Assignment (Module - 1 Fundamentals)

Q:- What is SDLC ?

A:- SDLC is a **structured process** followed by software development teams to design, develop, test, and maintain high-quality software. It ensures that software is built **systematically, efficiently**, and with **minimal risk**.

### **Phases of SDLC:**

1. **Requirement Gathering & Analysis**
   * Understand what the customer wants.
   * Gather functional and non-functional requirements.
2. **Planning**
   * Estimate cost, time, and resources.
   * Prepare schedules and risk management plans.
3. **Design**
   * Create software architecture.
   * Design database, UI, and system flow.
4. **Development**
   * Actual coding begins.
   * Developers write the code based on the design.
5. **Testing**
   * Test the software for bugs, errors, and performance.
   * Ensure it meets the requirements.
6. **Deployment**
   * Release the software to users.
   * Either in stages (beta) or full release.
7. **Maintenance**
   * Fix issues after deployment.
   * Add updates and new features if required.

### **Purpose of SDLC:**

* Improve software **quality**.
* Ensure **on-time** and **within-budget** delivery.
* Reduce **project risks**.

Q:- What is software testing?

A:- **Software Testing** is the process of **evaluating a software application** to check whether it meets the **required specifications** and to ensure that it is **free of defects or bugs**.

### **Purpose of Software Testing:**

* To **verify** that the software works as expected.
* To **identify bugs** or errors before the software is released.
* To ensure **quality**, **performance**, **reliability**, and **security** of the product.

### **ypes of Software Testing:**

1. **Manual Testing**
   * Testers check the software manually without using tools.
2. **Automation Testing**
   * Automated tools (like Selenium, JUnit, etc.) are used to test software.

### **Levels of Software Testing:**

1. **Unit Testing** – testing individual components or functions.
2. **Integration Testing** – testing how components work together.
3. **System Testing** – testing the complete system as a whole.
4. **Acceptance Testing** – checking if the software is ready for delivery.

### **Benefits of Testing:**

* Improves software **quality**
* Increases **user satisfaction**
* Reduces **costs** by detecting bugs early

Q:- What is agile methodology?

A:- **Agile methodology** is a **flexible and iterative approach** to software development where the product is built in **small, workable parts** (called iterations or sprints), with **continuous feedback** from customers or users.

### **Key Features of Agile:**

1. **Iterative development** – software is built in repeated cycles (called sprints).
2. **Customer collaboration** – clients are involved throughout the process.
3. **Flexibility to change** – changes in requirements are welcomed, even late in the project.
4. **Working software delivery** – priority is to deliver a working product quickly.
5. **Daily stand-up meetings** – short daily team meetings to track progress.

### **Example Agile Framework: Scrum**

* **Sprint**: 1–4 weeks cycle of development.
* **Product Owner**: Person who defines the features.
* **Scrum Master**: Facilitates the team and removes obstacles.
* **Team**: Developers, testers, etc.

### **Agile vs Traditional (Waterfall):**

|  |  |
| --- | --- |
| **Waterfall** | **Agile** |
| Sequential process | Iterative & incremental |
| Customer at the end | Customer involved always |
| Hard to make changes | Flexible to changes |

Q:- What is SRS ?

A:- **SRS** stands for **Software Requirements Specification**.

SRS is a **detailed document** that describes the **functional and non-functional requirements** of a software system. It acts as an **agreement** between the client and the development team.

### **Purpose of SRS:**

* To clearly define **what the software will do**.
* To avoid **misunderstandings** between stakeholders.
* To serve as a **reference** throughout the development and testing phases.

### **Contents of an SRS Document:**

1. **Introduction**
   * Purpose, scope, definitions, overview
2. **Overall Description**
   * Product perspective, user needs, assumptions
3. **Functional Requirements**
   * What the system should do (features, inputs, outputs)
4. **Non-functional Requirements**
   * Performance, reliability, security, usability, etc.
5. **System Requirements**
   * Hardware and software requirements
6. **Use Cases / Diagrams**
   * Visual representation of system behavior

### **Example (Functional Requirement):**

“The system shall allow users to log in using email and password.”

Q:- What is **OOPs**?

A:- **OOPs** stands for **Object-Oriented Programming System**.

It is a programming paradigm (style) that is based on the concept of **“objects”**, which contain **data** and **methods** to operate on that data.

### **Purpose of OOPs:**

* To make code more **modular**, **reusable**, and **easier to maintain**.
* To model real-world entities more effectively.

### **Four Main Pillars of OOPs:**

1. **Encapsulation**
   * Wrapping data and code together in a single unit (class).
   * Example: A class with private variables and public methods to access them.
2. **Abstraction**
   * Hiding complex implementation and showing only essential features.
   * Example: You use a car's steering wheel without knowing the internal engine mechanism.
3. **Inheritance**
   * One class can inherit features (properties and methods) from another.
   * Example: A Car class can inherit from a Vehicle class.
4. **Polymorphism**
   * One function or method behaves differently based on context.
   * Example: drive() method behaves differently for Bike and Car.

Q:- Write Basic Concepts of **OOPs**.

A:- **Basic Concepts of OOPs (Object-Oriented Programming)**

There are **6 fundamental concepts** in Object-Oriented Programming that form the foundation of OOP languages like Java, C++, Python, PHP, etc.

### **1. Class**

* A **blueprint** for creating objects.
* It defines properties (variables) and methods (functions).

### **2. Object**

* An **instance** of a class.
* It is a real-world entity that holds data and can perform actions.

### **3. Encapsulation**

* Binding data and methods into a single unit (class).
* Protects data using **access modifiers** like private, public, protected.

### **4. Abstraction**

* Shows only **essential features** and hides the internal details.
* Reduces complexity and increases efficiency.
* Achieved using **abstract classes** or **interfaces**.

### **5. Inheritance**

* One class can **inherit** the properties and methods of another class.
* Promotes **code reusability**.

### **6. Polymorphism**

* One method behaves **differently** based on the object.
* Types:  
  + **Compile-time (Method Overloading)** – Not supported directly in PHP.
  + **Run-time (Method Overriding)** – Supported.

Q:- What is **object** ?

A:- An **object** is a **real-world entity** or an **instance of a class** in Object-Oriented Programming (OOP).

An **object** is something that has **state** (data) and **behavior** (functions or methods).

### **Real-World Example:**

Think of a **Car**:

* **State (Properties)**: color, brand, speed
* **Behavior (Methods)**: start(), stop(), drive()

So, a **car** is an object created from the **Car class** blueprint.

### **Key Characteristics of Objects:**

* **Identity**: Unique name or reference (like $myCar)
* **State**: Values of its properties (like color = Red)
* **Behavior**: Methods it can perform (like drive())

Q:- What is **Class**?

A:- A **class** is a **blueprint or template** used to create **objects** in Object-Oriented Programming (OOP).

A class defines the **properties (data)** and **methods (functions)** that its objects will have.

### **Purpose of a Class:**

* Organizes code into **reusable** and **modular** units.
* Represents **real-world entities** in programming (like Car, Person, BankAccount, etc.).

### **Key Points:**

* A **class** does **not occupy memory** until you create an **object**.
* Objects are created **from** classes.

Q:- What is **Encapsulation** ?

A:- **Encapsulation** is one of the **core principles of Object-Oriented Programming (OOPs)**.

It means **binding data (variables)** and the **methods (functions)** that operate on that data into a **single unit**, typically a **class**.

**Encapsulation** is the process of **hiding internal details** of an object and only exposing what is necessary.

### **Purpose of Encapsulation:**

* To **protect** data from direct access and unauthorized modification.
* To make code **secure**, **modular**, and **easy to maintain**.
* Helps achieve **data hiding**.

### **How it is done:**

1. **Declare variables as private**
2. **Provide public getter and setter methods** to access and modify them

### **Key Benefits:**

* Protects the object’s internal state
* Prevents accidental changes
* Makes the code cleaner and easier to debug

Q:- What is **inheritance**?

A:- **Inheritance** is a fundamental concept in Object-Oriented Programming (OOP) that allows a **class to inherit properties and methods** from another class.

**Inheritance** means one class (child) can **reuse the code** of another class (parent).

### 

### 

### **Real-World Example:**

* **Parent class**: Vehicle → has methods like start(), stop()
* **Child class**: Car → inherits start(), stop() and can have its own method openSunroof()

### **Types of Inheritance (General OOP – PHP supports some):**

1. **Single Inheritance** – One child, one parent ✔️
2. **Multilevel Inheritance** – Class inherits from a child of another class ✔️
3. **Hierarchical Inheritance** – Multiple children from the same parent ✔️
4. **Multiple Inheritance** – One child inherits from multiple parents ❌ (Not directly supported in PHP)

### **Benefits:**

* **Code reusability**
* **Faster development**
* **Easy maintenance**

Q:- What is **Polymorphism**?

A:- **Polymorphism** means "**many forms**".  
 In Object-Oriented Programming (OOP), polymorphism allows **the same method or function to behave differently** depending on the object or context.

**Polymorphism** allows one interface (function/method) to be used for **different types** of objects.

### **Types of Polymorphism:**

1. **Compile-Time Polymorphism (Method Overloading)**
   * Same method name with **different parameters**
   * *Not directly supported in PHP*
2. **Run-Time Polymorphism (Method Overriding)**
   * A **child class redefines** a method of the parent class

### **Why Polymorphism is Useful:**

* Increases **flexibility** and **maintainability**
* Enables use of **common interfaces** for different types
* Helps implement **dynamic behavior** at runtime

Q:- Draw Usecase on online bill payment system (**paytm**).

A:- [PayTM bill.drawio](https://drive.google.com/file/d/1fUYcopEDlY1g9C-sFCpRFTkbUSSW4V0R/view?usp=sharing)

Q:- Draw Usecase on banking system for customers.

A:-

[Banking System.drawio](https://drive.google.com/file/d/12NF3PhH8gwr3spsAUgW99I5GPfVur0I_/view?usp=sharing)

Q:- Draw Usecase on Broadcasting System.

A:-

[Broadcasting System .drawio](https://drive.google.com/file/d/1fpatRisAXwI40b183pHCCSA5Tep08F--/view?usp=sharing)

Q:- Write SDLC phases with basic introduction.

A:- **SDLC** stands for **Software Development Life Cycle**.

It is a **step-by-step process** used by software developers and project managers to **plan, design, develop, test, and maintain software**.

### **SDLC Phases (with basic introduction):**

#### **1. Requirement Gathering and Analysis**

* Understand what the client wants.
* Meetings with stakeholders to gather functional and non-functional requirements.
* Output: **SRS (Software Requirement Specification)** document.

#### **2. System Design**

* Based on SRS, software architecture is designed.
* Defines **overall system structure**, modules, database, and UI design.
* Output: Design documents like **DFD**, ER diagrams, etc.

#### **3. Implementation (Coding)**

* Actual coding starts based on the design.
* Developers write code using selected languages and tools.
* Output: **Working software modules**.

#### **4. Testing**

* Testing team checks software for **bugs, errors, and quality issues**.
* Types: Unit Testing, Integration Testing, System Testing, User Acceptance Testing.
* Output: **Tested and verified software**.

#### **5. Deployment**

* The software is deployed to the **user environment** or live server.
* May be done in stages (beta, full release).
* Output: **Software is now in use**.

#### **6. Maintenance**

* After deployment, issues may arise that need fixing.
* Includes updates, bug fixes, and enhancements.
* Ensures **software runs smoothly long-term**.

Q:- Explain Phases of the **Waterfall model**.

A:- The **Waterfall Model** is one of the **oldest and most traditional** software development models.

It follows a **linear and sequential** flow — each phase must be completed **before the next one begins**.

### **Phases of the Waterfall Model:**

#### **1. Requirement Gathering & Analysis**

* All **user needs and system requirements** are collected.
* Focus is on **what the software must do**.
* Output: **SRS (Software Requirements Specification)** document.

#### **2. System Design**

* Design the software architecture based on requirements.
* Includes **UI design, database schema, data flow**, etc.
* Output: **Design Documents** (like flowcharts, DFDs, ER diagrams).

#### **3. Implementation (Coding)**

* Developers write the actual **source code**.
* Code is written in modules based on the design.
* Output: **Executable software modules**.

#### **4. Testing**

* Each module and the full system are tested for **bugs, errors, and security**.
* Types: Unit Testing, Integration Testing, System Testing.
* Output: **Bug-free and validated software**.

#### **5. Deployment**

* The finished product is **installed and launched** for the end-users.
* Can be done on a **server, website, app store**, etc.
* Output: **Live running system**.

#### **6. Maintenance**

* Fix issues that arise after deployment.
* Provide **updates, upgrades, and technical support**.
* Output: **Stable and improved system over time**.

### **Important Points:**

* Once a phase is completed, **you can't go back** (like a waterfall).
* Best used for **small and well-understood projects**.
* Not flexible for **changing requirements**.

Q:- Write phases of **Spiral Model.**

A:- The **Spiral Model** is a **risk-driven software development model** that combines elements of both **Waterfall** and **Prototyping** models.

It focuses on **repeating cycles (spirals)**, allowing for **continuous refinement** through each phase.

Software is developed in a series of **iterations (spirals)**, and each spiral includes planning, risk analysis, development, and evaluation.

### **Phases of the Spiral Model:**

#### **1. Planning Phase**

* Identify the **objectives, requirements**, and constraints for the current cycle.
* Includes requirement gathering, cost estimation, scheduling.

#### **2. Risk Analysis Phase**

* Identify and analyze **potential risks** (technical, business, time-related).
* Create strategies to **reduce or eliminate risks**.
* This is the **key feature** that differentiates Spiral from other models.

#### **3. Engineering (Development & Testing) Phase**

* Based on planning and risk handling, **develop a prototype** or part of the system.
* **Code, test, and integrate** the components for the current cycle.

#### **4. Evaluation (Customer Review) Phase**

* The customer or stakeholders **evaluate the output** of this iteration.
* Gather feedback to improve the next cycle.

Each cycle leads to the next **version of the product**, refining it step by step.

### **Advantages:**

* Ideal for **large and high-risk projects**
* **Flexible** and supports changes
* Frequent **customer feedback**

Q:- Write **Agile Manifesto** principles.

A:- The **Agile Manifesto** outlines **12 guiding principles** that form the foundation of Agile software development. These principles emphasize **customer collaboration, flexibility, and continuous improvement**.

### **12 Principles of the Agile Manifesto:**

1. **Customer Satisfaction** Our highest priority is to **satisfy the customer** through early and continuous delivery of valuable software.
2. **Welcome Changing Requirements** Even late in development, agile processes **welcome changes** to meet customer needs.
3. **Frequent Delivery** Deliver working software **frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. **Collaboration** **Business people and developers must work together** daily throughout the project.
5. **Motivated Individuals** Build projects around **motivated individuals**. Give them the environment and support they need, and trust them to get the job done.
6. **Face-to-Face Conversation** The most efficient and effective method of conveying information is **face-to-face communication**.
7. **Working Software is the Measure of Progress** The primary measure of progress is **working software**.
8. **Sustainable Development** Agile processes promote **sustainable development**. Teams should be able to maintain a **constant pace** indefinitely.
9. **Technical Excellence** Continuous attention to **technical excellence and good design** enhances agility.
10. **Simplicity**

Simplicity—the art of **maximizing the amount of work not done**—is essential.

1. **Self-Organizing Teams**

The best architectures, requirements, and designs **emerge from self-organizing teams**.

1. **Reflect and Adjust**

At regular intervals, the team reflects on how to become more effective, then **tunes and adjusts** its behavior accordingly.

### **Summary:**

Agile isn't just a process—it's a **mindset** that focuses on:

* **Customer collaboration**
* **Rapid delivery**
* **Adapting to change**
* **Continuous improvement**

Q:- Explain the working methodology of the Agile **model** and also write pros and cons.

A:- The **Agile Model** is an **iterative and incremental approach** to software development.  
 It focuses on **delivering small, workable parts of the software** frequently, with **continuous customer feedback and improvements** in every cycle (called a **sprint**).

### 

### **Working Methodology of Agile Model**

1. **Project is broken into small iterations** (sprints), usually 1–4 weeks long.
2. In each sprint:  
   * Requirements are discussed with the client.
   * A **small module** is developed, tested, and delivered.
   * Regular team meetings (**daily stand-ups**) are held.
   * Customer feedback is collected and used in the next sprint.
3. This process **repeats** until the full product is developed.

Each cycle results in a **usable version** of the product.

### **Key Features:**

* Short, time-boxed development cycles (Sprints)
* Continuous client involvement
* Flexible to changes
* Working software delivered early and often

**Advantages of Agile Model:**

**Pros**

* Faster delivery of usable software
* Highly flexible to changes
* Frequent client feedback improves quality
* Encourages teamwork and communication
* Reduces project risk through early testing and delivery
* Focus on customer satisfaction

**Disadvantages of Agile Model:**

**Cons**

* Difficult to predict final cost and time
* Requires experienced team members
* Not ideal for small projects with fixed requirements
* Documentation may be limited
* Constant involvement from client is needed

### **Best Used For:**

* Medium to large projects with **changing requirements**
* Teams that work closely with clients
* Projects that need **early delivery and feedback**

Q:- Draw Use Case on **OTT Platform.**

A:-

[OTT Platform.drawio](https://drive.google.com/file/d/1ZDSWbX0AuQgJAPsl-tEO08XkPfKRi6sb/view?usp=sharing)

Q:- Draw Use Case on **E-commerce** **application**.

A:-

[E Commerce.drawio](https://drive.google.com/file/d/1vq5H-oVgNshZbGe7e0MP3-1t3_sZKwNs/view?usp=sharing)

Q:- Draw Use Case on **Online shopping products** using **payment gateway.**

A:-

[Payment Gateway.drawio](https://drive.google.com/file/d/1VpUnvo-zJkOxDpFt12C8WVsrmeAggt82/view?usp=sharing)