Unit 1: Introduction to Software Development

Exercise Short Answer Questions.

1. **Differentiate between functional and non-functional requirements.**

Difference between Functional and Non Functional Requirements:	
Functional Requirements	**Non-Functional Requirements**
Define specific behaviors or functions of the system	Define the quality attributes and constrai -nts of the system
What the system should do	How the system should perform
Example: User can borrow books	Example: System should handle 1000 users simultaneously
Directly related to user interactions and system tasks	Related to system performance, usability, reliability, etc.

2. **Explain why the testing phase is important in the Software Development Life Cycle (SDLC), and provide two reasons for its significance.**
   Once the software has been developed, it undergoes a crucial phase called testing. Testing is the process of checking the software to identify any bugs, errors, or issues. Think of it as a quality check to make sure everything works as expected. During testing, the software is run under various conditions to see if it behaves correctly.
   - Testing helps in identifying any hidden issues that were not apparent during development.
   - By fixing these issues, developers ensure that the software runs smoothly and meets the user's needs, providing a better and more reliable experience.

3. **Illustrate the concept of continuous integration in Agile Methodology and discuss its importance in software development.**
   Continuous Integration: Regularly merging code changes into a central repository to detect and fix issues early.
   Continuous Integration (CI) is a key concept in **Agile methodology** that means developers regularly add their code to a shared project often many times a day. Each time someone adds new code, the system automatically checks to make sure everything still works by running tests. This helps catch errors early, before they become big problems.
   CI is important in software development because it makes the process faster, more reliable, and less stressful. Instead of waiting weeks or months to combine everyone's work, CI allows small, frequent updates that are easier to test and fix. It also encourages teamwork, as all developers work together on the same codebase. In Agile, where changes happen quickly and often, CI helps teams deliver better software faster and with fewer bugs.

4. **Identify the key components of the Scrum framework and analyze how each contributes to effective project management.**
   Key Components of Scrum:
   - **Roles:** The primary roles in Scrum are the Product Owner, Scrum Master, and Development Team. The

Product Owner defines the product backlog and ensures that the team is working on the highest-priority items. The Scrum Master facilitates the process, removes obstacles, and ensures that the team follows Scrum practices. The Development Team, consisting of cross-functional members, is responsible for delivering the product increments.

- **Events:** Scrum employs a series of events to ensure regular progress and review. These include Sprints (time-boxed iterations), Sprint Planning, Daily Standups, Sprint Reviews, and Sprint Retrospectives.
- **Artifacts:** Key artifacts in Scrum are the Product Backlog (a prioritized list of features and requirements), Sprint Backlog (a list of tasks to be completed in a Sprint), and Increment (the working product that is the result of the current Sprint).

5. Evaluate the main steps involved in risk assessment and management and assess their importance in a software project.

Steps in Risk Assessment and Management:

1. **Identify Risks:** List all potential risks that could affect the project. These could be technical risks, such as technology changes; operational risks, like resource shortages; or external risks, such as market fluctuations.
2. **Analyze Risks:** Evaluate the likelihood of each risk occurring and its potential impact on the project. This helps in prioritizing which risks need more attention.
3. **Develop Mitigation Strategies:** For each significant risk, develop a plan to reduce its likelihood or minimize its impact. This could involve adding buffers to the schedule, securing backup resources, or conducting additional testing.
4. **Monitor and Review:** Continuously monitor the project for new risks and review existing risks to adjust strategies as necessary.

Example: A project is using a new, untested technology. The risk is that the technology may not work as expected, causing delays and additional costs.

6. Explain the purpose of a Use Case Diagram in software development.

Use Case Diagrams are used for several purposes:

1. Capturing Functional Requirements: They help in identifying and documenting the functional requirements of the system.
2. Understanding User Interactions: They illustrate how different users will interact with the system.
3. Planning and Testing: They aid in planning the development process and in designing test cases for validating system functionalities.

7. Compare and contrast a Sequence Diagram with an Activity Diagram, highlighting the key differences.

Sequence Diagrams:

Sequence Diagrams show how objects in a system interact with each other in a particular sequence. They help in understanding the flow of messages between objects over time.

Interactions:

open(): User opens each box.
put toys/books/clothes inside: User puts the respective items into the boxes.
close(): User closes each box.

Activity Diagrams illustrate the flow of activities or steps in a process. They are useful for modeling the logic of complex operations.
Example: In a restaurant management system, an activity diagram can represent the process from 'Order Placement' to 'Food Preparation' and finally to 'Order Delivery'.

Feature	Sequence Diagram	Activity Diagram
Purpose	Shows how objects interact step-by-step.	Shows the flow of actions in a process.
Focus	Order of messages between objects.	Workflow and decision-making steps.
Time	Time flows top to bottom.	Time is not shown, only control flow.
Main Elements	Objects, messages, lifelines.	Activities, arrows, decision nodes.
Parallel Actions	Limited support.	Supports parallel flows easily.
Best Used For	Detailing object communication.	Describing a complete process or logic flow.

8. Describe the Factory Pattern and explain how it differs from directly creating objects, with an example.

Factory Pattern:

The Factory Design Pattern is like having a special workshop that knows how to create different products, but you don't need to worry about the details of how those products are made. Instead, you just tell the factory what you need, and it gives you the finished product.
Example:
Imagine you are building a software for a **vehicle company**. The company makes **Cars** and **Bikes**.

We can create objects like this:

Car car = new Car();

Bike bike = new Bike();

But what if the company adds more vehicles like **Truck**, **Bus**, etc.? You'd have to change your code everywhere.

Using Factory Pattern:

Instead, you can use a **VehicleFactory** that creates the vehicle for you.

VehicleFactory.getVehicle("Car") → returns a Car object

VehicleFactory.getVehicle("Bike") → returns a Bike object

Now you just ask the factory, and it gives you the object — you **don't need to know how it's created**.