**Lec#04**

**1. What is SDLC?**

The **Software Development Life Cycle (SDLC)** is a structured process used to design, develop, test, and maintain high-quality software. It ensures that the final product meets user requirements and is delivered on time and within budget.

**2. Core Activities of SDLC**

1. **Specification**: Define what the software should do (e.g., "The app must allow users to transfer money securely").
2. **Development**: Write code and build the software.
3. **Validation**: Test the software to ensure it works correctly (e.g., checking if payments are processed without errors).
4. **Evolution**: Update the software over time to meet new needs (e.g., adding fingerprint login).

**3. Basic Terminologies**

**1. Process**

* **What?** A **process** is a set of steps or activities that need to be followed to achieve a goal. In software engineering, it’s the series of steps used to develop software.
* **Example**: The steps to build a mobile app (e.g., planning, designing, coding, testing, and launching).

**2. Model**

* **What?** A **model** is a simplified representation of a software process. It’s like a blueprint that shows how the software will be developed.
* **Example**: The **Waterfall Model** is a model that shows a linear, step-by-step approach to software development.

**3. Product**

* **What?** A **product** is the final output of a software development process. It’s also called an **artifact**.
* **Example**: A mobile app, a website, or a desktop software program.

**4. Stakeholders**

* **What?** **Stakeholders** are people or groups who have an interest in the success of the project. They can be customers, developers, managers, or even end-users.
* **Example**: For a banking app, stakeholders include the bank’s management, the app developers, and the customers who will use the app.

**5. Software System**

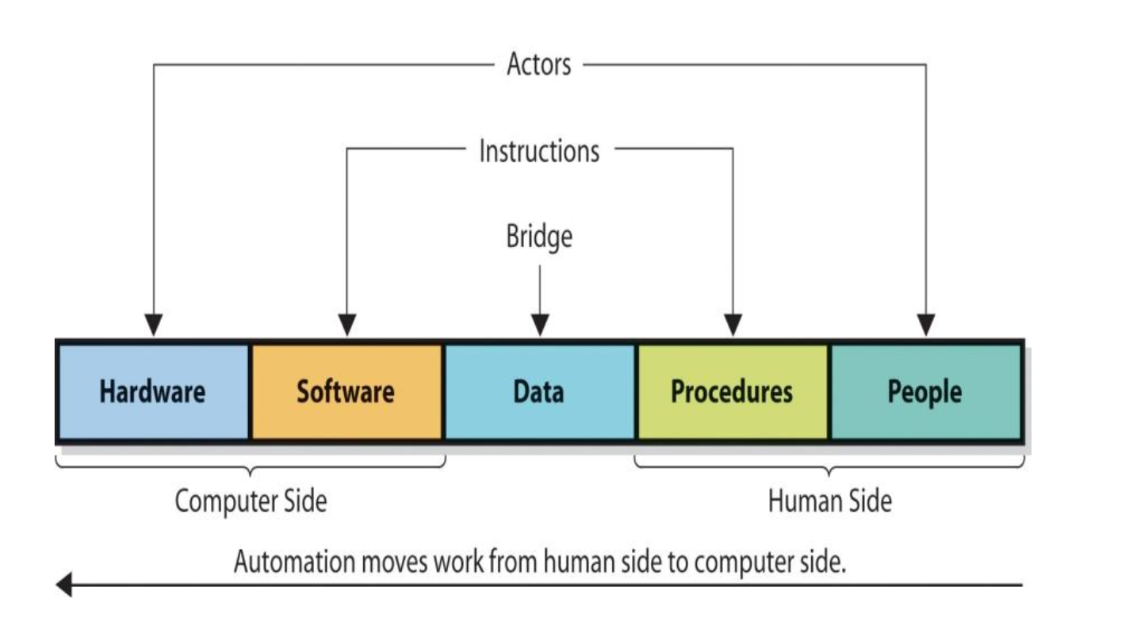
* **What?** A **software system** is a complete, working piece of software that performs specific tasks.
* **Example**: An online banking system that allows users to check their balance, transfer money, and pay bills.

**6. Project**

* **What?** A **project** is the work being done to develop the software. It includes all the activities, resources, and timelines needed to create the software.
* **Example**: The project to build a new e-commerce website.

**7. Software System Components**

**4. Software System Components**



**5. Core Activities of SDLC**

1. **Specification**: Decide what the software should do.
   * Example: A banking app must let users check their balance and transfer money.
2. **Development**: Write the code and build the software.
   * Example: Developers create a login system and a payment feature.
3. **Validation**: Test the software to make sure it works.
   * Example: Test if users can transfer money without errors.
4. **Evolution**: Update the software over time to add new features or fix issues.
   * Example: Add fingerprint login for better security.

**6. Software Process Framework**

This is the **overall structure** of the SDLC, which includes:

1. **Specification**: Define what the software should do.
2. **Modeling**: Create models (e.g., diagrams) to represent the system.
3. **Development**: Build the software.
4. **Validation**: Test the software.
5. **Deployment**: Release the software to users.
6. **Evolution**: Update the software over time

**7. Supporting Activities (Umbrella Activities)**

These are tasks that **support** the core activities throughout the SDLC:

1. **Project Tracking**: Monitor progress and ensure the project stays on schedule.
2. **Risk Management**: Identify and reduce risks (e.g., technical or budget issues).
3. **Quality Assurance**: Ensure the software meets quality standards.
4. **Technical Reviews**: Evaluate the software at different stages.
5. **Measurement**: Track metrics like time, cost, and performance.
6. **Configuration Management**: Manage changes to the software.
7. **Reusability Management**: Reuse existing components to save time and effort.
8. **Work Product Preparation**: Create documents like user manuals and reports.

**8. Software Development Principles**

These are **guidelines** to follow during software development:

1. **The Reason It All Exists**: Focus on delivering value to users.
2. **Keep It Simple, Stupid (KISS)**: Avoid unnecessary complexity.
3. **Maintain the Vision**: Stay focused on the main goal.
4. **What You Produce, Others Will Consume**: Build software that is easy for others to use and maintain.
5. **Be Open to the Future**: Design software that can adapt to changes.
6. **Plan Ahead for Reuse**: Create components that can be reused in future projects.
7. **Think**: Always analyze and plan before acting.

**9. Software Development Models (SDLC Models):**

**Software Development Process Models**

These are **different approaches** to developing software. Each model has its own strengths and weaknesses:

1. **Waterfall Model**:
   * A **linear, step-by-step** approach where each phase must finish before the next begins.
   * **Pros**: Simple, easy to manage, good for small projects.
   * **Cons**: Rigid, not suitable for complex or evolving projects.
   * **Best For**: Stable, well-defined projects (e.g., payroll systems).
2. **V-Model**:
   * Emphasizes **testing at every stage**. The "V" shape links development phases to testing phases.
   * **Pros**: Early bug detection, high quality.
   * **Cons**: Inflexible, expensive to fix errors late.
   * **Best For**: Safety-critical systems (e.g., medical devices).
3. **Iterative Model**:
   * Build the software in **repeated cycles (iterations)**. Each iteration improves the whole system.
   * **Pros**: Early working versions, flexible.
   * **Cons**: Requires more planning.
   * **Best For**: Projects where requirements might change (e.g., a new social media app).
4. **Incremental Model**:
   * Deliver the software in **small, usable parts (increments)**. Each part adds new features.
   * **Pros**: Early delivery of key features.
   * **Cons**: Integration of parts can be tricky.
   * **Best For**: Apps where basic features are needed first (e.g., a messaging app).
5. **Prototyping**:
   * Create a **quick, basic version (prototype)** to get user feedback before building the full system.
   * **Pros**: Reduces misunderstandings, early user involvement.
   * **Cons**: Can lead to scope creep.
   * **Best For**: Projects with unclear requirements (e.g., a startup’s new app).
6. **Spiral Model**:
   * Combines **iterative development** with **risk analysis**. Focuses on managing risks early.
   * **Pros**: Flexible, good for complex projects.
   * **Cons**: Expensive, time-consuming.
   * **Best For**: High-risk projects (e.g., defense systems).
7. **Agile Models (Scrum & XP)**:

**i) Extreme Programming (XP)**

**What?**: An **Agile methodology** focused on **high-quality code**, **frequent releases**, and **customer feedback**.

**Key Practices**: Pair programming, Test-Driven Development (TDD), continuous integration, small releases.

**When to Use?**: Small teams, dynamic environments, high customer involvement, focus on quality.

**Example**: Building a **startup’s mobile app** with frequent updates based on user feedback.

**ii) Scrum:**

**What?:** An **Agile framework** for **iterative development** using **sprints** (2-4 weeks).

**Key Components**: Sprints, Product Owner, Scrum Master, Daily Standups, Sprint Reviews.

**When to Use?:** Complex projects, cross-functional teams, frequent delivery, high collaboration.

**Example**: Developing an **e-commerce platform** with evolving requirements.