2021 spring OOSD

Date
Page

1a OOAD with example

→ OOAD stands for Object Oriented Analysis & Design. It is a methodology used in software development to model, design & create systems using the principles of object-oriented programming. OOAD focuses on understanding real-world entities, their interactions & their behaviour & then translating this understandy into a well-organized software architecture

eg:. Library Mgmt System

* OOA
- during this phase, the focus is on understanding the requirements & identifying the main objects & their relationships.

• Objects
- Objects might include Book, Member & Library each book has attributes like name, membership/contact.

• Relationships
- Books are associated with the library & can be checked out by members. Has-many, has-one.

* OOD
- during this phase, the focus is on creating the actual software architecture based on the analysis.

• Classes
- Classes are designed based on the identified

objects. For eg there might be a 'Book' class with attributes (title, author) & methods checkOut, returnBook)

- Inheritance.
  - we can create a 'StudentMember' class & an 'EmployeeMember' class both inherit from the Member class
- Encapsulation.
  - each classes encapsulates its data, attributes & exposes well-defined methods for interaction.

Q.B

→ UML or Unified Modeling Language is a standardized graphical language used in software engineering to visualize, specify, construct, & document various aspects of a software-intensive system. It provides a set of notations & guidelines for creating visual models that help stakeholders understand & communicate about the system's structure, behavior & interactions.

→ Building blocks of UML

(1) Structural Diagrams
   a. Class diagram
   b. Object diagram

(10) Behavioral Diagram
a. Use Case diagram
b. Sequence Diagram
c. Activity Diagram

(11) Interaction Diagrams
a. Communication diagram
b. Timing diagram.

eg:- Use case for only shopping system

- Actors: Customer, Admin
- Use cases: Browse Products
  Add to Cart
  Remove from cart
  Checkout
  Manage Products (Admin)
  Manage orders (Admin)

Date
Page

Q2 a

→ The iterative & incremental model is ideal for software development projects where requirements are not well-defined upfront & may evolve over time. It suits projects that require flexibility, adaptability, & the ability to accommodate changes as they occur.

→ Phases in iterative & incremental model

(i) Requirements Gathering
(ii) Iteration 1
(iii) Iteration 2
(iv) Iteration N
(v) Incremental Development
(vi) Integration & Testing
(vii) Feedback & Refinement
(viii) Completion & Deployment

→ It is used when:-

(i) Project is evolving & under requirements
(ii) Frequent changes in scope or technology
(iii) A need for rapid delivery of partial functionality
(iv) Focus on risk management & defect detection

Date
Page

Q2h)

→ 4'p of software development

(i) People

- People are the cornerstone of any software development endeavor.
- Collaborative teamwork, effective communication & a positive working environment are crucial for successful software development.

(ii) Product

- refers to the software system being developed
- Focus on understanding & satisfying user requirements, as well as delivering a product that adds value & solves real-world problem.

(iii) Process

- refers to the methodology & approach used for software development.
- includes the set of activities, practices & workflows that guide the project from initiation to completion

(iv) Project

- The project encompasses the management aspects of software development, including planning, scheduling, budgeting & risk management.

Q3a

→ The UP (Unified Process) is a popular iterative & incremental software development methodology that provides a structured approach to creating high-quality software systems. It emphasizes collaboration among team members, flexibility in accomodating changing requirements & a focus on producing effective software solution. UP is often associated with the UML.

→ Core workflow of Unified Process

1. Requirement workflow
   - In this workflow, the team identifies, gathers, & analyzes the requirements of the software system. This involves understanding user needs, defining functional & non-functional requirements & prioritizing features.

2. Analysis & Design workflow
   - This workflow focuses on creating a design for the software system based on the requirements. It involves creating models, specifying system architecture, & designing user interfaces.

3. Implementation workflow
- In the workflow, the software system is actualy implemented based on the design specifications. Code is written, modules are integrated & unit tests is conducted.

4. Testing workflow
- This workflow involves comprehensive testing of the software system to identify & fix defects, ensure the system's functionality & reliability & performance.

Q3b

→ A design pattern is a reusable & proven solution to a recurring problem that arises during software design & development. It is a general, well-structured approach that addresses specific design challenges & provides a blueprint for solving similar problems in different contexts.

Design patterns capture best practices & design principles, helping developers create efficient, maintainable, & scalable software solutions.

Date
Page

Importance of Design Patterns

1. Reusable Solutions
- Design patterns provide tried-and-tested solutions to common design problems.

2. Consistency:
- Design patterns promote consistency in software design. By using established patterns, teams can create a unified & coherent design that is easier to understand & maintain

3. Scalability
- Design patterns help create modular & loosely coupled designs. This scalability allows for easier addition of new features, components or functionalities.

4. Flexibility & Adaptability:
- Design patterns make software more adaptable to changes.

5. Efficiency:
- Design patterns optimize software design, leading to better performance & resource utilization

6. Problem Solving

7. Quality Assurance.

Date
Page

Q9

→ In OOSD, the design pattern used to provide object creation mechanism is the "Creational Design Pattern".

Creational design patterns deal with the process of object instantiation, encapsulating the object creation process, & making it more flexible, efficient & manageable.

→ There are 5 well-known creational design patterns.

(i) Singleton Pattern
(ii) Factory Method Pattern
(iii) Abstract Factory Pattern
(iv) Builder Pattern
(v) Prototype Pattern.

(i) Singleton Pattern.
- ensures a class has only one instance & provides a global point of access to that instance.
- Applicability: When you want to ensure that a class has only one instance

(ii) Factory Method Pattern.
- purpose: defines an interface for creating objects, but lets subclasses decide which class to instantiate.

Applicability: when a class cannot anticipate the type of objects it must create or when a class wants its subclasses to specify the objects it creates

3. Abstract Factory Pattern.
- Purpose – Provides an interface for creating families of related or dependent objects without specifying their concrete classes.
- Applicability: when you need to create families of related objects or when you want to ensure a system is independent of how its objects are created, composed & represented.

4. Builder Pattern
- Purpose: Separates the construction of a complex object from its representation, allowing the same construction process to create different representations.
- Applicability: when a complex object needs to be created with a step-by-step approach or when multiple representations of the object are required.

Q45

→ Concurrency patterns are design solutions that address the challenges & complexities associated with managing & coordinating the execution of multiple concurrent threads are processes in a software application. Concurrency is the ability of a system to handle multiple tasks or operations simultaneously, & concurrency patterns help developers ensure correct, efficient, & manageable concurrent execution.

Concurrency patterns are essential in scenarios where multiple threads or processes need to work together, share resources, & avoid conflicts.

Below is few commonly used concurrency patterns:

1. Mutex (Mutual Exclusion)
- ensures that only one thread can access a critical section of code or a shared resource at a time.

2. Semaphore
- controls access to a resource by maintaining a count of the number of threads that can access it simultaneously.

Date
Page

3. Reader-writer lock
- Allows multiple threads to read a shared resource simultaneously, but only one thread can write to the resource at a time.

4. Producer - Consumer
- coordinates the interaction between producer threads & consumer threads to avoid issues like overproduction or resource exhaustion.

5. Barrier.
- synchronizes a group of threads by forcing them to wait until all threads have reached a certain point before proceeding.

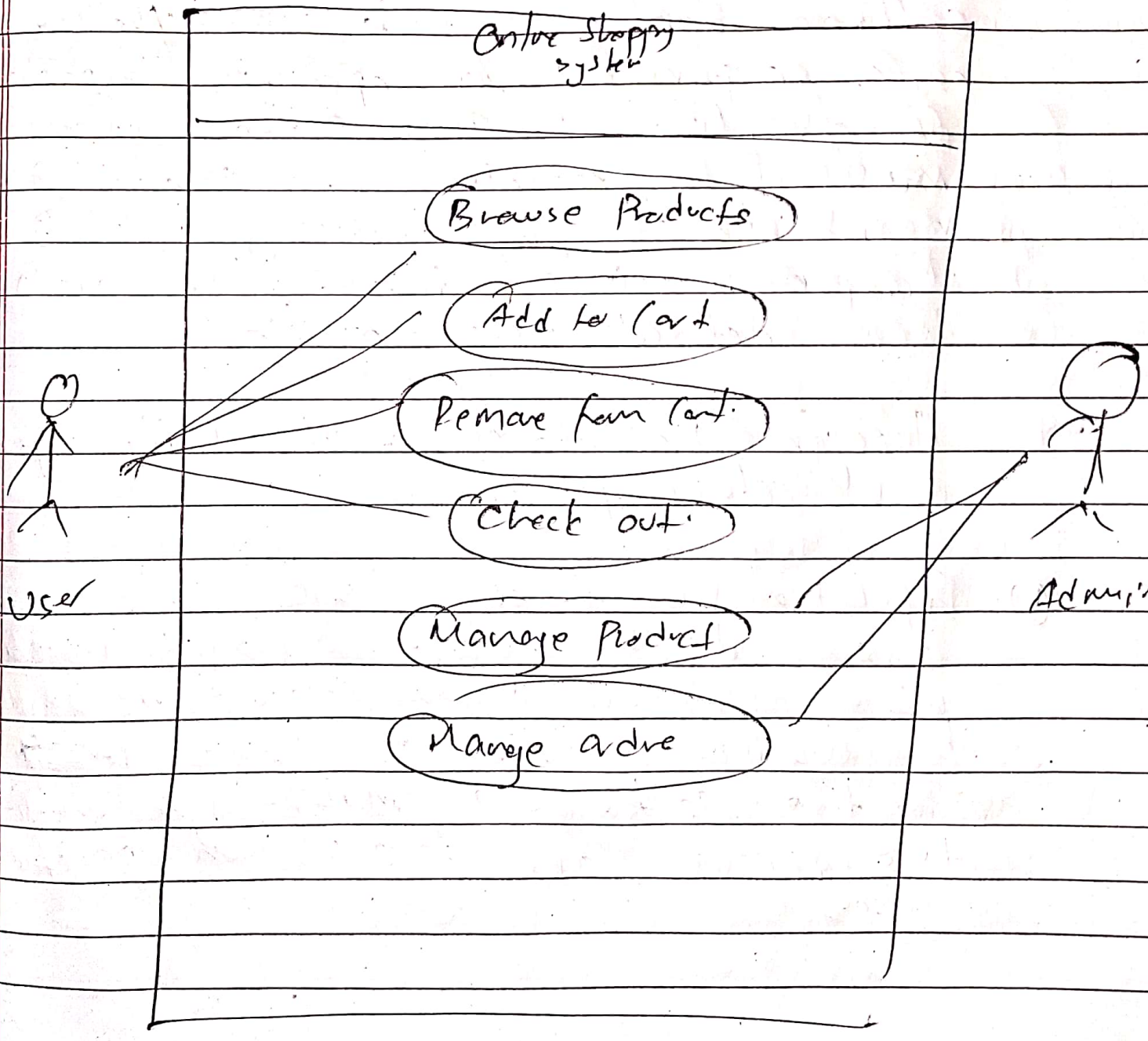
Q 5
→ Software architecture refers to the high-level structure & organization of a software system. It defines the components, relationships, interactions & design decisions that guide the development & evolution of the software. A well designed software architecture.

Date
Page

provides a blueprint for buildy a system that meets its functional & non-functional requirements while ensuring scalability, maintainability & robustness.

→ eg.- online shopping system

1. Use Case Diagram.



Date
Page

MVC architectural pattern.
Done in lab report.

The multi-layer architectural pattern, is a design approach that devides a software application into multiple layers, each responsible for specific aspects of functionality. This pattern promotes modularity, separation of concern & maintainability by organizing code & components into distinct layer with well defined responsibilities.

Design principles of the multi-layer architectural pattern:

- Separation of Concerns (SoC)
  It dictates that different aspects of a software system should be handled by distinct modules or layer. In the context of multi-layer architecture, each layer focuses on specific concerns.

(ii) Modularity
- each layer is a self-contained module with a well-defined interface. This modularity allows for easy maintenance, testing, & changes to individual layers without affecting the entire system.

(iii) Reusability
- By isolating specific functionality within layers, components can be reused across different parts of the application or even in other projects.

(iv) Maintainability
- Changes or updates in one layer do not necessarily impact other layers, making maintenance & updates more manageable

(v) Scalability.
- Components can be scaled independently.

Q7a: Behavioral Design Pattern

- Behavioral design patterns are a subset of design patterns that focus on the interaction & communication between objects to achieve certain behaviors & responsibilities with a software system. These patterns address the dynamic aspects of software design, emphasizing how objects collaborate, delegate tasks & manage behavior

Common behavioral design patterns include:

(i) Observer Pattern
(ii) Strategy Pattern
(iii) Command Pattern
(iv) Mediator Pattern.
(v) Template method Pattern.

b. Life Cycle of Unified Software development Process

- The unified software Development Process (USDP) is a comprehensive framework for software development that encompasses various phases & workflows

→ phases:-

(i) Inception phase
(ii) Elaboration phase
(iii) Construction phase
(iv) Transition phase.