Software Engineering (IAE - 1)

▼ **Table of Contents**

2-Mark Questions

Q1. List down golden rules of user interface design.

Ben Shneiderman's Eight Golden Rules of Interface Design:

1. **Strive for Consistency** - Use consistent sequences of actions, terminology, colors, layout, and fonts throughout the interface
2. **Seek Universal Usability** - Design for diverse users including novices, experts, different age groups, and disabilities
3. **Offer Informative Feedback** - Provide appropriate feedback for every user action, modest for frequent actions and substantial for major actions
4. **Design Dialogs to Yield Closure** - Organize actions into clear beginning, middle, and end sequences
5. **Prevent Errors** - Design to prevent serious errors and provide simple error handling mechanisms

6. **Permit Easy Reversal of Actions** - Support undo functionality to reduce user anxiety and encourage exploration
7. **Support Internal Locus of Control** - Make users feel in control by responding to their actions predictably
8. **Reduce Short-term Memory Load** - Keep displays simple and provide online help when needed

Q2. What is feasibility study? What are the types of feasibility study?

Feasibility Study is an evaluation process that analyzes whether a proposed software project is practical and viable before actual development begins.

Types of Feasibility Study:

1. **Technical Feasibility** - Assesses available hardware, software, and technical resources
2. **Economic Feasibility** - Analyzes cost-benefit ratio and financial viability
3. **Operational Feasibility** - Evaluates ease of operation and maintenance post-deployment
4. **Legal Feasibility** - Investigates compliance with legal requirements and regulations
5. **Schedule Feasibility** - Determines if the project can be completed within given timeframes

Q3. Write short note on CMM levels.

CMM (Capability Maturity Model) is a framework developed by Carnegie Mellon University to assess and improve software development processes in organizations.
Purpose: CMM provides a framework for organizations to assess their current process maturity and identify areas for improvement to achieve higher quality software development.

The Five CMM Levels:

1. **Level 1 - Initial (Chaotic):**
    - Processes are ad hoc and unpredictable
    - Success depends on individual heroics; no formal processes
2. **Level 2 - Repeatable (Disciplined):**
    - Basic project management processes established
    - Requirements management and project tracking implemented
3. **Level 3 - Defined (Standardized):**
    - Processes documented, standardized, and integrated organization-wide
    - Well-characterized and understood development procedures
4. **Level 4 - Managed (Quantitative):**

- Detailed process metrics collected and analyzed
- Product and process quality quantitatively controlled

5. **Level 5 - Optimizing (Continuous Improvement):**
    - Continuous process improvement through quantitative feedback
    - Focus on defect prevention and innovative practices

Q4. Differentiate Prescriptive and Evolutionary models.

Aspect	Prescriptive Models	Evolutionary Models
Approach	Sequential, predefined phases	Iterative and incremental development
Requirements	Requirements fully defined upfront	Requirements evolve over iterations
Flexibility	Less flexible to changes	Highly adaptable to changing requirements
Examples	Waterfall, V-Model	Spiral, Prototyping
Feedback	Limited early feedback	Continuous user feedback
Risk	Higher risk due to late testing	Lower risk through iterative refinement

Q5. Describe advantages and limitations of large sized software projects.

Advantages:

- Enhanced productivity through specialized teams
- Ability to handle complex, large-scale problems
- Resource pooling and expertise sharing
- Faster development through parallel work streams
- Comprehensive testing capabilities

Limitations:

- **Communication Challenges** - Complex coordination between team members
- **Reduced Individual Ownership** - Difficulty in maintaining accountability
- **Increased Complexity** - Higher management overhead
- **Cost Overruns** - Budget management becomes difficult
- **Schedule Delays** - Coordination delays affect timelines

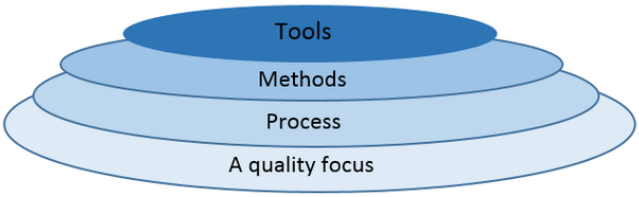
- **Quality Control Issues** - Maintaining consistent standards across teams

Q6. Describe the characteristics and nature of software and explain the layered structure of software engineering.

Software Characteristics:

- **Intangible** - Cannot be physically touched
- **Logical** - Consists of instructions and algorithms
- **Complex** - Involves intricate relationships between components
- **Maintainable** - Requires continuous updates and modifications

Layered Structure of Software Engineering:



Four-Layer Technology:

1. **Quality Focus (Foundation)** - Continuous process improvement, integrity, maintainability, and usability
2. **Process Layer** - Framework binding all layers; defines activities, actions, and tasks for software development
3. **Methods Layer** - Technical approaches including communication, planning, modeling, construction, and deployment
4. **Tools Layer** - Automated support for process and methods including CASE tools and development environments

Q7. Write short note on Scrum.

Scrum is an agile framework for managing software development projects that emphasizes teamwork, iterative development, and continuous improvement.

Key Components:

Scrum Roles:

- **Scrum Master** - Facilitates the process and removes impediments
- **Product Owner** - Defines requirements and priorities
- **Development Team** - Self-organizing team that delivers the product

Scrum Events:

- **Sprint** - Time-boxed iteration (2-4 weeks)
- **Sprint Planning** - Plan work for upcoming sprint
- **Daily Scrum** - 15-minute daily synchronization meeting
- **Sprint Review** - Demonstrate completed work
- **Sprint Retrospective** - Reflect and improve processes

Q8. Explain elements of Analysis model.

Analysis Model Elements:

Four Primary Elements:

1. **Scenario-based Elements**
   - Use cases and user stories
   - Represent system from user's perspective
2. **Class-based Elements**
   - Class diagrams and collaboration diagrams
   - Define objects, attributes, and relationships
3. **Behavioral Elements**
   - State diagrams and sequence diagrams
   - Show system state changes and event responses
4. **Flow-oriented Elements**
   - Data Flow Diagrams (DFD) and Control Flow Diagrams
   - Show data transformation through system functions

Core Components:

- **Data Dictionary** - Repository of all data objects
- **Entity Relationship Diagram (ERD)** - Depicts data relationships
- **Process Specifications (PSPEC)** - Describes function details

Q9. Differentiate Agile & Evolutionary process models.

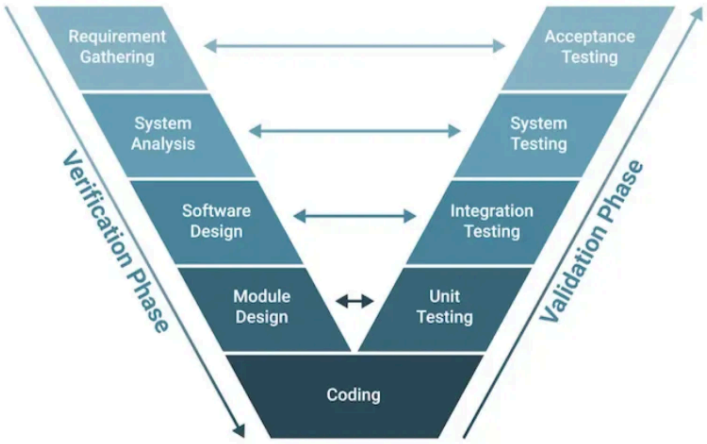
Aspect	Agile Models	Evolutionary Models
Philosophy	Values and principles focused	Iterative refinement focused
Time Management	Fixed time-boxed iterations	Variable iteration lengths
Customer Involvement	Continuous collaboration	Periodic feedback
Documentation	Minimal, working software preferred	Moderate documentation
Team Structure	Self-organizing, cross-functional	Traditional hierarchy possible
Change Management	Embraces change	Adapts to change

Q10. What is V model? Draw its diagram.

V-Model is a sequential software development model that extends the waterfall model by associating each development phase with a corresponding testing phase.

Key Characteristics:

- Also known as **Verification and Validation Model**
- Left side represents **Verification phases** (development)
- Right side represents **Validation phases** (testing)
- **Coding phase** connects both sides at the bottom



Q11. What are 3Ps in Software Project spectrum?

The 3Ps in Software Project Management:

1. **People**

- Most important component of software development
- Includes project managers, team leaders, stakeholders, analysts, and IT professionals
- Requires proper team management and role definition

2. **Product**
    - The deliverable or result of the project
    - Includes project objectives, benefits, outcomes, and deliverables
    - Defines scope and technical requirements
3. **Process**
    - Framework that regulates development approach
    - Includes documentation, implementation, deployment, and interaction phases
    - Key to successful product delivery

Note: Some sources refer to 4Ps, adding **Project** as the fourth P, which serves as the blueprint combining all other elements.

5-Mark Questions

Q1. Explain design principles and concepts.

Software Design Principles:

Fundamental Design Principles:

1. **Separation of Concerns**
    - Divide complex problems into manageable parts
    - Each component should handle specific functionality
    - Reduces complexity and improves maintainability
2. **Modularity**
    - Break system into independent, interchangeable modules
    - Enables parallel development and easier testing
    - Facilitates reusability and maintenance
3. **Abstraction**
    - Hide implementation details while exposing necessary interfaces
    - Reduces complexity for users of the module
    - Enables focus on essential characteristics

4. **Encapsulation**
    - Bundle data and methods together
    - Restrict direct access to internal components
    - Promotes data integrity and security

Design Concepts:

Cohesion and Coupling:

- **High Cohesion** - Elements within module work together for single purpose
- **Low Coupling** - Minimize dependencies between modules
- Leads to maintainable and scalable systems

Design Patterns Categories:

1. **Creational Patterns** - Focus on object creation (Factory, Singleton, Builder)
2. **Structural Patterns** - Deal with object composition (Adapter, Facade, Decorator)
3. **Behavioral Patterns** - Focus on communication between objects (Observer, Strategy, Template Method)

Q2. Explain the characteristics of SRS. Prepare SRS for developing software for online course registration system.

Characteristics of SRS:

Essential SRS Characteristics:

1. **Correctness** - All requirements accurately reflect user needs
2. **Completeness** - Includes all functional and non-functional requirements
3. **Consistency** - No conflicting requirements or contradictions
4. **Unambiguous** - Each requirement has only one interpretation
5. **Verifiable** - Requirements can be tested and validated
6. **Modifiable** - Can be updated as project evolves
7. **Traceable** - Links to other documents and design elements
8. **Ranked** - Requirements prioritized by importance
9. **Feasible** - Technically and economically achievable

SRS for Online Course Registration System:

1. Introduction

- **Purpose:** Automate course registration process for educational institutions

- **Scope:** Web-based system for students, faculty, and administrators
- **Definitions:** Student - enrolled learner, Course - academic subject offering

2. Overall Description

- **Product Perspective:** Standalone web application with database integration
- **Product Functions:** User authentication, course browsing, registration, schedule management
- **User Classes:** Students, Faculty, Administrators, Academic Staff
- **Operating Environment:** Web browsers, database server, application server

3. Functional Requirements

- **FR1:** System shall allow student login with credentials
- **FR2:** Students shall view available courses with details
- **FR3:** System shall process course registration requests
- **FR4:** Faculty shall update course information and capacity
- **FR5:** Administrators shall generate enrollment reports

4. Non-Functional Requirements

- **Performance:** Handle 1000 concurrent users
- **Security:** Encrypted data transmission and storage
- **Usability:** Intuitive interface accessible on mobile devices
- **Reliability:** 99.9% uptime during registration periods

5. External Interface Requirements

- **User Interface:** Responsive web interface
- **Hardware Interface:** Standard web server configuration
- **Software Interface:** Database connectivity, email service integration
- **Communication Interface:** HTTPS protocol support

Q3. Explain Design model & Draw Deployment diagram/Swimlane diagram for online shopping.

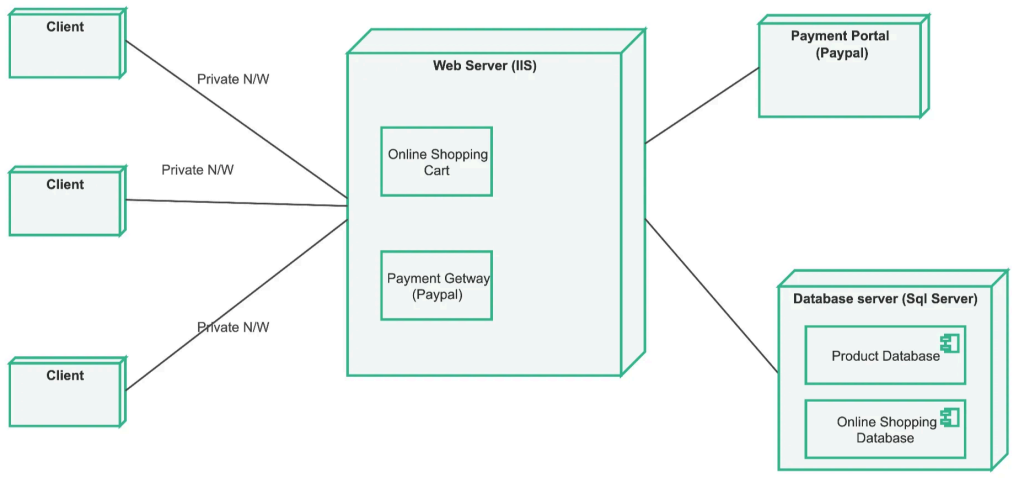
Design Model Components:

Design Model Elements:

- **Architectural Design** - Overall system structure and components
- **Interface Design** - User interaction mechanisms

- **Component Design** - Individual module specifications
- **Data Design** - Database structure and relationships
- **Deployment Design** - Physical system architecture

Deployment Diagram for Online Shopping System:
A deployment diagram shows the physical architecture and how software components are distributed across hardware nodes.



Hardware Nodes:

- **Client Devices** - Customer browsers, mobile apps
- **Web Server (IIS)** - Application hosting environment
- **Database Server (SQL Server)** - Data storage and management
- **Payment Portal (PayPal)** - Secure payment processing

Software Components:

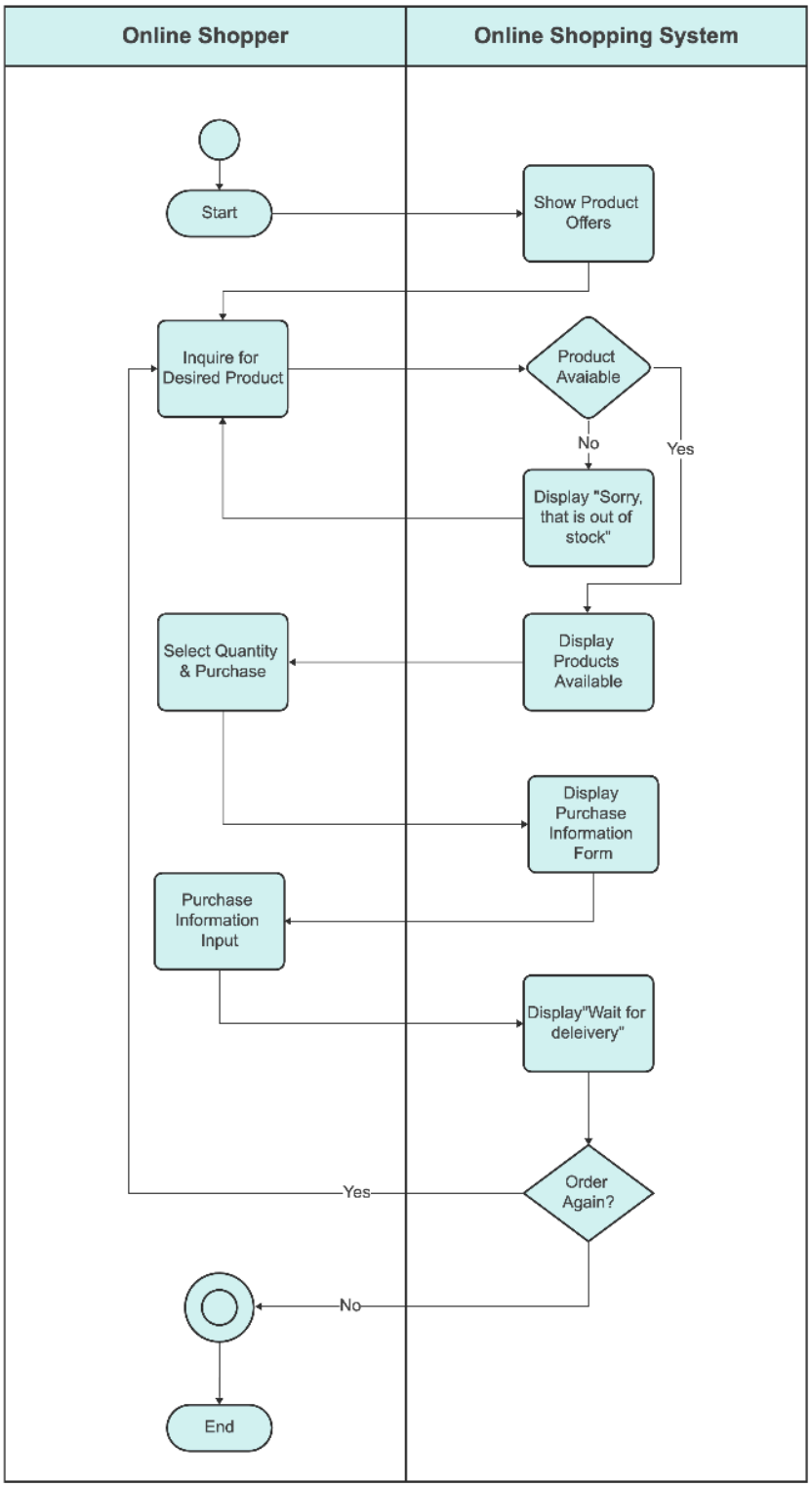
- **Online Shopping Cart** - Session management and cart functionality
- **Payment Gateway (PayPal)** - Payment processing integration
- **Product Database** - Product catalog and inventory
- **Online Shopping Database** - Customer and order data

Network Connections:

- **Private Network** - Secure internal communication between servers
- **HTTPS** - Encrypted client-server communication
- **Database Connections** - Secure data access protocols

Swimlane Activity Diagram for Online Shopping Process

Swimlane Activity Diagram for Online Shopping System



Key Process Flow:

- **Sequential Activities** - Clear step-by-step customer journey
- **Decision Points** - Product availability and repeat order checks
- **Cross-Lane Interactions** - Customer inputs trigger system responses
- **Error Handling** - Out-of-stock notifications and alternative flows
- **Process Boundaries** - Clear start and end points with option loops

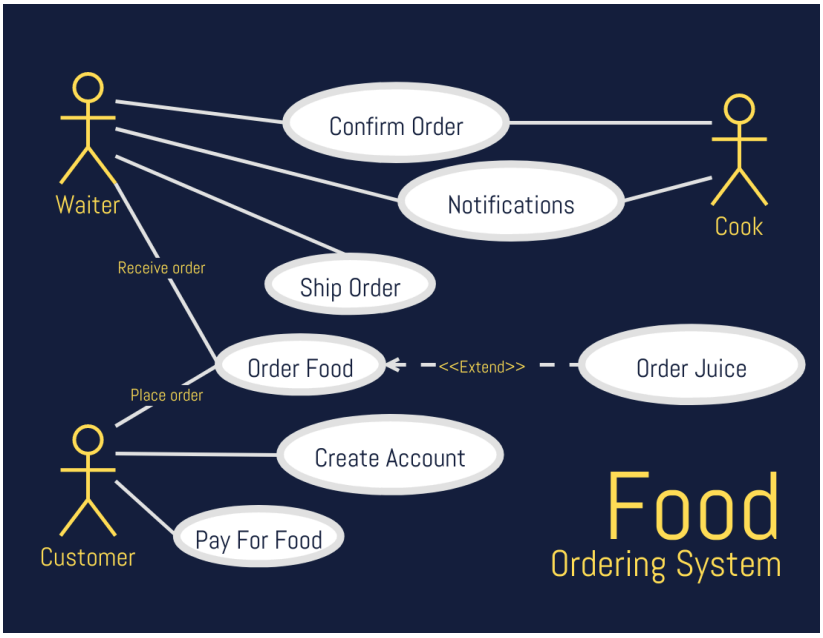
Benefits:

- **Role Clarity** - Separates customer actions from system responses
- **Process Visualization** - Shows complete shopping workflow
- **Error Management** - Handles inventory and system exceptions
- **User Experience** - Demonstrates customer interaction patterns

This design model provides complete architectural guidance for implementing a robust online shopping system with clear component distribution and user workflow management.

Q4. Draw Use case / Class diagram for online food ordering system.

Use Case Diagram for Online Food Ordering System:



Actors:

- **Customer** - Places orders, makes payments

- **Restaurant Manager** - Manages menu, processes orders
- **Delivery Person** - Handles food delivery
- **System Administrator** - Manages system settings
- **Payment Gateway** - Processes payments (external actor)

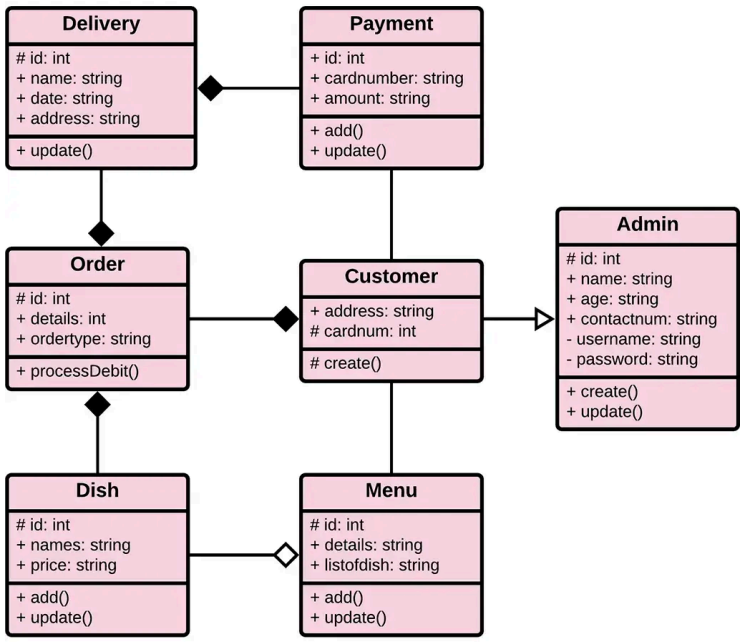
Use Cases:

- **User Management:** Register, Login, Update Profile
- **Menu Management:** Browse Menu, Search Food Items, View Details
- **Order Management:** Add to Cart, Place Order, Track Order, Cancel Order
- **Payment Processing:** Select Payment Method, Process Payment, Generate Receipt
- **Delivery Management:** Assign Delivery, Update Delivery Status, Confirm Delivery
- **Administration:** Manage Users, Generate Reports, System Maintenance

Relationships:

- **Include:** Place Order includes Process Payment
- **Extend:** Track Order extends from Place Order
- **Generalization:** Premium Customer inherits from Customer

Class Diagram Components:



Core Classes:

- **User** (attributes: userID, name, email, phone)

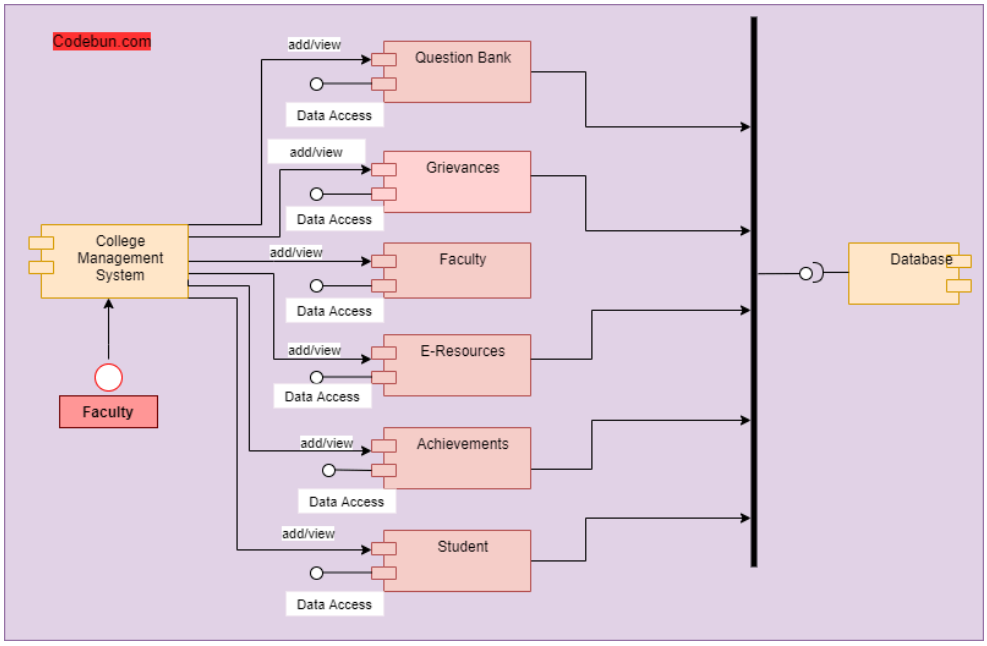
- **Customer** (inherits from User, additional: address, paymentMethod)
- **Restaurant** (restaurantID, name, location, cuisine)
- **FoodItem** (itemID, name, price, description, category)
- **Order** (orderID, orderDate, status, totalAmount)
- **OrderItem** (quantity, price, customizations)
- **Payment** (paymentID, amount, method, status)
- **Delivery** (deliveryID, address, status, estimatedTime)

Relationships:

- Customer **places** Order (1:many)
- Order **contains** OrderItem (1:many)
- OrderItem **references** FoodItem (many:1)
- Order **has** Payment (1:1)
- Order **requires** Delivery (1:1)

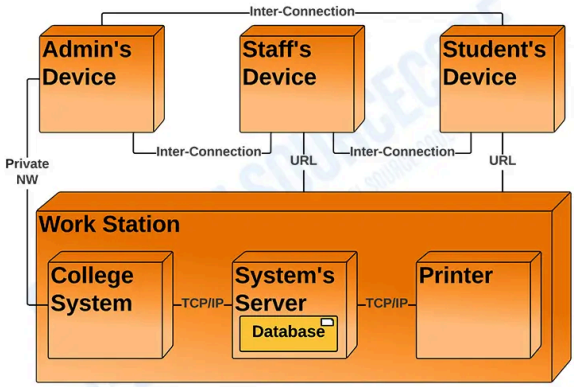
Q5. Draw UML component/Deployment diagram for college management system.

UML Component Diagram for College Management System:



UML Deployment Diagram for College Management System:

COLLEGE MANAGEMENT SYSTEM



DEPLOYMENT DIAGRAM

Physical Architecture Components:

Hardware Nodes:

- **Client Workstations** - Student/Faculty computers
- **Web Server** - Application hosting environment
- **Database Server** - Academic data storage
- **File Server** - Document and resource storage
- **Print Server** - Academic document printing
- **Network Infrastructure** - Campus network backbone

Software Components:

- **Student Portal** - Web-based student interface
- **Faculty Management System** - Academic staff tools
- **Academic Management Module** - Course and curriculum management
- **Examination System** - Test and grading management
- **Library Management** - Resource tracking system
- **Fee Management** - Financial transaction processing
- **Database Management System** - Data persistence layer

Deployment Specifications:

Server Configurations:

- **Application Server:** Java EE/Spring Boot deployment

- **Database Server:** MySQL/PostgreSQL with backup systems
- **Web Server:** Apache/Nginx for HTTP request handling
- **Security Layer:** SSL/TLS encryption and firewall protection

Network Connections:

- **LAN Connections** - Campus-wide network access
- **Internet Gateway** - External connectivity for remote access
- **Database Connections** - Secure data access protocols
- **Backup Connections** - Redundant data protection links

System Integration:

- **Identity Management** - LDAP/Active Directory integration
- **Email System** - SMTP server for notifications
- **Document Management** - File storage and retrieval system
- **Reporting Engine** - Academic report generation tools

Advantages:

- Visualizes physical system architecture
- Shows hardware-software relationships
- Models system distribution and scalability
- Facilitates deployment planning and resource allocation
- Enables performance optimization and load balancing

Q6. What is Agility? Explain Kanban in detail.

Agility in Software Engineering:

Definition: Agility refers to the ability to rapidly adapt to changing requirements, deliver working software frequently, and respond effectively to uncertainty in software development.

Agility Characteristics:

- **Adaptability** - Quick response to changing requirements
- **Iterative Development** - Short development cycles
- **Customer Collaboration** - Continuous stakeholder engagement
- **Working Software** - Focus on functional deliverables over documentation
- **Flexibility** - Ability to pivot when necessary

Kanban Methodology in Detail:

Kanban Overview:

Kanban is a visual project management framework that optimizes workflows through real-time tracking and collaboration. Originally from Toyota Production System, it's adapted for software development.

Core Kanban Principles:

1. **Start with what you do now** - Begin with existing processes
2. **Agree to pursue incremental change** - Implement gradual improvements
3. **Respect current process and roles** - Don't disrupt existing hierarchy
4. **Encourage leadership at all levels** - Distributed decision making

Kanban Practices:

1. **Visualize Workflow** - Use Kanban boards to show all work items
2. **Limit Work in Progress (WIP)** - Control amount of concurrent work
3. **Manage Flow** - Optimize task progression through system
4. **Make Process Policies Explicit** - Clear rules and procedures
5. **Improve Collaboratively** - Continuous team-based enhancement

Kanban Board Structure:

- **To Do** - Backlog of pending tasks
- **In Progress** - Currently active work items
- **Done** - Completed deliverables
- **Additional columns** can be customized based on workflow needs

Benefits:

- **Reduced Cycle Time** - Faster feature delivery
- **Improved Responsiveness** - Better adaptation to priority changes
- **Enhanced Visibility** - Clear progress tracking
- **Waste Reduction** - Elimination of non-value activities
- **Flexibility** - Easy implementation without major organizational changes

Q7. Explain types of coupling & cohesion, and also benefits of high cohesion & low coupling.

Types of Coupling:

Coupling measures the degree of interdependence between software modules.

Coupling Types (from loosest to tightest):

1. **Data Coupling** - Modules share data through parameters; most desirable
2. **Stamp Coupling** - Modules share composite data structures
3. **Control Coupling** - One module controls behavior of another
4. **External Coupling** - Modules depend on external environment
5. **Common Coupling** - Modules share global data
6. **Content Coupling** - One module directly accesses another's content; least desirable

Types of Cohesion:
Cohesion measures how closely related elements within a module work together.

Cohesion Types (from strongest to weakest):

1. **Functional Cohesion** - All elements perform single, well-defined task; most desirable
2. **Sequential Cohesion** - Elements arranged in sequence, output of one is input of next
3. **Communicational Cohesion** - Elements operate on same input data
4. **Procedural Cohesion** - Elements follow specific sequence of execution
5. **Temporal Cohesion** - Elements executed at same time
6. **Logical Cohesion** - Elements perform similar functions but not related
7. **Coincidental Cohesion** - Elements grouped arbitrarily; least desirable

Benefits of High Cohesion:

- **Easier Maintenance** - Related functionality grouped together
- **Better Testability** - Focused modules easier to test
- **Improved Readability** - Clear module purpose and responsibilities
- **Enhanced Reusability** - Well-defined modules can be reused
- **Reduced Complexity** - Single-purpose modules easier to understand
- **Better Error Isolation** - Problems contained within modules

Benefits of Low Coupling:

- **Improved Maintainability** - Changes in one module don't affect others
- **Enhanced Flexibility** - Modules can be modified independently
- **Better Scalability** - System can grow without increasing dependencies
- **Easier Testing** - Modules can be tested in isolation
- **Reduced System Complexity** - Fewer interdependencies to manage

- **Improved Reliability** - Failures in one module don't cascade
- **Better Team Collaboration** - Teams can work on modules independently

Combined Benefits:
Systems with **high cohesion and low coupling** achieve:

- **Optimal Modularity** - Well-structured, manageable components
- **Maximum Maintainability** - Easy to update and extend
- **Superior Quality** - Fewer defects and better performance
- **Cost Effectiveness** - Reduced development and maintenance costs
- **Team Productivity** - Parallel development and clear responsibilities

Q8. Discuss various types of design patterns.

Design Pattern Categories:
Design patterns are reusable solutions to common software design problems, divided into three main categories.

1. Creational Patterns:

Purpose: Focus on object creation mechanisms and class instantiation.

Key Patterns:

- **Singleton** - Ensures single instance of class exists globally
  - *Use Case:* Database connections, logging services
  - *Benefits:* Controlled access, memory efficiency
- **Factory Method** - Creates objects without specifying exact classes
  - *Use Case:* Creating UI elements based on platform
  - *Benefits:* Flexibility, loose coupling
- **Abstract Factory** - Creates families of related objects
  - *Use Case:* Cross-platform applications
  - *Benefits:* Consistency, easy switching between product families
- **Builder** - Constructs complex objects step by step
  - *Use Case:* Creating configuration objects
  - *Benefits:* Flexible construction, readable code
- **Prototype** - Creates objects by cloning existing instances
  - *Use Case:* Game object creation, document templates
  - *Benefits:* Performance optimization, dynamic object creation

2. Structural Patterns:

Purpose: Deal with object composition and relationships between entities.

Key Patterns:

- **Adapter -** Allows incompatible interfaces to work together
  - *Use Case:* Legacy system integration
  - *Benefits:* Code reusability, interface compatibility
- **Facade -** Provides simplified interface to complex subsystems
  - *Use Case:* API wrappers, system interfaces
  - *Benefits:* Simplified usage, decoupling
- **Decorator** - Adds new functionality to objects dynamically
  - *Use Case:* UI component enhancement
  - *Benefits:* Flexible extension, runtime modification
- **Composite** - Treats individual and composite objects uniformly
  - *Use Case:* File system structures, UI hierarchies
  - *Benefits:* Simplified client code, uniform treatment
- **Bridge** - Separates abstraction from implementation
  - *Use Case:* Device drivers, graphics rendering
  - *Benefits:* Platform independence, runtime binding

3. Behavioral Patterns:

Purpose: Focus on communication between objects and assignment of responsibilities.

Key Patterns:

- **Observer** - Notifies multiple objects about state changes
  - *Use Case:* Event handling, MVC architecture
  - *Benefits:* Loose coupling, dynamic relationships
- **Strategy** - Defines family of algorithms and makes them interchangeable
  - *Use Case:* Payment processing, sorting algorithms
  - *Benefits:* Algorithm flexibility, easy maintenance
- **Template Method** - Defines algorithm skeleton, subclasses provide details
  - *Use Case:* Data processing workflows
  - *Benefits:* Code reuse, consistent structure

- **Command** - Encapsulates requests as objects
  - *Use Case:* Undo operations, macro recording
  - *Benefits:* Decoupling, queuing requests
- **State** - Changes object behavior based on internal state
  - *Use Case:* State machines, game character behavior
  - *Benefits:* Clean state management, easy extension

Design Pattern Benefits:

Development Advantages:

- **Reusability** - Proven solutions for common problems
- **Communication** - Common vocabulary for developers
- **Best Practices** - Established design principles
- **Maintainability** - Well-structured, modular code
- **Time Savings** - Avoid reinventing solutions

Quality Improvements:

- **Flexibility** - Easy to modify and extend
- **Reliability** - Battle-tested solutions
- **Scalability** - Support for growing requirements
- **Testability** - Clear separation of concerns
- **Documentation** - Self-documenting code structure

Selection Criteria:

- **Problem Context** - Match pattern to specific problem
- **Complexity Trade-offs** - Balance flexibility vs. simplicity
- **Performance Considerations** - Evaluate overhead
- **Team Expertise** - Consider team familiarity
- **Maintenance Requirements** - Long-term support needs