# **TOPS TECHNOLOGY**

# **MODULO -1**

1. **What is SDLC**

* SDLC is a structure imposed on the development of a software product that defines the process for planning, implementation, testing, documentation, deployment, and ongoing maintenance and support.
* There are a number of different development models. A Software Development Life Cycle is essentially a series of steps, or phases, that provide a model for the development and lifecycle management of an application or piece of software.
* The methodology within the SDLC process can vary across industries and organizations, but standards such as ISO/IEC 12207 represent processes that establish a lifecycle for software, and provide a mode for the development, acquisition, and configuration of software systems.
* **SDLC Phases**
* **Requirements Collection/Gathering**

**-** Establish Customer Needs.

* **Analysis**

**-** Model and Specify the requirements- “What”.

* **Implementation**

**-** Construct a Solution in Software.

* **Testing**

**-** Validate the solution against the requirements.

* **Maintenance**

**-** Repair defects and adapt the solution to the new requirements.

1. **What is software testing?**

* Software Testing is a process used to identify the correctness, completeness, and quality of developed computer software.
* Test execution is only a part of testing, but not all of the testing activities
* Test activities exist before and after test execution
* It can also be stated as the process of validating and verifying that a software program or application or product **:**
  + Meets the business and technical requirements that guided it’s design and development
* Works as expected
* Can be implemented with the same characteristic
* **Let’s break the definition of Software testing into the following parts:**
* **Process**:

Testing is a process rather than a single activity.

* **All Life Cycle Activities:**

Testing is a process that’s take place throughout the Software Development Life Cycle (SDLC). The process of designing tests early in the life cycle can help to prevent defects from being introduced in the code. Sometimes it’s referred as “verifying the test basis via the test design”.

The test basis includes documents such as the requirements and design specifications.

* **Static Testing:**

It can test and find defects without executing code. Static Testing is done during verification process. This testing includes reviewing of the documents (including source code) and static analysis. This is useful and cost effective way of testing. For example: reviewing, walkthrough, inspection, etc.

* **Dynamic Testing:**

In dynamic testing the software code is executed to demonstrate the result of running tests. It’s done during validation process. For example: unit testing, integration testing, system testing, etc.

* **Planning:**

We need to plan as what we want to do. We control the test activities, we report on testing progress and the status of the software under test.

* **Preparation:**

We need to choose what testing we will do, by selecting test conditions and designing test cases.

* **Evaluation:**

During evaluation we must check the results and evaluate the software under test and the completion criteria, which helps us to decide whether we have finished testing and whether the software product has passed the tests.

* **Software products and related work products:**

Along with the testing of code the testing of requirement and design specifications and also the related documents like operation, user and training material is equally important.

1. **What is agile methodology?**

* Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.
* Agile Methods break the product into small incremental builds.
* These builds are provided in iterations.
* Each iteration typically lasts from about one to three weeks.
* Every iteration involves cross functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing.
* At the end of the iteration a working product is displayed to the customer and important stakeholders.
* Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In agile the tasks are divided to time boxes (small time frames) to deliver specific features for a release.
* Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.
* Agile thought process had started early in the software development and started becoming popular with time due to its flexibility and adaptability.

1. **What is SRS**

* A software requirements specification (SRS) is a complete description of the behavior of the system to be developed.
* It includes a set of use cases that describe all of the interactions that the users will have with the software.
* Use cases are also known as functional requirements. In addition to use cases, the SRS also contains non functional (or supplementary) requirements.
* Non-functional requirements are requirements which impose constraints on the design or implementation (such as performance requirements, quality standards, or design constraints).
* Recommended approaches for the specification of software requirements are described by IEEE 830-1998.
* This standard describes possible structures, desirable contents, and qualities of a software requirements specification.

1. **What is oops**

* Identifying objects and assigning responsibilities to these objects.
* Objects communicate to other objects by sending messages.
* Messages are received by the methods of an object
* An object is like a black box.
* An object is like a black box.
* Object is derived from abstract data type
* Object-oriented programming has a web of interacting objects, each house-keeping its own state.
* Object-oriented programming has a web of interacting objects, each house-keeping its own state.
* If you can write a demonstration, you can make a program.
* So, programming is also easy. But, actually, programming is not so easy, because a real good program is not easily programmed. It needs the programmers’ lots of wisdom, lots of knowledge about programming and lots of experience.
* It is like writing, to be a good writer needs lots of experience and lots of knowledge about the world. Learning and practise is necessary
* **Object-Oriented Languages**
* An object-based programming language is one which easily supports object-orientation.
* C++
* Java
* C#
* Identifying objects and assigning responsibilities to these objects.
* Objects communicate to other objects by sending messages. Messages are received by the methods of an object.
* An object is like a black box.
* The internal details are hidden.
* Object is derived from abstract data type Object-oriented programming has a web of interacting objects, each house-keeping its own state.
* Objects of a program interact by sending messages to each other.

1. **Write Basic Concepts of oops**

* Object
* Class
* Encapsulation
* Inheritance
* Polymorphism

**\*** Overriding

**\*** Overloading

* Abstraction

1. **What is object**

* An object represents an individual, identifiable item, unit, or entity, either real or abstract, with a well-defined role in the problem domain.
* An "object" is anything to which a concept applies.
* This is the basic unit of object oriented programming(OOP).
* That is both data and function that operate on data are bundled as a unit called as object.
* **The two parts of an object Object = Data + Methods**

Object = Data + Methods

or

to say the same differently

* An object has the responsibility to know and the responsibility to do.

1. **What is class**

* When you define a class, you define a blueprint for an object.
* This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.
* A class represents an abstraction of the object and abstracts the properties and behavior of that object.
* Class can be considered as the blueprint or definition or a template for an object and describes the properties and behavior of that object, but without any actual existence.
* An object is a particular instance of a class which has actual existence and there can be many objects (or instances) for a class.
* In the case of a car or laptop, there will be a blueprint or design created first and then the actual car or laptop will be built based on that.
* We do not actually buy these blueprints but the actual objects.

1. **What is encapsulation**

* Encapsulation is the practice of including in an object everything it needs hidden from other objects. The internal state is usually not accessible by other objects.
* Encapsulation is placing the data and the functions that work on that data in the same place. While working with procedural languages, it is not always clear which functions work on which variables but object oriented programming provides you framework to place the data and the relevant functions together in the same object.
* Encapsulation in Java is the process of wrapping up of data (properties) and behavior (methods) of an object into a single unit; and the unit here is a Class (or interface).
* Encapsulate in plain English means to enclose or be enclosed in or as if in a capsule. In Java, a class is the capsule (or unit).
* Encapsulation enables data hiding, hiding irrelevant information from the users of a class and exposing only the relevant details required by the user.
* We can expose our operations hiding the details of what is needed to perform that operation
* We can protect the internal state of an object by hiding its attributes from the outside world (by making it private), and then exposing them through setter and getter methods. Now modifications to the object internals are only controlled through these methods.
* In Java, everything is enclosed within a class or interface, unlike languages such as C and C++, where we can have global variables outside classes.

1. **What is inheritance**

* Inheritance means that one class inherits the characteristics of another class. This is also called a “is a” relationship.
* One of the most useful aspects of object-oriented programming is code reusability. As the name suggests Inheritance is the process of forming a new class from an existing class that is from the existing class called as base class, new class is formed called as derived class.
* This is a very important concept of object-oriented programming since this feature helps to reduce the code size.
* Inheritance describes the relationship between two classes. A class can get some of its characteristics from a parent class and then add unique features of its own.
* In general, Java supports single-parent, multiple-children inheritance and multilevel inheritance (Grandparent-> Parent -> Child) for classes and interfaces. Java supports multiple inheritances (multiple parents, single child) only through interfaces.
* In a class context, inheritance is referred to as implementation inheritance, and in an interface context, it is also referred to as interface inheritance.
* For example consider a Vehicle parent class and its child class Car.
* Vehicle class will have all common properties and functionalities for all vehicles in common and Car will inherit those common properties from the Vehicle class and then add those properties which are specific to a car.
* Here, Vehicle is known as base class, parent class, or super class.
* Car is known as derived class, Child class or subclass.
* A car is a vehicle
* A dog is an animal
* A teacher is a person

1. **What is polymorphism**

* Polymorphism means “having many forms”.
* It allows different objects to respond to the same message in different ways, the response specific to the type of the object.
* The most important aspect of an object is its behaviour (the things it can do). A behaviour is initiated by sending a message to the o
* bject (usually by calling a method).
* The ability to use an operator or function in different ways in other words giving different meaning or functions to the operators or functions is called polymorphism.
* Poly refers to many. That is a single function or an operator functioning in many ways different upon the usage is called polymorphism.
* E.g. the message displayDetails() of the Person class should give different results when send to a Student object (e.g. the enrolment number).
* The ability to change form is known as polymorphism.
* There is two types of polymorphism in Java
* Compile time polymorphism(Overloading)
* Runtime polymorphism(Overriding)

1. **Draw Usecase on Online book shopping**

**https://app.diagrams.net/#G1FqSlOD2q9gGh-0L5Ar94oKb8dmojCzjh#%7B%22pageId%22%3A%22dpItedeFby\_4ksfA-dKd%22%7D**

1. **Draw Usecase on online bill payment system (paytm)**

**https://app.diagrams.net/#G1Tz7hMtxgOUUhWLBF1omTOVjhE7oukLWs#%7B%22pageId%22%3A%22XC8LjOhxze-NxXyIzf5o%22%7D**

1. **Write SDLC phases with basic introduction**

* **SDLC PHASES :**

1. **Requirements Collection/Gathering**
2. **Analysis**
3. **Design**
4. **Implementation**
5. **Testing**
6. **Maintenance.**

.

* **Requirement Gathering :**

1. Requirements definitions usually consist of natural language, supplemented by (e.g., UML) diagrams and tables.
2. Although requirements may be documented in written form, they may be incomplete, unambiguous, or even incorrect.
3. Requirements will Change!
4. User and business needs change during the project
5. Validation is needed throughout the software lifecycle, not only when the “final system” is delivered.
6. Early prototyping [e.g., UI] can help clarify the requirements

* **Three types of problems can arise:**

1. **Lack of clarity:** It is hard to write documents that are both precise and easy-to-read.
2. **Requirements confusion:** Functional and Non-functional requirements tend to be intertwined.
3. **Requirements Amalgamation:** Several different requirements may be expressed together.

**Types of Requirements :**

1. **Functional Requirements:** describe system services or functions.

* Compute sales tax on a purchase
* Update the database on the server

1. **Non-Functional Requirements:** are constraints on the system or

the development process.

Non-functional requirements may be more critical than functional requirements.

If these are not met, the system is useless!

* **Analysis:**

1. The analysis phase defines the requirements of the system, independent of how these requirements will be accomplished.
2. This phase defines the problem that the customer is trying to solve.
3. This phase defines the problem that the customer is trying to solve.
4. This phase defines the problem that the customer is trying to solve.
5. Details on computer programming languages and environments, machines, packages, application architecture, distributed architecture layering, memory size, platform, algorithms, data structures, global type definitions, interfaces, and many other engineering details are established.
6. The requirement documentaries to capture the requirements from the customer's perspective by defining goals.

* **Design:**

1. The Design team can now expand upon the information established in the requirement document.
2. The requirement document must guide this decision process.
3. Analyzing the trade-offs of necessary complexity allows for many things to remain simple which, in turn, will eventually lead to a higher quality product.

**\*** Design Architecture Document

\* Implementation Plan

\* Critical Priority Analysis

**\***  Performance Analysis

\* Test Plan

* **Implementation:**

1. In the implementation phase, the team builds the components either from scratch or by composition.
2. Given the architecture document from the design phase and the requirement document from the analysis phase, the team should build exactly what has been requested, though there is still room for innovation and flexibility.
3. The implementation phase deals with issues of quality, performance, baselines, libraries, and debugging.
4. The end deliverable is the product itself. There are already many established techniques associated with implementation.
5. For example, a component may be narrowly designed for this particular system, or the component may be made more general to satisfy a reusability guideline.

**1.** Implementation – Code

**2.** Critical Error Removal

* **Testing:**

1. A customer satisfied with the quality of a product will remain loyal and wait for new functionality in the next version.
2. It is emuch easier to explain to a customer why there is a missing feature than to explain to a customer why the product lacks quality.
3. Simply stated, quality is very important. Many companies have not learned that quality is important and deliver more claimed functionality but at a lower quality level.
4. The testing phase is a separate phase which is performed by a different team after the implementation is completed.
5. There is merit in this approach; it is hard to see one’s own mistakes, and a fresh eye can discover obvious errors much faster than the person who has read and re-rad the material many times.
6. Quality is a distinguishing attribute of a system indicating the degree of excellence.

Regression Testing

Internal Testing

Unit Testing Application Testing

Stress Testing

* **Maintenance phase :**
* Software maintenance is one of the activities in software engineering, and is the process of enhancing and optimizing deployed software (software release), as well as fixing defects.
* Software maintenance is also one of the phases in the System Development Life Cycle (SDLC), as it applies to software development. The maintenance phase is the phase which comes after deployment of the software into the field.
* Maintenance is the process of changing a system after it has been deployed.
* Corrective maintenance: identifying and repairing defects
* Adaptive maintenance: adapting the existing solution to the new platforms.
* Perfective Maintenance: implementing the new requirements
* The developing organization or team will have some mechanism to document and track defects and deficiencies.
* The software is released with the issues because the development organization decides the utility and value of the software at a particular level of quality outweighs the impact of the known defects and deficiencies.

1. **Explain Phases of the waterfall model**

* The six phases of the waterfall model are :

1. **Requirements:** The first phase involves gathering requirements from stakeholders and analyzing them to understand the scope and objectives of the project.
2. **Design:** Once the requirements are understood, the design phase begins. This involves creating a detailed design document that outlines the software architecture, user interface, and system components.
3. **Development:** The Development phase include implementation involves coding the software based on the design specifications. This phase also includes unit testing to ensure that each component of the software is working as expected.
4. **Testing:** In the testing phase, the software is tested as a whole to ensure that it meets the requirements and is free from defects.
5. **Deployment:** Once the software has been tested and approved, it is deployed to the production environment.
6. **Maintenance:** The final phase of the Waterfall Mmaintenance, which involves fixing any issues that arise after the software has been deployed and ensuring that it continues to meet the requirements over time.

1. **Write phases of spiral model**

* **Phases of the Spiral Model**

1. **Objectives Defined: I**n first phase of the spiral model we clarify what the project aims to achieve, including functional and non-functional requirements.
2. **Risk Analysis:** In the risk analysis phase, the risks associated with the project are identified and evaluated.
3. **Engineering:** In the engineering phase, the software is developed based on the requirements gathered in the previous iteration.
4. **Evaluation:** In the evaluation phase, the software is evaluated to determine if it meets the customer’s requirements and if it is of high quality.
5. **Planning:** The next iteration of the spiral begins with a new planning phase, based on the results of the evaluation.
6. **Write agile manifesto principles**

* **Customer Satisfaction through Early and Continuous Delivery:** This principle concentrates on the importance of customer satisfaction by providing information to customers early on time and also with consistency throughout the development process.
* **Welcome Changing Requirements, Even Late in Development:**Agile processes tackle change for the customer’s competitive advantage. Even late in development, changes in requirements are welcomed to ensure the delivered software meets the evolving requirements of the customer.
* **Deliver Working Software Frequently:** This principle encourages the regular release of functional software increments in short iterations. This enables faster feedback and adaptation to changing requirements.
* **Collaboration between Business Stakeholders and Developers:**This says the businesspeople and developers must work together daily throughout the project. There should be communication and collaboration between stakeholders and the development team regularly. This is crucial for understanding and prioritizing requirements effectively.
* **Build Projects around Motivated Individuals**: This promotes in giving developers the environment and support they need and trusts them to complete the job successfully. Motivated and empowered individuals are more likely to produce work with quality and make valuable contributions to the project
* **Face-to-face communication is the Most Effective:** Face-to-face communication is the most effective method of discussion and conveying information. This principle depicts the importance of direct interaction which helps minimize misunderstandings, and hence effective communication is achieved.
* **Working Software is the Primary Measure of Progress:** This principle emphasizes delivering functional and working software as the primary metric for project advancement. It encourages teams to prioritize the continuous delivery of valuable features, so it ensures that good progress is consistently achieved throughout the process. The primary goal is to provide customers with incremental value and also gather feedback early in the project life cycle.
* **Maintain a Sustainable Pace of Work:** Agile promotes sustainable development. All people involved: The sponsors, developers, and users should be able to maintain a constant pace indefinitely. This principle depicts the need for a sustainable and consistent development pace. This helps in avoiding burnout and ensures long-term project success.
* **Continuous Attention to Technical Excellence and Good design:**This principle is on the importance of maintaining high standards of technical craft and design, so it ensures the long-term ability in maintenance and adaptability of the software.
* **Simplicity—the Art of Maximizing the Amount of Work Not Done:** Simplicity is essential. The objective here is to concentrate on the most valuable features and tasks and avoid unnecessary complexity as the art of maximizing the amount of work not done is crucial.
* **Self-Organizing Teams:** Self-organizing teams provide the best architectures, requirements, and designs. These help in empowering teams to make decisions and organize to optimize efficiency and creativity.
* **Regular Reflection on Team Effectiveness:**This makes the team reflect on how to become more effective at regular intervals and then adjust accordingly. Continuous improvement is very crucial for adapting to changing circumstances and optimizing the team’s performance over time.

1. **Explain working methodology of agile model and also write pros and cons.**

* Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.
* Agile Methods break the product into small incremental builds.
* These builds are provided in iterations.
* Each iteration typically lasts from about one to three weeks. Every iteration involves cross functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing.
* At the end of the iteration a working product is displayed to the customer and important stakeholders.

**Pros :**

* Is a very realistic approach to software development
* Promotes teamwork and cross training.
* Functionality can be developed rapidly and demonstrated.
* Resource requirements are minimum.
* Suitable for fixed or changing requirements
* Delivers early partial working solutions.
* Good model for environments that change steadily.
* Minimal rules, documentation easily employed.
* Enables concurrent development and delivery within an overall planned context.
* Little or no planning required Easy to manage Gives flexibility to developers
* Easy to manage
* Gives flexibility to developers

**Cons :**

* Not suitable for handling complex dependencies.
* More risk of sustainability, maintainability and extensibility.
* An overall plan, an agile leader and agile PM practice is a must without which it will not work.
* Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
* Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
* There is very high individual dependency, since there is minimum documentation generated.
* Transfer of technology to new team members may be quite challenging due to lack of documentation.

1. **Draw usecase on Online shopping product using cod.**

[**https://app.diagrams.net/#G1YGp8RS8TPvtMkiNETk6KV3UbxpBjHyaX#%7B%22pageId%22%3A%22exJhWMVv3bOAPERQqS0O%22%7D**](https://app.diagrams.net/#%7B%22pageId%22%3A%22exJhWMVv3bOAPERQqS0O%22%7D)

1. **Draw usecase on Online shopping product using payment gateway.**

[**https://app.diagrams.net/#G103Na\_NTsUPcrLktvT9LOdOxIoR1fIx\_L#%7B%22pageId%22%3A%22cymv5U-KASu0xyg6UA1n%22%7D**](https://app.diagrams.net/#%7B%22pageId%22%3A%22cymv5U-KASu0xyg6UA1n%22%7D)