD (High-Level Design)

This provides an architectural blueprint of the system:

- **Major modules,**
- **Data flow,**
- **Technologies,**
- **Integration points, and**
- **Overall structure.**

LLD (Low-Level Design)

This zooms in

- Specifying detailed logic,
- Class diagrams,
- Database schema,
- Algorithms, and
- How each component behaves internally.

2. Module/Component Development

- Refers to the stage in software development where individual parts of the system—called *modules* or *components* are built independently.
- Each module usually represents a specific feature or functionality of the system.

During this phase: The output is a set of functional components that will later be integrated to form the complete application.

3. Unit Testing

- Each small piece of code (function, class, or module) is tested immediately by the developer.

Goal:

Catch bugs as early and as close to the source as possible.

4. Code Integration

- Developed modules are combined (integrated) into larger parts of the system.
- Use Continuous Integration (CI) tools to automate building and testing on every code commit.

5. Code Reviews

Code reviews (also called peer reviews) are a systematic examination of source code by one or more developers who did not write the code.

goal

Improve the overall quality of the software before it is merged into the main codebase.

6. Bug Fixing (at unit/integration level)

Fixing bugs immediately when they are discovered at the unit or early integration stage.

5.Implementation Phase

5. Implementation:

1. Roll out the fully tested system to the live (production) environment where end-users will access it.
2. This includes installing software, configuring servers, migrating data, and ensuring all components are operational in the real-world setting.

3. **Train users and provide documentation: Conduct training sessions for end-users and administrators to ensure they can effectively use the system.**
4. **Supply comprehensive documentation (user manuals, quick-reference guides) to support learning, troubleshooting, and best practices.**

5. Use strategies to minimize disruptions

Choose an appropriate strategy to reduce risk and business interruption:

Parallel implementation :

Run the new system alongside the old one temporarily, allowing comparison and fallback.

Phased implementation : Roll out the system in stages (e.g., by department or module) to manage impact.

Direct implementation : Replace the old system immediately with the new one (suitable for smaller, low-risk changes).

The selected strategy ensures smooth transition and minimal downtime.

6. Maintenance Phase

1. Monitor system performance and address issues:
2. Continuously track system case, usage, and performance metrics (e.g., response time, uptime, error rates) using monitoring tools.

3. Detect, Diagnose, and resolve incidents, bugs, or performance bottlenecks to ensure reliability and availability.
4. Apply updates, patches, or enhancements as needed: Regularly install security, bug fixes, and software updates to protect against vulnerabilities and improve stability.

5. Implement functional enhancements or new features based on user feedback or performance analysis to extend system value.
6. Adapt the system to evolving business or technological requirements

7. Modify the system in response to changing business needs (e.g., new processes, regulations, or market demands) or technological advancements.
8. keep the system relevant and efficient over time.

The 6 Main Phases of the Software Development Life Cycle (SDLC)

#	Phase	Main Purpose & Key Activities
1	Planning	Decide whether the project is worth doing. ▪ Define project goals and scope ▪ Perform feasibility study (technical, economic, legal, operational) ▪ Estimate cost, resources, timeline ▪ Identify risks ▪ Get approval to proceed

2	Analysis	▪ Gather detailed requirements from stakeholders ▪ Analyze and prioritize requirements ▪ Create Requirements Specification Document (SRS) ▪ Define functional & non-functional requirements

#	Phase	Main Purpose & Key Activities
3	Design	▪ Create High-Level Design (architecture, tech stack, modules) ▪ Create Low-Level Design (detailed class diagrams, database schema, UI mockups) ▪ Produce design documents that developers will follow.

#	Phase	Main Purpose & Key Activities
4	Development	Actually build the software. ▪ Developers write real code according to design documents ▪ Develop modules/ components ▪ Perform unit testing ▪ Integrate code (continuous

End