1. **What is SDLC (Software Development Life Cycle)?**

The **Software Development Life Cycle (SDLC)** is a structured methodology used for planning, designing, developing, testing, and maintaining software. It provides a clear and systematic approach to software development, ensuring quality, and efficiency. SDLC defines the different phases involved in building software, from initial concept through to its final deployment.

**Phases of SDLC:**

1. **Requirement Gathering**: Collecting detailed and clear requirements from stakeholders.
2. **System Design**: Creating the architecture and design of the system based on requirements.
3. **Implementation**: Writing the actual code based on the design.
4. **Testing**: Verifying the software to ensure it works as intended and fixing any defects.
5. **Deployment**: Releasing the software to users.
6. **Maintenance**: Ongoing updates and bug fixes after deployment.

**2. What is Software Testing?**

**Software Testing** is the process of evaluating a software application to identify any defects, ensure that it meets the specified requirements, and confirm that it works as expected. Testing verifies the software’s functionality, reliability, performance, security, and usability.

There are two main types of software testing:

* **Manual Testing**: Testers manually execute test cases and report defects.
* **Automated Testing**: Using automated tools (e.g., Selenium, QTP) to perform testing tasks automatically.

**3. What is Agile Methodology?**

**Agile methodology** is a flexible and iterative approach to software development where software is developed incrementally in small units or **sprints** (usually 1-4 weeks). The Agile process involves continuous collaboration between developers, testers, and stakeholders, allowing for rapid feedback and changes in requirements.

**Key principles of Agile:**

* **Iterative Development**: Delivering software in smaller, manageable chunks.
* **Continuous Testing**: Testing is performed frequently throughout the development cycle.
* **Flexibility**: Agile welcomes changes, even late in development.
* **Collaboration**: Developers, testers, and business stakeholders work closely together.
* **Customer Feedback**: The customer is involved and their feedback is incorporated into the development.

**Popular Agile frameworks**: Scrum, Kanban, and Extreme Programming (XP).

**4. What is SRS (Software Requirements Specification)?**

An **SRS (Software Requirements Specification)** is a document that outlines the functional and non-functional requirements of a software application. It serves as a blueprint for the development team to understand what needs to be built and is used by testers to create test cases.

**Contents of an SRS:**

* **Functional Requirements**: What the system should do (e.g., user authentication, data processing).
* **Non-Functional Requirements**: How the system should perform (e.g., response time, security).
* **System Features**: Detailed description of each feature of the system.
* **Constraints**: Limitations or restrictions in terms of technology, time, or resources.

**5. What is OOPs (Object-Oriented Programming)?**

**OOP (Object-Oriented Programming)** is a programming paradigm based on the concept of **objects**, which represent real-world entities. OOP emphasizes the use of classes and objects to model data and the methods that operate on that data.

**Key OOP Concepts:**

* **Encapsulation**
* **Inheritance**
* **Polymorphism**
* **Abstraction**

**6. Write Basic Concepts of OOPs**

* **Encapsulation**: The bundling of data (attributes) and methods (functions) that operate on that data into a single unit, i.e., a class. It also involves hiding the internal state of objects and providing access only through public methods (getters/setters).
* **Inheritance**: A mechanism by which one class (subclass) inherits attributes and methods from another class (superclass). It allows for code reuse and hierarchical relationships between classes.
* **Polymorphism**: The ability of a single method or function to behave differently based on the context, such as a method taking different forms (method overriding or method overloading).
* **Abstraction**: The process of hiding complex implementation details and exposing only the essential features of the object or class to the user. It simplifies interaction with the system.

**7. What is Object?**

An **Object** is an instance of a class in Object-Oriented Programming. It represents a real-world entity and contains:

* **Attributes**: Data that describes the object.
* **Methods**: Functions or behaviors that the object can perform.

For example, consider a class Car. An object of the class Car might represent a specific car with attributes like color, model, and methods like start() or stop().

**8. What is Class?**

A **Class** is a blueprint or template for creating objects. It defines the properties (attributes) and behaviors (methods) that the objects of that class will have.

**9. What is Encapsulation?**

**Encapsulation** is the concept of bundling the data (attributes) and methods (functions) that operate on that data into a single unit, the class. It also involves hiding the internal workings of an object and exposing only the necessary functionality to the outside world through methods.

**Advantages of Encapsulation:**

* **Data Hiding**: Internal details of an object are hidden from the outside world.
* **Control**: Access to data can be controlled via getter and setter methods.
* **Flexibility**: Changes to internal data representations do not affect other parts of the program as long as the interface remains the same.

**10. What is Inheritance?**

**Inheritance** is a mechanism where a new class (subclass or child class) can inherit attributes and methods from an existing class (superclass or parent class). It allows code reuse and can establish a hierarchical relationship between classes.

**11. What is Polymorphism?**

**Polymorphism** is the ability of a single method, function, or operator to operate on different types of objects or take on multiple forms. It allows objects of different classes to be treated as objects of a common superclass, and each class can provide its own implementation of a method.

**Types of Polymorphism:**

* **Method Overloading**: Same method name, but with different parameters.
* **Method Overriding**: A subclass provides a specific implementation of a method already defined in its superclass.

**12. What are the SDLC phases with a basic introduction?**

The **Software Development Life Cycle (SDLC)** is a systematic process for software development that helps in delivering high-quality software while ensuring it meets the client's requirements. It involves various phases, each with its specific tasks. Here are the typical SDLC phases:

**Phases of SDLC:**

1. **Requirement Gathering and Analysis:**
   * The first phase involves gathering all the requirements from stakeholders and understanding the needs of the system. A document known as the Software Requirements Specification (SRS) is created to define the requirements.
2. **System Design:**
   * This phase translates the requirements into a blueprint for building the software. It includes both high-level and detailed design of the system’s architecture and components.
3. **Implementation (Coding):**
   * In this phase, developers begin writing the actual code based on the design documents. This phase also includes integration of various modules.
4. **Testing:**
   * After development, the system undergoes rigorous testing to ensure it works as expected. This phase involves various types of testing, such as unit testing, integration testing, and acceptance testing.
5. **Deployment:**
   * Once testing is completed, the software is deployed to the production environment. It is either released to end-users or made available for beta testing.
6. **Maintenance:**
   * After deployment, the software enters the maintenance phase, where it is monitored for bugs, and necessary updates or changes are made to improve performance and usability.

**13. Explain the phases of the Waterfall model.**

The **Waterfall Model** is a traditional, linear approach to software development. Each phase is completed sequentially, meaning one phase must be finished before the next begins. This model works best for projects where the requirements are well understood upfront and unlikely to change during the development process.

**Phases of the Waterfall Model:**

1. **Requirement Analysis:**
   * All system requirements are collected and documented in detail at the start of the project.
2. **System Design:**
   * The system’s architecture and design are planned based on the requirements. Detailed design specifications are created.
3. **Implementation (Coding):**
   * Developers start coding based on the design specifications.
4. **Integration and Testing:**
   * Once the system is developed, it is integrated and tested to identify and fix any issues.
5. **Deployment:**
   * After testing, the software is deployed in the production environment for use by end-users.
6. **Maintenance:**
   * After deployment, the software enters the maintenance phase where bugs are fixed, and any necessary updates are made.

**14. What are the phases of the Spiral model?**

The **Spiral Model** combines the features of both the Waterfall and Prototyping models. It focuses on iterative development and risk analysis throughout the life cycle. The project is divided into smaller, manageable parts (called "spirals") and each iteration produces a working product.

**Phases of the Spiral Model:**

1. **Planning (Objectives Setting):**
   * Initial project objectives are defined, and requirements are gathered. Preliminary designs are created, and potential risks are identified.
2. **Risk Analysis:**
   * This phase focuses on assessing risks, such as technical risks or business risks, and creating risk mitigation strategies. Prototypes may be developed to handle unknowns.
3. **Engineering (Development and Testing):**
   * The system is developed and tested based on the defined requirements. This includes coding, designing, and testing the working model.
4. **Evaluation (Customer Feedback):**
   * After each spiral iteration, the product is reviewed by the stakeholders or customers for feedback, which is used to improve the product in the next iteration.
5. **Planning for the Next Iteration:**
   * Based on the feedback from the customer, planning for the next iteration of the cycle is done. This helps in refining the system and minimizing risks.

**15. What are the Agile manifesto principles?**

The **Agile Manifesto** was created in 2001 by a group of software developers who were seeking a better, more efficient way to build software. The manifesto emphasizes the values of collaboration, flexibility, and customer satisfaction, while minimizing rigid processes and extensive documentation.

**The Four Core Values of the Agile Manifesto:**

1. **Individuals and interactions over processes and tools:**
   * Agile emphasizes the importance of people and communication over following strict processes or using specific tools.
2. **Working software over comprehensive documentation:**
   * Agile values delivering working software that provides value to the customer, instead of spending too much time on documentation.
3. **Customer collaboration over contract negotiation:**
   * Agile promotes working closely with the customer and adapting to their feedback throughout the project rather than sticking to contract terms rigidly.
4. **Responding to change over following a plan:**
   * Agile is flexible and embraces changes, even late in the development cycle, rather than rigidly sticking to a pre-defined plan.

**16. Explain the working methodology of the Agile model and write its pros and cons.**

The **Agile model** follows an iterative and incremental approach to software development. The software is developed in small, manageable increments called **sprints** (typically 1 to 4 weeks long). After each sprint, a working version of the software is delivered, and feedback is gathered from stakeholders or end-users. This feedback is then incorporated into the next iteration.

**Working Methodology of the Agile Model:**

1. **Sprint Planning:**
   * At the start of each sprint, the team identifies the tasks to be completed and sets goals for that sprint.
2. **Daily Stand-ups:**
   * A daily meeting is held where team members discuss what they did the previous day, what they plan to do today, and any blockers they might have.
3. **Sprint Execution:**
   * The team works on completing the tasks defined during the sprint planning phase. This involves coding, testing, and reviewing.
4. **Sprint Review:**
   * At the end of each sprint, the team demonstrates the working product to the stakeholders, collects feedback, and determines improvements for the next sprint.
5. **Sprint Retrospective:**
   * The team reviews how the sprint went and discusses areas of improvement for the next sprint.

**Pros of Agile:**

1. **Flexibility and Adaptability:**
   * Agile allows changes even after the development has started. This is useful when business requirements evolve over time.
2. **Faster Time to Market:**
   * Agile delivers working software frequently, allowing the customer to start using the software sooner and providing quicker feedback.
3. **Customer Collaboration:**
   * Agile involves the customer throughout the development process, ensuring the product aligns with their needs and expectations.
4. **Improved Quality:**
   * Continuous testing, reviews, and feedback help detect and fix issues early, improving the overall quality of the software.
5. **Better Risk Management:**
   * Regular iterations help identify risks early, and the iterative nature of Agile allows for risk mitigation strategies.

**Cons of Agile:**

1. **Requires Experienced Team Members:**
   * Agile requires skilled team members who are capable of managing tasks autonomously and adapting to changes quickly.
2. **Can Lead to Scope Creep:**
   * Since Agile embraces changes and constant feedback, it can sometimes lead to continuous changes in the project’s scope, which can impact timelines and budget.
3. **Resource Intensive:**
   * Agile requires a high level of collaboration and communication, which can be resource-intensive, especially in larger teams.
4. **Difficult to Scale for Large Projects:**
   * Agile works well for small to medium-sized projects, but it may be harder to scale for large, complex projects with multiple teams.

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

<https://drive.google.com/file/d/1cCwWjS-2UhaeNmjov_zeF9ePMtgE6TGu/view?usp=drive_link>

**18. Draw Usecase on banking system customers.**

[**https://drive.google.com/file/d/1vj9LEGy41YLvtBR7Fb8M6waCAHMvSLyR/view**](https://drive.google.com/file/d/1vj9LEGy41YLvtBR7Fb8M6waCAHMvSLyR/view)

**19. Draw Usecase on Broadcasting System.**

<https://drive.google.com/file/d/1upYybCJtb_YLv8pUr-K4pr2DWSa1J8hJ/view>

**20. Draw usecase on OTT Platform.**

<https://drive.google.com/file/d/1jEUAqm6xFqVXiHmxXrsF0mvQACioBuIN/view?usp=drive_link>

**21. Draw usecase on E-commerce application.**

<https://drive.google.com/file/d/1qPixSsTDkN8Wlvy5S7JKLLM-dQJFjcmz/view?usp=drive_link>

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

<https://drive.google.com/file/d/1DG3ypkmZB2_rggNtGva9khLGxlIHi4wM/view?usp=drive_link>