**Week 9**

Q1. Write a report on the stages of the system development life cycle.

**Stages of the System Development Life Cycle (SDLC)**

**Introduction**

The System Development Life Cycle (SDLC) is a structured approach to designing, developing, testing, and maintaining information systems. It provides a well-defined process for building efficient, reliable, and scalable software systems, ensuring that each stage of development is carefully planned, executed, and evaluated. This report outlines the key stages of SDLC, highlighting their importance and the steps involved in each.

**1. Planning**

The planning stage is the foundation of the SDLC. It involves understanding the project’s goals, defining its scope, and identifying resources. During this stage, feasibility studies are conducted to assess whether the project is viable in terms of technical, financial, and operational constraints. Key activities in this stage include:

* Defining the project objectives.
* Assessing potential risks and mitigation strategies.
* Estimating costs and setting timelines.
* Assigning roles and responsibilities.
* Evaluating technology needs.

The main outcome of this stage is a project plan or roadmap, which outlines the project's scope and timeline.

**2. System Analysis**

In the system analysis stage, the requirements of the system are gathered and analyzed. This stage focuses on understanding the needs of end users and defining what the system must do to meet those needs. Techniques like interviews, surveys, and documentation review are often used to gather requirements. The deliverables from this stage typically include:

* **Requirements Specification Document (RSD)**: A formal document outlining functional and non-functional requirements.
* **Data Flow Diagrams (DFD)** or other visual models: These models illustrate how data will flow through the system.

This stage is crucial for identifying potential bottlenecks or challenges early in the process.

**3. System Design**

Once the requirements are clear, the system design phase begins. Here, the blueprint for the system is created. The design phase breaks down the system's architecture, databases, user interfaces, and software modules. Key elements include:

* **High-Level Design (HLD)**: Describes the overall system architecture and design specifications.
* **Low-Level Design (LLD)**: Focuses on individual components, defining how each part of the system will function.
* **Database design**: Determines the structure of data storage and access.

The goal of this stage is to produce a detailed design document that serves as the guide for the developers.

**4. Development**

The development phase is where the actual coding of the system takes place. Developers write the code based on the design specifications. This stage often involves:

* Setting up the development environment.
* Coding and programming using appropriate languages and frameworks.
* Version control and collaborative development tools to manage code changes.

This is typically the most time-consuming stage and requires continuous integration and unit testing to ensure code quality.

**5. Testing**

After development, the system undergoes rigorous testing to identify and fix any bugs or issues. The testing phase ensures that the system meets the specified requirements and is free of defects. Key activities in this stage include:

* **Unit testing**: Testing individual components or modules.
* **Integration testing**: Ensuring that different modules work together.
* **System testing**: Verifying the system as a whole.
* **User Acceptance Testing (UAT)**: Ensuring the system meets user expectations.

Any issues discovered during testing are documented and sent back to the development team for fixes.

**6. Implementation/Deployment**

Once the system has passed testing, it is ready to be deployed in a live environment. This stage involves:

* Installing and configuring the system on production servers.
* Migrating data from old systems (if applicable).
* Training users to use the system effectively.

This stage may be done in phases (e.g., a pilot implementation) or in a full-scale rollout, depending on the project's complexity.

**7. Maintenance**

The final stage of the SDLC is maintenance. After the system is live, ongoing support is necessary to address any issues that arise, apply updates, and ensure that the system continues to meet user needs. Maintenance can be:

* **Corrective**: Fixing issues discovered post-deployment.
* **Adaptive**: Updating the system to accommodate changes in the environment, such as new operating systems or regulations.
* **Perfective**: Enhancing the system to improve performance or add new features.

The maintenance phase is continuous and ensures that the system remains functional and relevant.

**Conclusion**

The System Development Life Cycle is a crucial framework for managing the complexities of software development. Each stage, from planning to maintenance, plays a vital role in ensuring that the final product is reliable, scalable, and meets user expectations. By following the structured approach of SDLC, organizations can reduce risks, improve efficiency, and achieve better outcomes in system development projects.

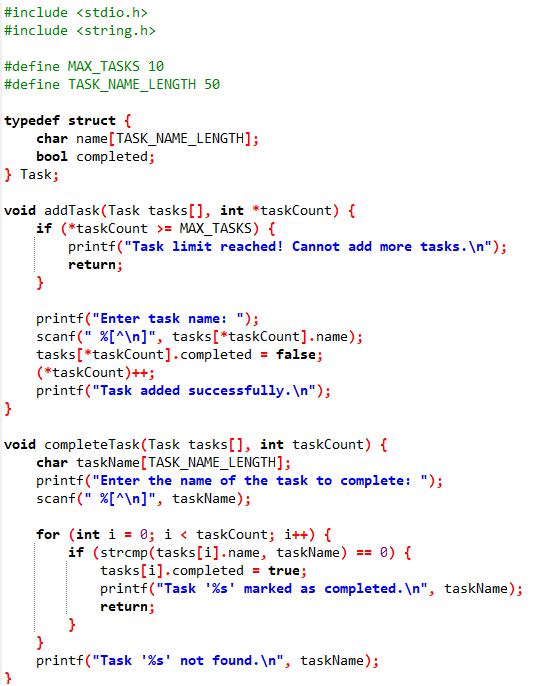
Q2. Compare and contrast SSAD and OOAD.

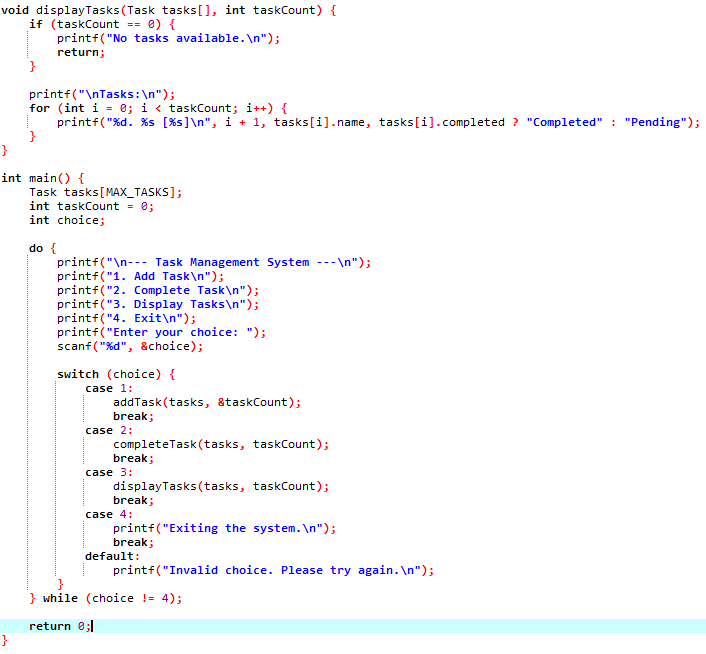
|  |  |  |
| --- | --- | --- |
| **Aspect** | **SSAD (Structured Systems Analysis and Design)** | **OOAD (Object-Oriented Analysis and Design)** |
| Approach | Function-Oriented: Focuses on breaking down system functions. | Object-Oriented: Focuses on modeling real-world objects. |
| Design Philosophy | Top-Down: Starts with a high-level view of the system. | Bottom-Up: Starts with objects and their relationships. |
| Focus | Data flow between processes and functions. | Interactions between objects with encapsulated data and behavior. |
| Modeling Techniques | Data Flow Diagrams (DFDs), Entity-Relationship Diagrams (ERDs). | Use Case Diagrams, Class Diagrams, Sequence Diagrams. |
| Separation | Treats data and processes as separate entities. | Combines data and behavior in a single entity (object). |
| Development | Process-centric: Breaks down the system into processes. | Object-centric: Models the system as a collection of objects. |
| Reusability | Limited reuse; functions are often tightly coupled to data. | High reuse; objects and classes can be reused across systems. |
| Scalability | Difficult to scale for large systems. | Well-suited for large, complex, and scalable systems. |
| Flexibility | Rigid design; changes in requirements lead to widespread modifications. | Flexible; changes often localized to specific objects or classes. |
| Suitability | Best for smaller, process-driven systems (e.g., payroll systems). | Ideal for complex, modular systems (e.g., enterprise applications). |
| Learning Curve | Easier to learn and implement for smaller systems. | Steeper learning curve, especially with advanced OOP concepts. |
| Applications | Suitable for legacy systems and data-centric applications. | Best for systems using OOP languages (Java, C++, Python). |
| Design Components | Processes, data flows, and data stores. | Objects, classes, attributes, and methods. |
| Maintenance | More difficult to maintain as the system grows. | Easier to maintain due to encapsulation and modularity. |

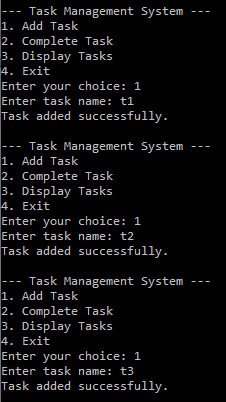
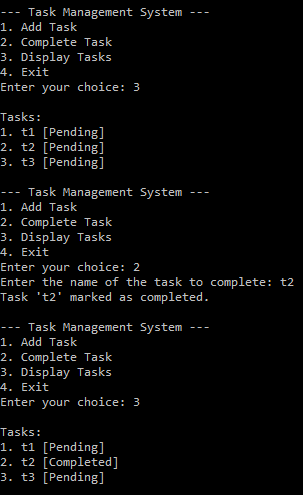
Q3. Create a diagram representing the SDLC stages.

****

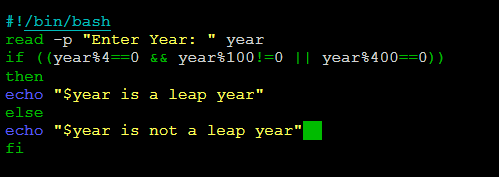
Q4.Write a C program to simulate a simple system development task.

****

****

** **

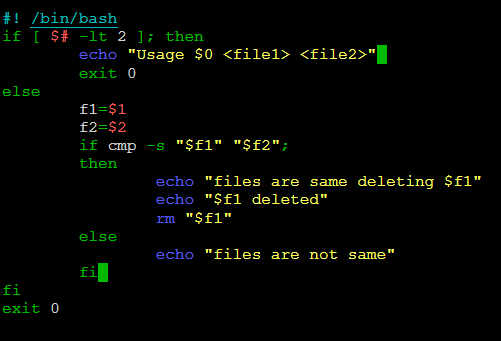
Q5.Write a shell script which receives any year from the keyboard and determines whether the year is a leap year or not. If no argument is supplied the current year should be assumed.



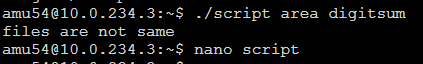
**Output**

**Q.5 output.png**

Q6. Write a shell script which receives two filenames as arguments. It should check whether the two file’s contents are same or not. If they are same the second file should be deleted. (Hint: Use the cmp command to compare files)



**Output**

****