Software Engineering (Practical) – Lab Manual

BCA Semester 6

List of Practicals

1. Study and Comparison of Software Process Models (Prepare a comparative table listing advantages, disadvantages, and suitable project types for each).
2. Preparation of Software Requirement Specification (SRS) Document (Analyze a given problem statement (e.g., Library Management System, College Attendance System) and create an IEEE-compliant SRS).
3. Structured vs. Object-Oriented Analysis for the Same Problem (Perform both Structured Analysis (DFD, ER Diagram) and Object-Oriented Analysis (Class Identification, Use Cases) for the same project).
4. Project Planning and Estimation (W5HH & COCOMO) (Use the W5HH principle to plan a project (Who, What, When, Where, How, how much). Perform software effort estimation using Basic COCOMO Model).
5. Risk Management and Quality Planning Report (Identify at least 5 major project risks, analyze their impact and probability, and propose mitigation strategies. Include a short quality plan covering QA and QC activities).
6. Use Case Modeling and Diagrams (Prepare Use Case Descriptions and corresponding Use Case Diagrams for the SRS developed earlier. Identify main actors, use cases, and relationships).
7. Design UML Diagrams (Class, Sequence, and Activity).
8. Software Testing – Test Case Design and Verification (Write at least 10 test cases for one module using Black Box and White Box techniques. Indicate input, expected output, and actual output.)
9. Software Testing Levels and Execution Report (Demonstrate different testing levels (Unit, Integration, System, Acceptance) conceptually using one module. Prepare a short testing report).
10. Demonstration of CASE Tools (Explore a CASE tool such as StarUML, Visual Paradigm, or Rational Rose. Demonstrate how it supports software design (UML creation, code generation, or documentation).

☑ Practical: 1 Study and Comparison of Software Process Models (Prepare a comparative table listing advantages, disadvantages, and suitable project types for each).

Aim

To study and compare various Software Process Models used in software development.

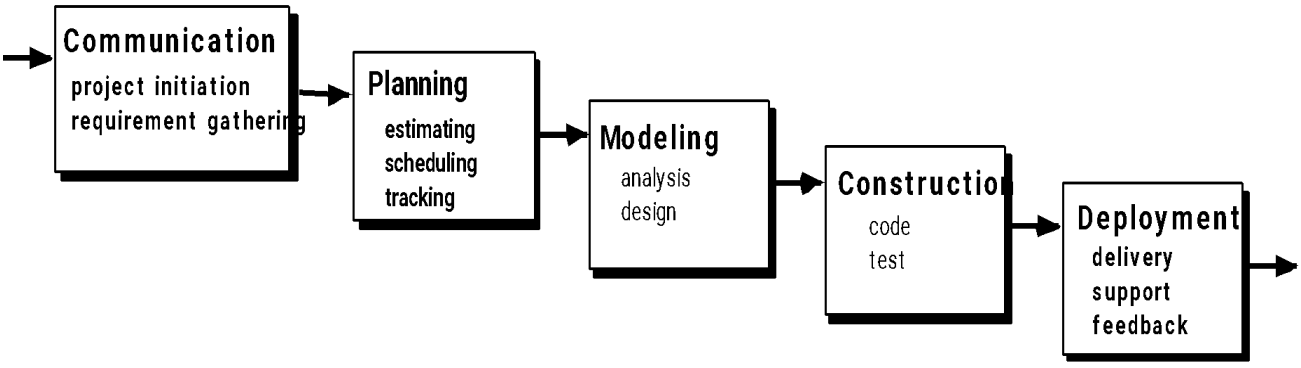
Theory

Software Process Models define the structured approach used to plan, develop, test, and maintain software systems. Choosing the right model improves project efficiency, quality, and risk management.

The commonly used software process models are:

---

1. Waterfall Model



Description

A linear sequential model where each phase must be completed before the next begins.
Phases: Requirements → Design → Implementation → Testing → Deployment → Maintenance.

Advantages

- Simple and easy to understand
- Well-defined stages and documentation
- Suitable for small, stable-requirement projects

Disadvantages

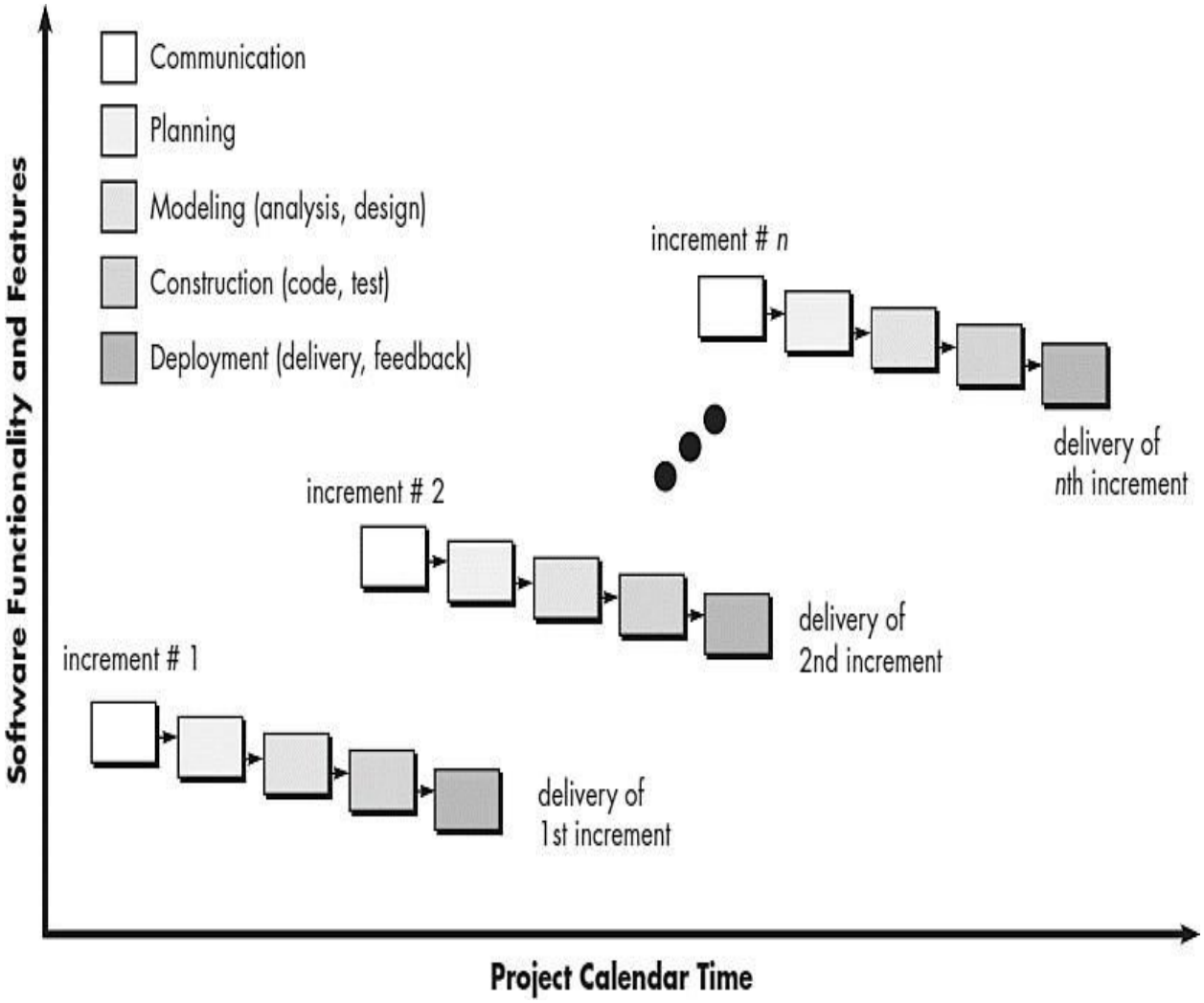
- No feedback loops
- Poor model if requirements change
- Late testing

Suitable For

- Well-defined projects with fixed requirements (e.g., payroll systems)

---

2. Incremental Model



Description

Software is built and delivered in small increments. Each increment adds more functionality.

Advantages

- Early delivery of working modules
- Flexible to requirement changes
- Easier risk management

Disadvantages

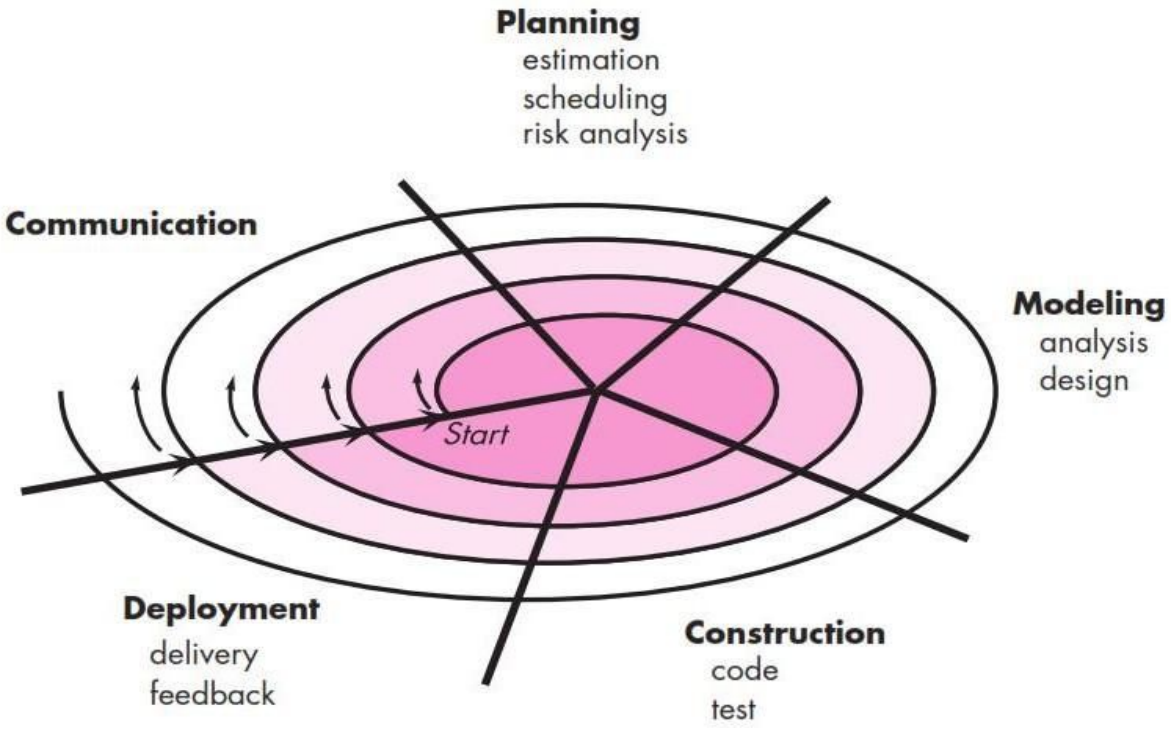
- Needs good planning
- Architecture must support incremental development

Suitable For

- Medium-sized projects with partial requirement clarity

---

3. Spiral Model



Description

Combines iterative development with systematic risk analysis. Each cycle (spiral) includes planning, risk evaluation, development, and review.

Advantages

- Best model for high-risk projects

- Frequent customer feedback
- Flexible and scalable

Disadvantages

- Expensive and complex
- Requires expertise in risk management

Suitable For

- Large, critical, and complex systems (e.g., aerospace, defense)

---

4. Agile Model

Description

An iterative, customer-centric approach with short development cycles (sprints). Focuses on collaboration, adaptability, and rapid delivery.

Advantages

- Highly flexible and adaptive
- Continuous feedback from customer
- Working software delivered frequently

Disadvantages

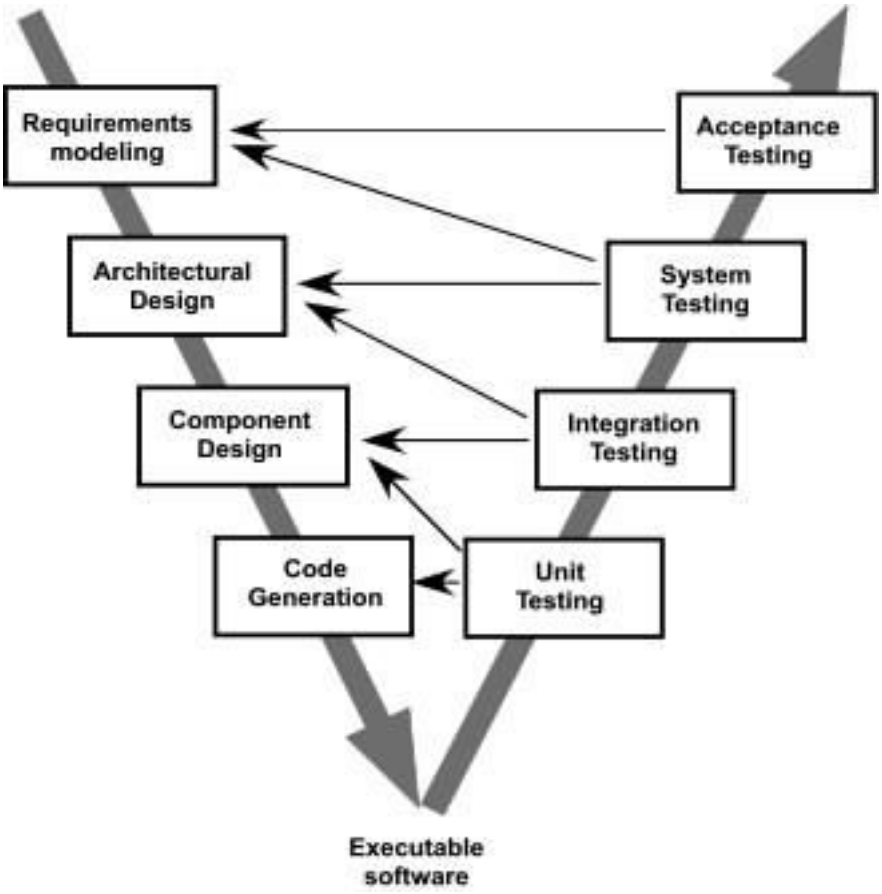
- Requires experienced team
- Documentation may be minimal
- Difficult for large distributed teams

Suitable For

- Projects with frequently changing requirements
- Start-ups and fast-evolving applications

---

5. V-Model (Verification & Validation Model)



Description

An extension of Waterfall that emphasizes testing at each development stage.

Advantages

- Testing is planned early
- Highly structured and disciplined
- Suitable for safety-critical systems

Disadvantages

- Rigid and not flexible
- Requirement changes are costly

Suitable For

- Medical devices, embedded systems

Comparison Table

Feature	Waterfall	Incremental	Spiral	Agile	V-Model
Requirement Flexibility	Low	Medium–High	High	Very High	Low
Risk Handling	Low	Medium	Very High	High	Medium
Customer Involvement	Low	Medium	High	Very High	Medium
Delivery	One-time	Incremental	Iterative with risk loops	Frequent Sprints	One-time
Cost	Low–Medium	Medium	High	Medium	Medium
Best For	Small stable projects	Medium projects	Large high-risk	Dynamic requirements	Critical systems

Conclusion

Different software process models are suitable for different project types.

- **Waterfall and V-Model** are ideal for stable and safety-critical environments.
- **Incremental and Agile** work well when requirements evolve.
- **Spiral** is best when risk management is essential.

Choosing the right model improves project success, cost efficiency, and product quality.

☑ Practical: 2 Preparation of Software Requirement Specification (SRS) Document (Analyze a given problem statement (e.g., Library Management System, College Attendance System) and create an IEEE-compliant SRS).

Software Requirement Specification (SRS)

Online Food Ordering System

1. Introduction

1.1 Purpose

The purpose of this SRS document is to describe the requirements for the **Online Food Ordering System (OFOS)**. It provides a detailed description of the functional and non-functional requirements, system features, constraints, and overall system behavior. This document will be used by developers, testers, project managers, and stakeholders.

1.2 Scope

The Online Food Ordering System allows customers to browse restaurants, view menus, place food orders, make online payments, and track delivery. Restaurant owners can manage menus, receive and process orders, and update order status. Admins can manage users, restaurants, and system-level configurations.

1.3 Definitions, Acronyms, & Abbreviations

- **OFOS** – Online Food Ordering System
- **UI** – User Interface
- **DBMS** – Database Management System
- **OTP** – One-Time Password

1.4 Overview

This document defines system features, constraints, functional and non-functional requirements, interface descriptions, and use cases for the Online Food Ordering System.

2. Overall Description

2.1 Product Perspective

The OFOS is a standalone web/mobile application designed for customers, restaurants, and administrators. It includes:

- Customer Interface
- Restaurant Dashboard
- Admin Panel
- Centralized Database

2.2 Product Functions

- User Registration & Login
- Browse Restaurants & Menus
- Add Items to Cart
- Place Order
- Payment (COD/Online)
- Order Tracking
- Restaurant Menu Management
- Admin Management

2.3 User Classes and Characteristics

User Type	Description
Customer	Places food orders
Restaurant Owner	Manages menu, accepts/rejects orders
Delivery Person	Delivers food and updates delivery status
Admin	Manages users, restaurants, system settings

2.4 Operating Environment

- Web browser / Android / iOS
- Backend: Java/Python/PHP/Node.js (any)

- Database: MySQL/PostgreSQL
- OS: Windows/Linux

2.5 Design and Implementation Constraints

- Must follow online payment security standards
- Internet connection required
- Responsive UI must be used

2.6 Assumptions and Dependencies

- User must have a valid mobile number/email
- Restaurants provide correct pricing and availability
- Payment gateway availability

---

3. System Features / Functional Requirements

3.1 User Registration and Login

Description: Allows users to create an account and login.
Functional Requirements:

- F1.1 User shall register using email/mobile and password.
- F1.2 System shall validate credentials during login.
- F1.3 System shall allow password recovery using OTP.

3.2 Browse Restaurants

- F2.1 User shall view a list of available restaurants.
- F2.2 User shall filter restaurants (by cuisine, rating, discount).
- F2.3 User shall view restaurant menu.

3.3 Cart Management

- F3.1 User shall add menu items to cart.
- F3.2 User shall increase/decrease quantity.
- F3.3 User shall remove item from cart.
- F3.4 System shall calculate total price.

3.4 Placing an Order

- F4.1 User shall place an order from selected restaurant.
- F4.2 User shall choose a payment method (COD/Online).
- F4.3 System shall send order details to the restaurant.

3.5 Order Processing (Restaurant Side)

- F5.1 Restaurant shall accept or reject order.
- **F5.2 Restaurant shall update order status (Preparing → Ready → Out for Delivery).**

3.6 Delivery Management

- F6.1 Delivery person shall view assigned orders.
- F6.2 Delivery person shall update order status (Delivered).

3.7 Admin Management

- F7.1 Admin shall manage restaurants (add/remove/update).
- F7.2 Admin shall manage users.
- F7.3 Admin shall generate system reports.

---

4. External Interface Requirements

4.1 User Interface Requirements

- Clean and intuitive layout
- Mobile-friendly design
- Restaurant images, dish images
- Order summary screen

4.2 Hardware Interfaces

- Mobile device or computer
- Payment gateway device (server-side)

4.3 Software Interface

- Database system
- Payment gateway APIs
- SMS/Email notification APIs

5. Non-Functional Requirements (NFRs)

5.1 Performance Requirements

- System should handle at least 1000+ concurrent users.
- Response time must be < 3 seconds.

5.2 Security Requirements

- Data encryption for passwords.
- Secure payment integration.
- Role-based access control.

5.3 Reliability

- System uptime must be 99%.
- Backup and recovery must be supported.

5.4 Usability

- Easy to use UI for both customers and restaurants.

5.5 Scalability

- System should support addition of more restaurants and users.

6. System Models

6.1 Use Case Diagram

Main Use Cases:

- Login / Register
- Browse Restaurants
- Add to Cart
- Place Order
- Track Order
- Manage Menu (Restaurant)
- Admin Controls

6.2 Data Flow Diagram (DFD)

DFD Level 0:
User → OFOS → Restaurant / Database → User

DFD Level 1 includes:

- User Module
- Restaurant Module
- Order Processing
- Payment Processing

---

7. Other Requirements

- System must support future integration with coupons, offers, and wallet system.
- Should support multi-language interface options.

---

8. Conclusion

This SRS provides a complete description of the Online Food Ordering System, including its functional and non-functional requirements, system architecture, and external interface details. It acts as a baseline for design, development, and testing.

Below are **well-structured Use Case Tables** for the **Online Food Ordering System** — suitable for practical / SRS documentation.

Use Case Tables – Online Food Ordering System

1. Use Case: User Registration

Use Case ID	**UC01**
Use Case Name	User Registration
Actor	Customer
Pre-condition	User must not have an existing account
Post-condition	User account is created and stored in the database

Use Case ID	UC01
Description	User provides details to register into the system
Normal Flow	1. User opens registration page 2. Enters name, email/phone, password 3. System validates details 4. System creates new account 5. Confirmation message shown
Alternate Flow	If email/phone already exists → show error
Exception Flow	Network failure → registration fails

2. Use Case: User Login

Use Case ID	UC02
Use Case Name	User Login
Actor	Customer
Pre-condition	User account must exist
Post-condition	User is authenticated and logged in
Normal Flow	1. User enters email/phone & password 2. System verifies credentials 3. Login successful
Alternate Flow	Wrong password → show “Invalid credentials”
Exception Flow	Account locked after multiple failed attempts

3. Use Case: Browse Restaurants

Use Case ID	UC03
Use Case Name	Browse Restaurants
Actor	Customer

Use Case ID	UC03
Pre-condition	User must be logged in
Post-condition	User can view restaurant list and menus
Normal Flow	1. User selects “Restaurants” 2. System displays restaurant list 3. User applies filters (optional) 4. User selects a restaurant to view menu
Exception Flow	No restaurants found in the area

4. Use Case: Add Item to Cart

Use Case ID	UC04
Use Case Name	Add to Cart
Actor	Customer
Pre-condition	User must be logged in; menu must be loaded
Post-condition	Item added to cart
Normal Flow	1. User selects item 2. Chooses quantity 3. System adds item to cart
Alternate Flow	If item is unavailable → system shows “Item not available”

5. Use Case: Place Order

Use Case ID	UC05
Use Case Name	Place Order
Actor	Customer

Use Case ID	UC05
Pre-condition	Items must be present in cart
Post-condition	Order is created and sent to restaurant
Normal Flow	1. User reviews cart 2. Selects payment method 3. Confirms order 4. System records order 5. System sends order to restaurant
Alternate Flow	Payment failed → user returns to payment page
Exception Flow	Restaurant becomes offline → order canceled automatically

6. Use Case: Accept/Reject Order (Restaurant)

Use Case ID	UC06
Use Case Name	Manage Incoming Orders
Actor	Restaurant Owner/Staff
Pre-condition	Order must be submitted by user
Post-condition	Order status updated
Normal Flow	1. Restaurant receives new order notification 2. Restaurant accepts or rejects order 3. System updates status
Alternate Flow	If restaurant rejects → system notifies customer

7. Use Case: Update Order Status

Use Case ID	UC07
Use Case Name	Update Order Status

Use Case ID	UC07
Actor	Restaurant Staff / Delivery Person
Pre-condition	Order must be accepted
Post-condition	Order status updated to next stage
Normal Flow	1. Staff updates status: Preparing → Ready → Out for Delivery 2. Delivery person updates Delivered
Exception Flow	Delivery failure → system marks “Delivery Failed”

8. Use Case: Manage Menu (Restaurant)

Use Case ID	UC08
Use Case Name	Menu Management
Actor	Restaurant Owner
Pre-condition	Restaurant must be registered and active
Post-condition	Updated menu is saved in database
Normal Flow	1. Owner adds/updates/deletes menu items 2. System updates menu database
Exception Flow	Invalid pricing or missing fields

9. Use Case: Admin Management

Use Case ID	UC09
Use Case Name	Admin Operations
Actor	Administrator
Pre-condition	Admin must be authenticated

Use Case ID	UC09
Post-condition	System-level changes saved
Normal Flow	1. Admin manages restaurants 2. Admin manages users 3. Admin generates reports
Exception Flow	Unauthorized access attempt blocked

10. Use Case: Track Order

Use Case ID	UC10
Use Case Name	Order Tracking
Actor	Customer
Pre-condition	User must have an active order
Post-condition	User sees real-time order status
Normal Flow	1. User opens “Track Order” 2. System retrieves latest status 3. Status displayed to user
Exception Flow	Network issue → unable to display live status

☑ Practical: 3 Structured vs. Object-Oriented Analysis for the Same Problem (Perform both Structured Analysis (DFD, ER Diagram) and Object-Oriented Analysis (Class Identification, Use Cases) for the same project).

Structured Analysis vs. Object-Oriented Analysis (OOA)

Aspect	Structured Analysis	Object-Oriented Analysis (OOA)
Definition	Focuses on **processes** and **data flow**. It models the system using **DFDs, data dictionaries, and process specifications**.	Focuses on **objects**, their **attributes**, **methods**, and **interactions**. It models the system using **UML diagrams** like class, sequence, activity, and use case diagrams.
Primary Focus	What the system **does** (functions, processes).	What the system **is** (objects/entities and their relationships).
Key Models/Diagrams	DFD (Level 0, 1, 2), Data Dictionary, State Diagrams.	Class Diagram, Use Case Diagram, Sequence Diagram, Activity Diagram, ER Diagram.
Approach	Top-down: Starts with **overall processes** → decomposed into sub-processes.	Bottom-up / iterative: Starts with **objects** → defines interactions and behaviors.
Data Representation	Data flows between processes and stores.	Data is encapsulated in **objects** (attributes) along with behavior (methods).
Function Handling	Functions (processes) are central; data is secondary.	Objects are central; functions (methods) belong to objects.
Reusability	Limited; processes and functions are tightly coupled with data flow.	High; objects and classes can be reused across systems.

Aspect	Structured Analysis	Object-Oriented Analysis (OOA)
System Example – OFOS	- DFD Level 0 shows the system as one process.- Level 1: Processes like User Management, Order Processing, Payment, Admin.- Focus on **how data moves** between Customer, System, Restaurant, Admin.	- Class Diagram defines objects: Customer, Order, Restaurant, Menu, Payment, Admin.- Use Case Diagram shows user interactions.- Sequence Diagram shows message flow.- Focus on **how objects interact and their responsibilities**.
Advantages	- Simple to understand for small systems.- Easier to track data flow.	- Promotes modularity and reusability.- Captures both **data and behavior**.- Better for complex systems.
Disadvantages	- Hard to maintain for large systems.- Behavior changes require redesign.	- Can be complex for beginners.- Requires detailed modeling upfront.
Best Suited For	Systems where **processes dominate** and data flows are simple.	Systems where **complex objects and interactions** exist, like Online Food Ordering System with multiple actors (Customer, Restaurant, Admin, Delivery Person).

Example Comparison – Online Food Ordering System

Structured Analysis (DFD Approach)

- **Processes:** User Management, Browse Restaurants, Cart & Order, Payment Processing, Restaurant Order Handling, Admin Management.
- **Data Stores:** Users, Restaurants, Orders, Payments.
- **Focus: Flow of information: Customer → System → Restaurant → Delivery → Admin.**

Object-Oriented Analysis (UML Approach)

- **Objects/Classes:** Customer, Restaurant, Menu, Item, Order, Payment, Admin.
- **Interactions:**

- **Customer places Order → triggers Payment → Restaurant updates Order → Delivery updates Status.**

- **Focus:** Responsibilities, behavior, and relationships of objects.

✅ **Key Insight:**

- **Structured Analysis** answers "**what the system does**" (process-centric).
- **Object-Oriented Analysis** answers "**what the system is and how it behaves**" (object-centric).

For modern systems like **Online Food Ordering**, **OOA is preferred** because it models real-world entities (Customer, Restaurant, Delivery) and their interactions, making maintenance, scalability, and enhancements easier.

DFD example :

1.1.Context Level DFD's

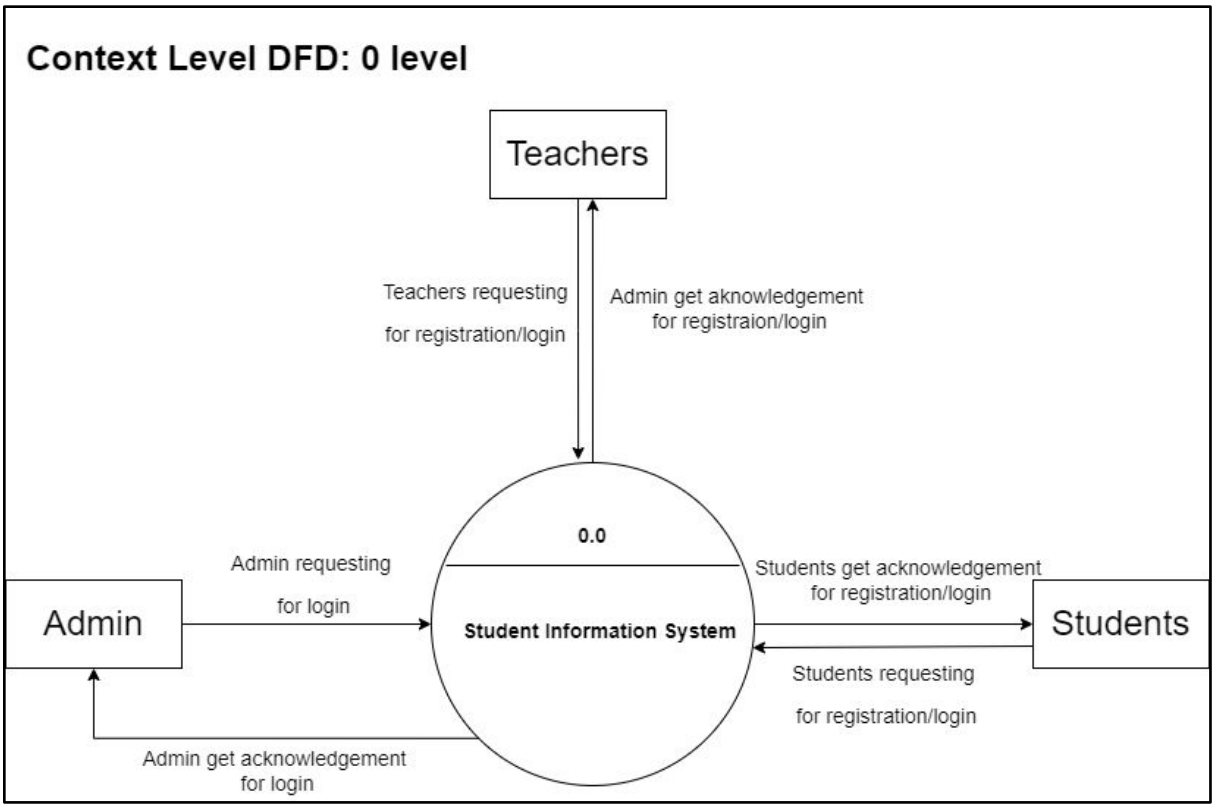


Figure 5.1.1. Context Level DFD: 0 Level

1.2.Level 1 DFD's:

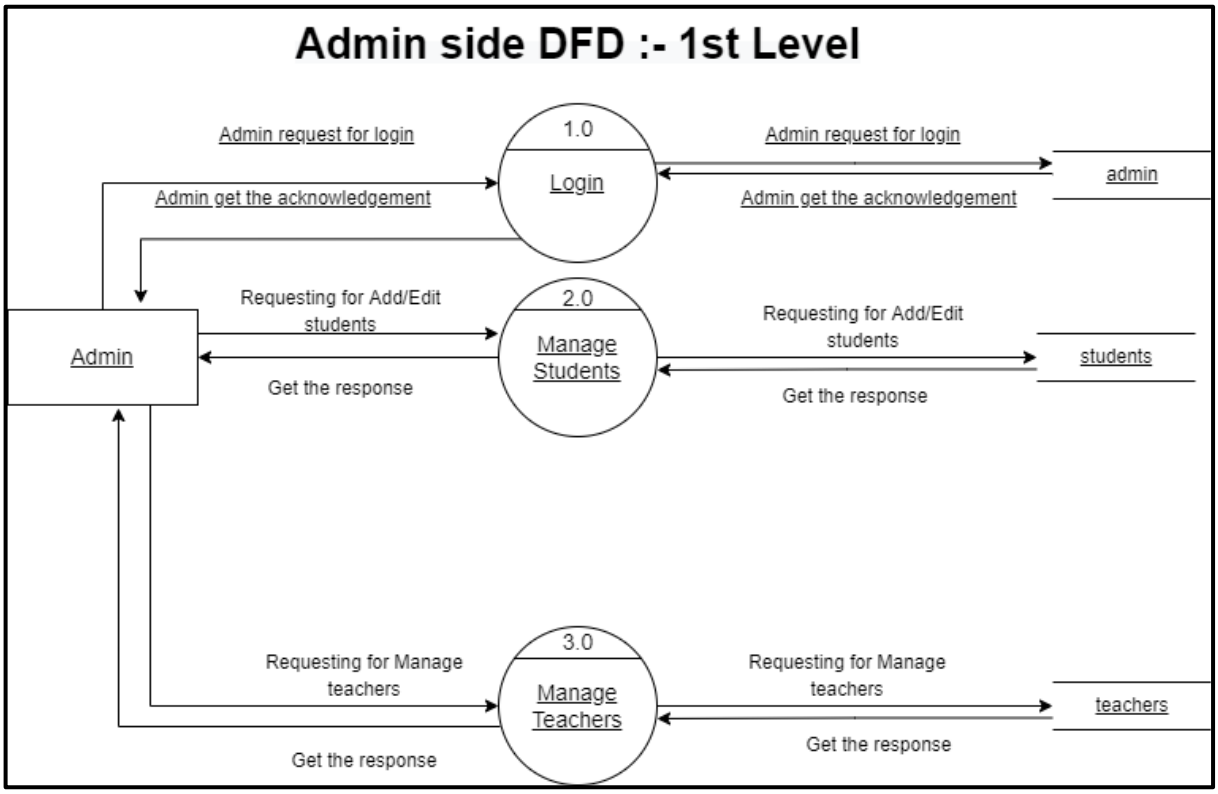


Figure 5.2.1. Admin Side DFD: 1st Level

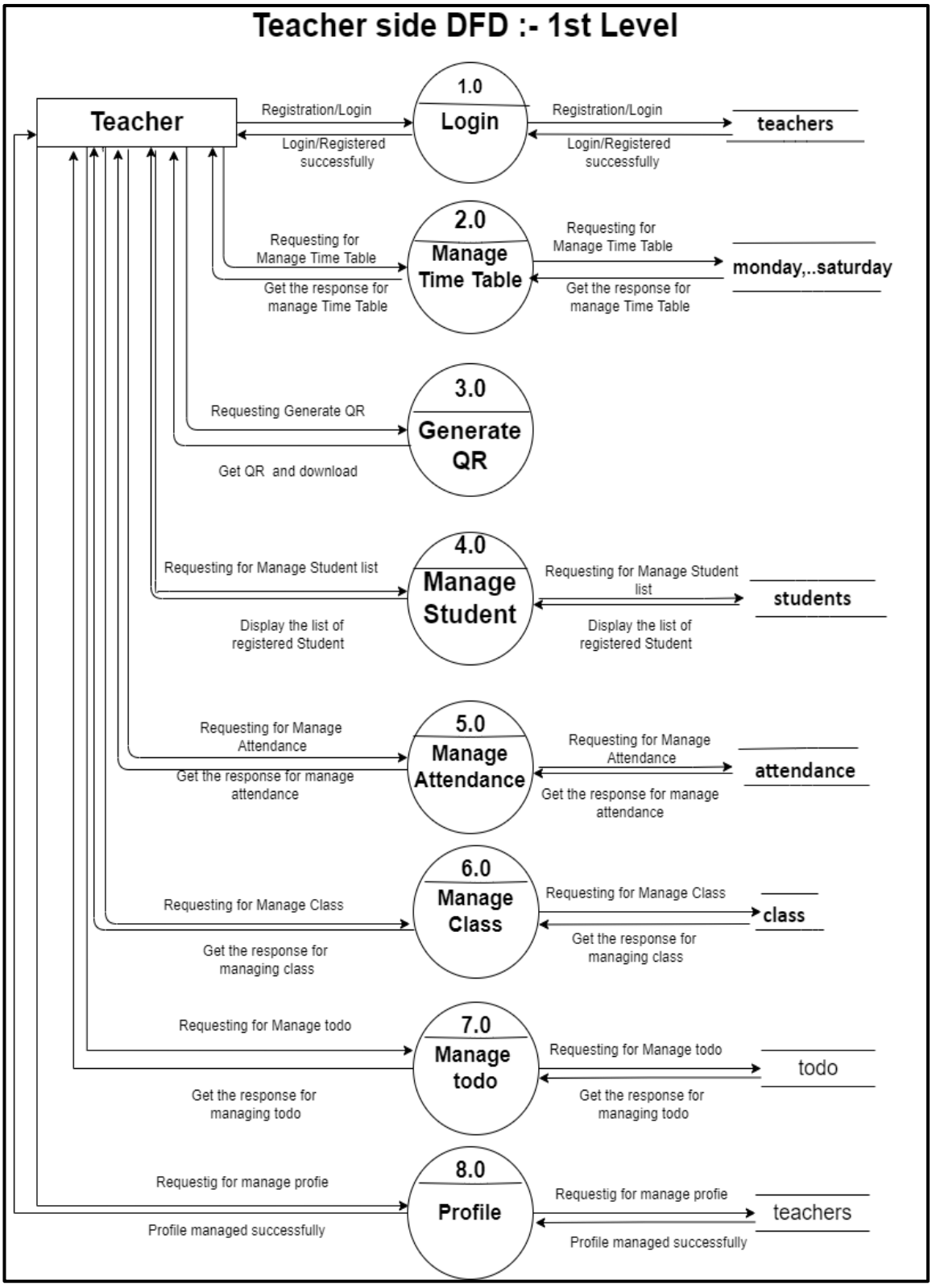


Figure 5.2.2. Teacher Side DFD: 1st Level

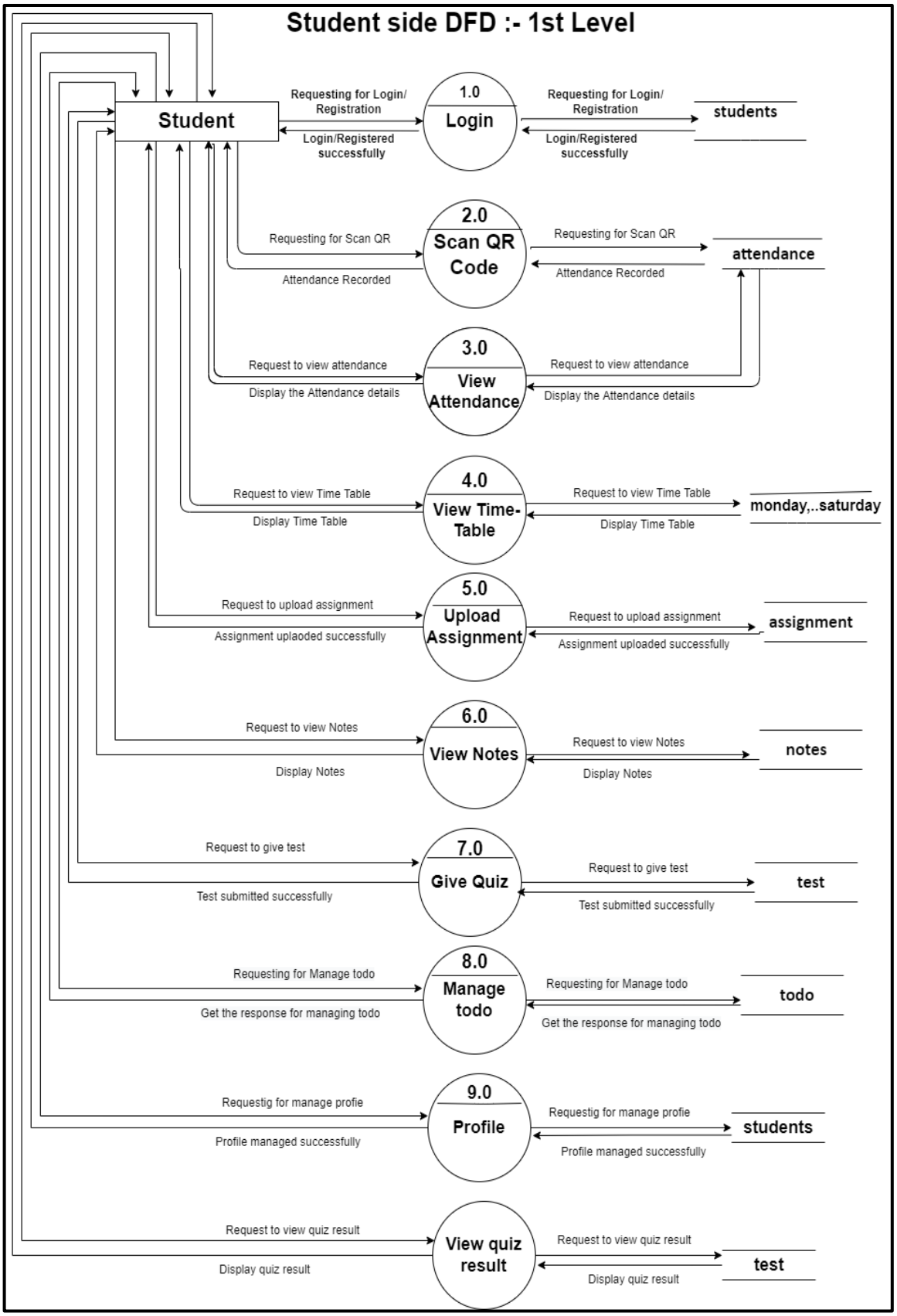

Student side DFD :- 1st Level
Student
1.0
Login
Requesting for Login/ Registration
Login/Registered successfully
Requesting for Login/ Registration
Login/Registered successfully
students
2.0
Scan QR Code
Requesting for Scan QR
Attendance Recorded
Requesting for Scan QR
Attendance Recorded
attendance
3.0
View Attendance
Request to view attendance
Display the Attendance details
Request to view attendance
Display the Attendance details
4.0
View Time-Table
Request to view Time Table
Display Time Table
Request to view Time Table
Display Time Table
monday,..saturday
5.0
Upload Assignment
Request to upload assignment
Assignment uplaoded successfully
Request to upload assignment
Assignment uploaded successfully
assignment
6.0
View Notes
Request to view Notes
Display Notes
Request to view Notes
Display Notes
notes
7.0
Give Quiz
Request to give test
Test submitted successfully
Request to give test
Test submitted successfully
test
8.0
Manage todo
Requesting for Manage todo
Get the response for managing todo
Requesting for Manage todo
Get the response for managing todo
todo
9.0
Profile
Requestig for manage profie
Profile managed successfully
Requestig for manage profie
Profile managed successfully
students
View quiz result
Request to view quiz result
Display quiz result
Request to view quiz result
Display quiz result
test

Figure 5.2.3. Student Side DFD: 1st Level

1.3. Level 2 DFD's

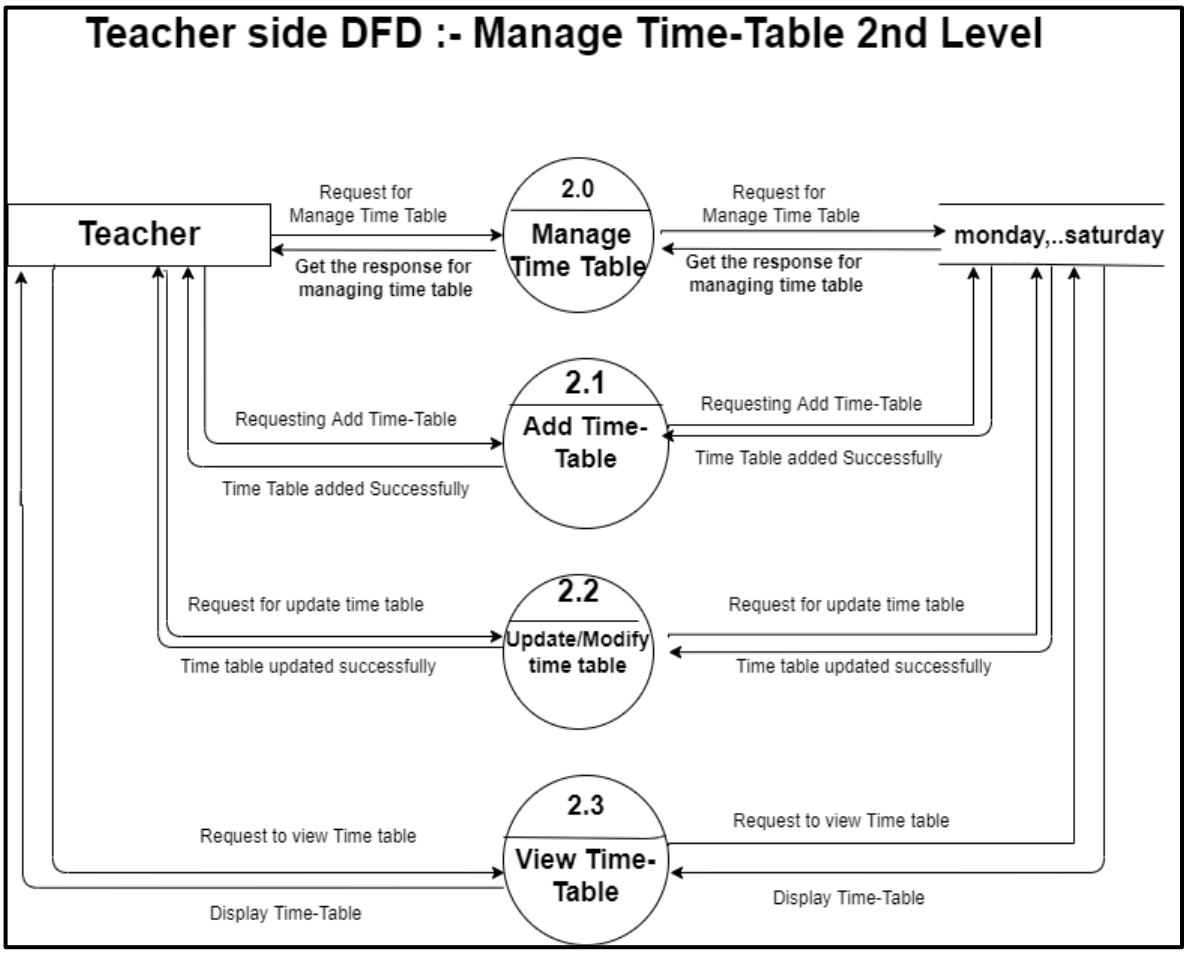


Figure 5.3.1. Teacher Side DFD: Manage Time-Table 2nd Level

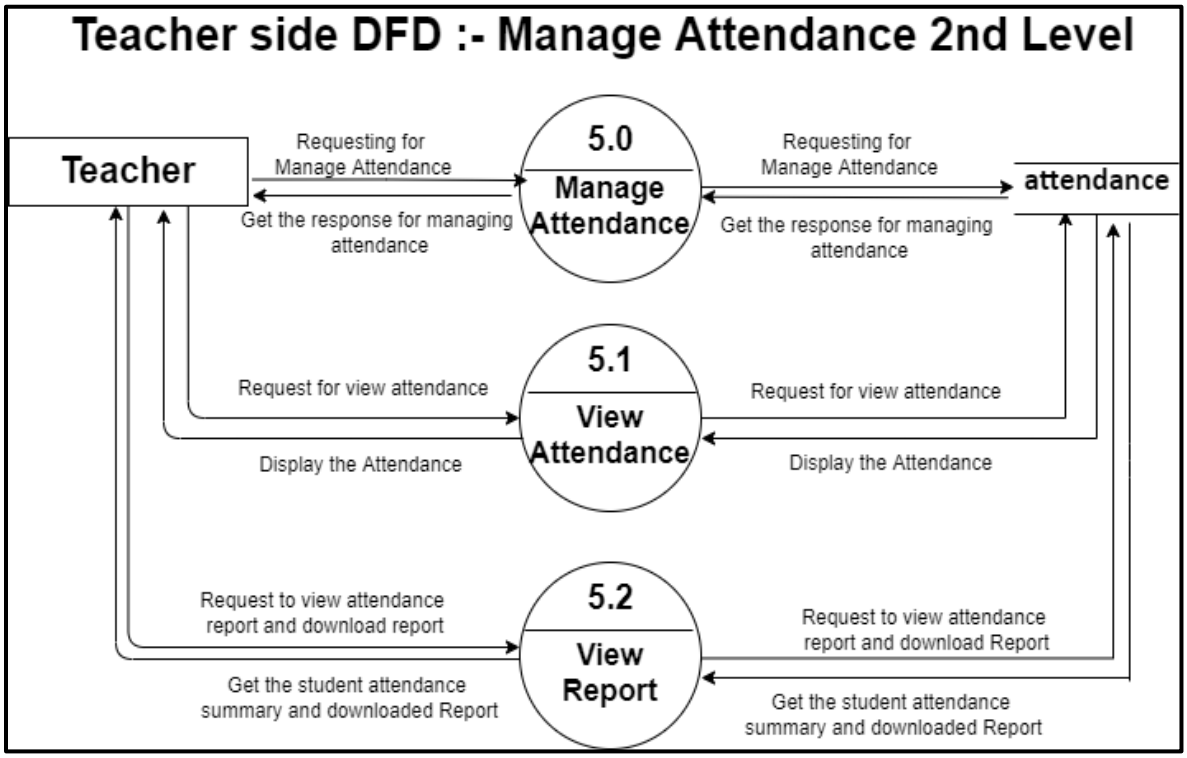


Figure 5.3.2. Teacher Side DFD: Manage Attendance 2nd Level

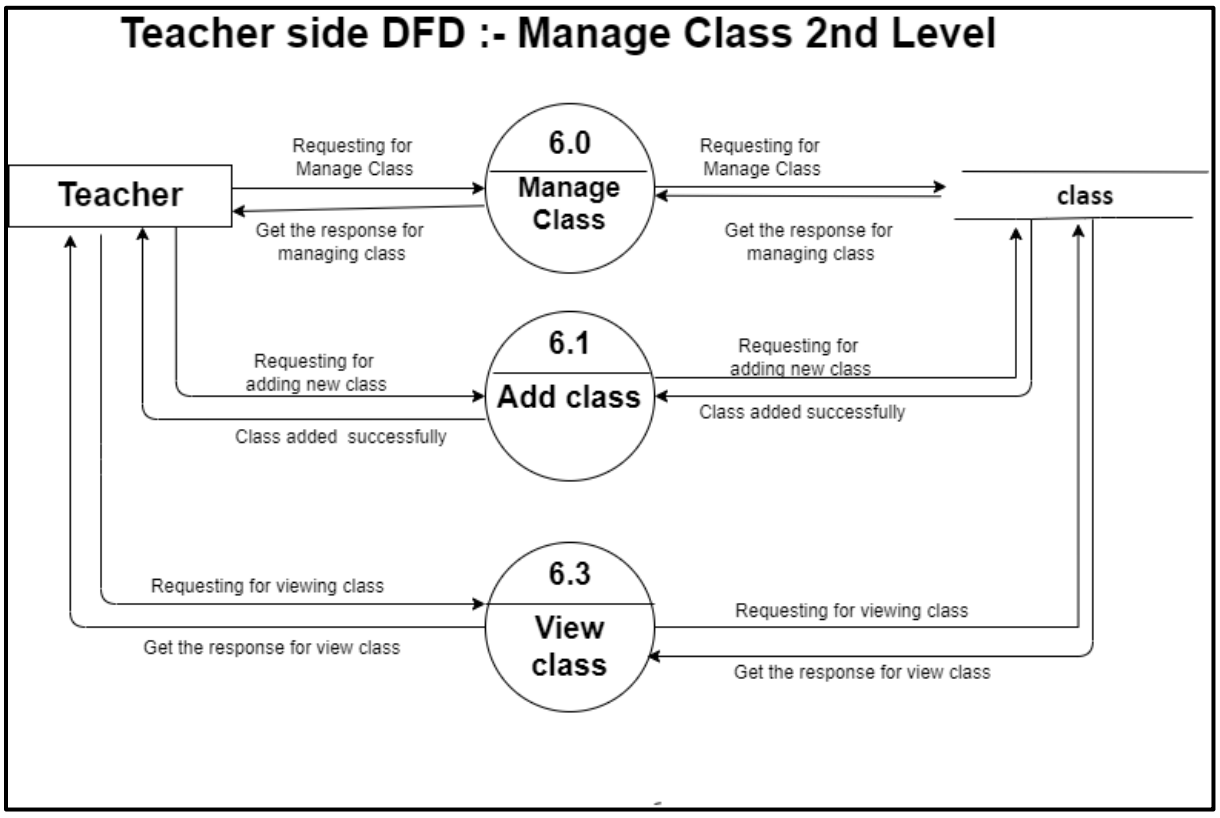


Figure 5.3.3. Teacher Side DFD: Manage Class-Table 2nd Level

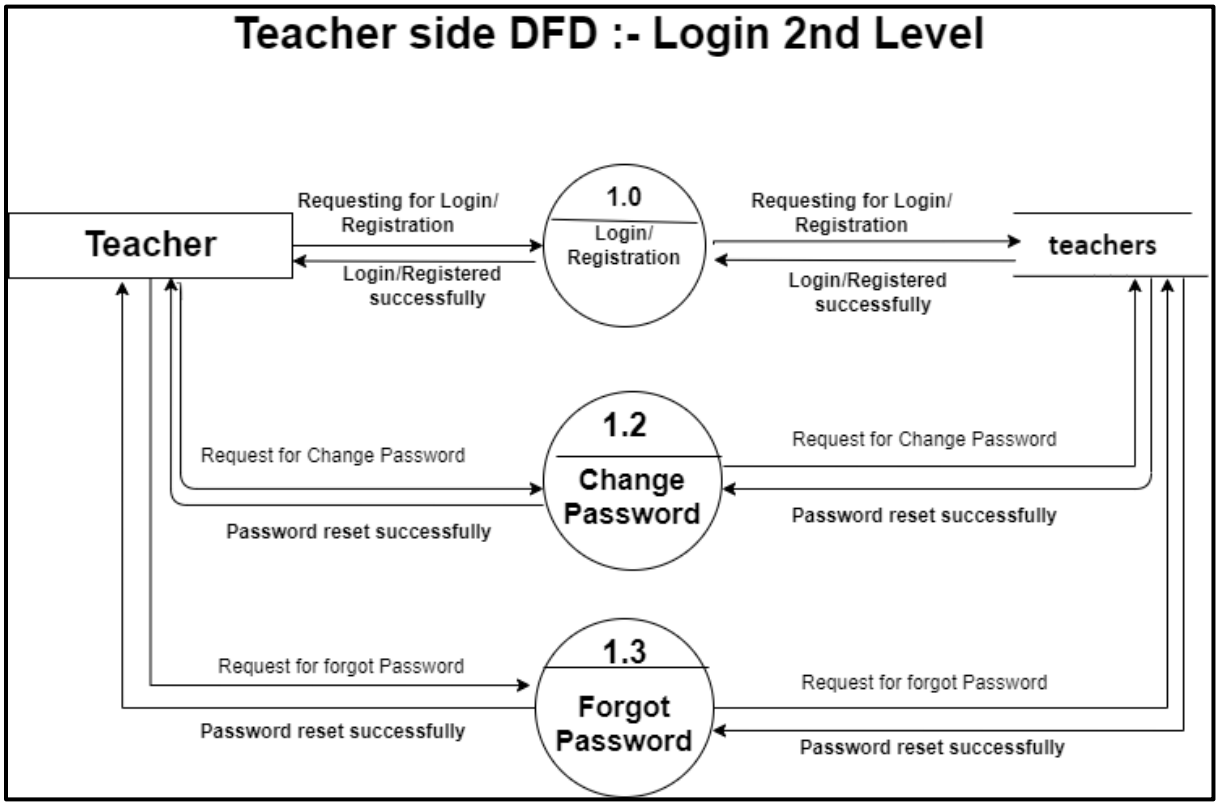


Figure 5.3.4. Teacher Side DFD: Login 2nd Level

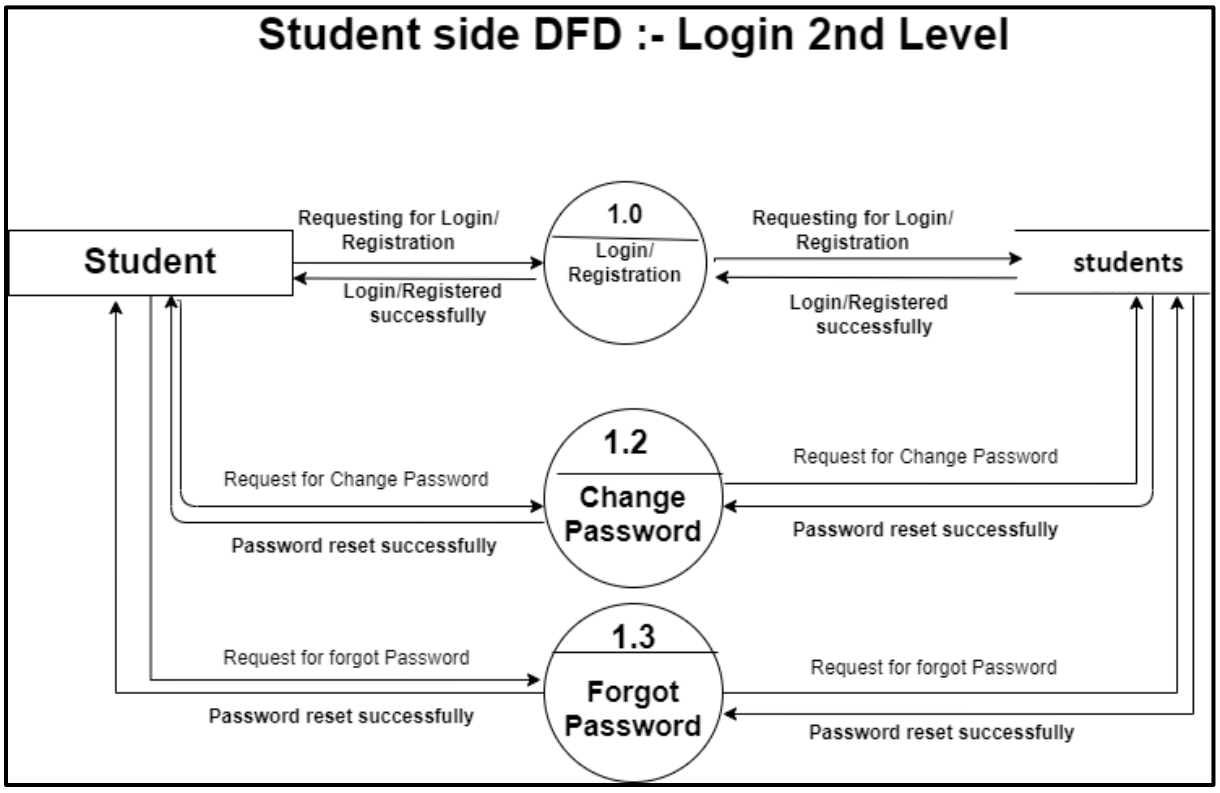


Figure 5.3.5. Student Side DFD: Login 2nd Level

1.4. Level 3 DFD's:

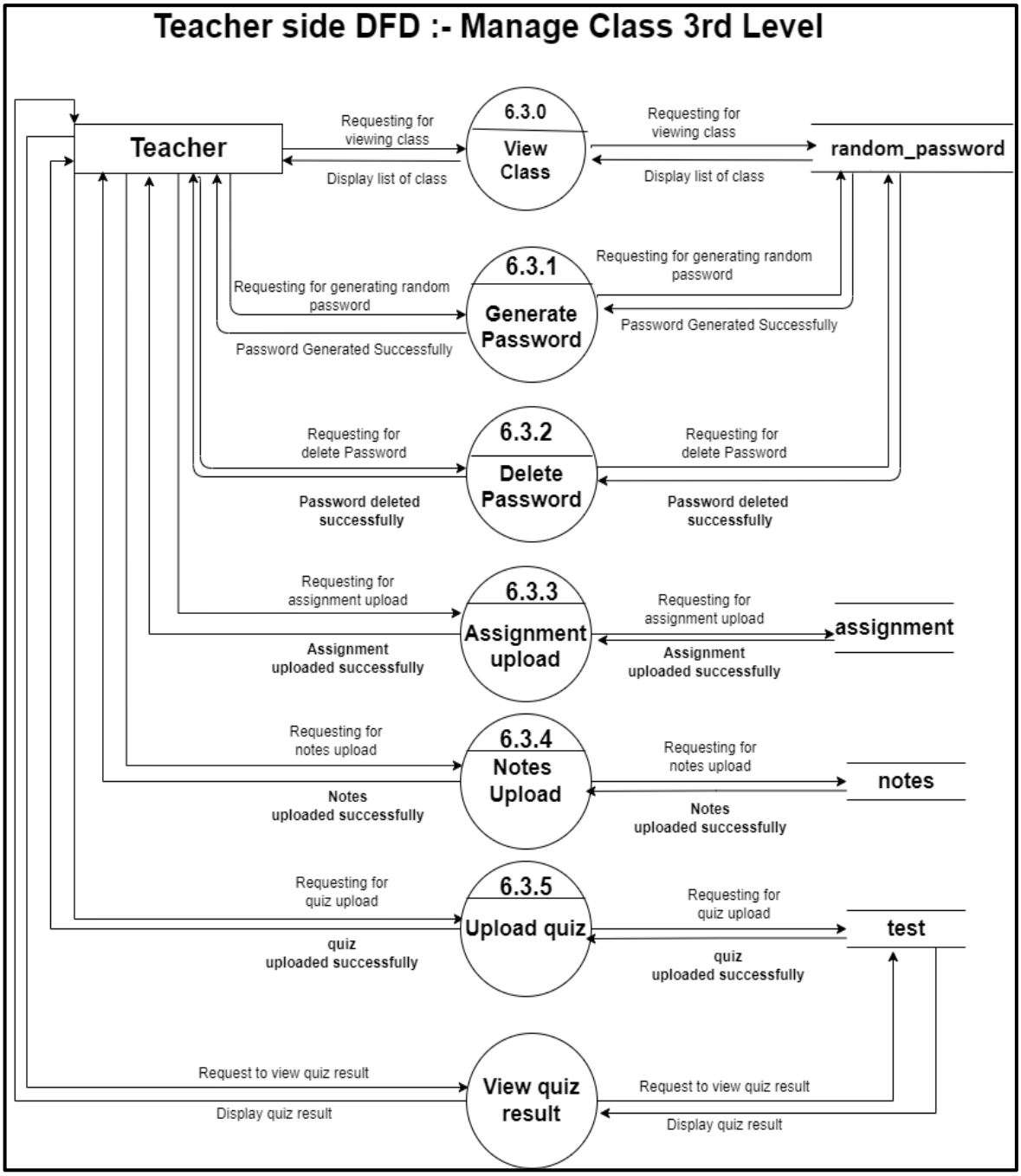
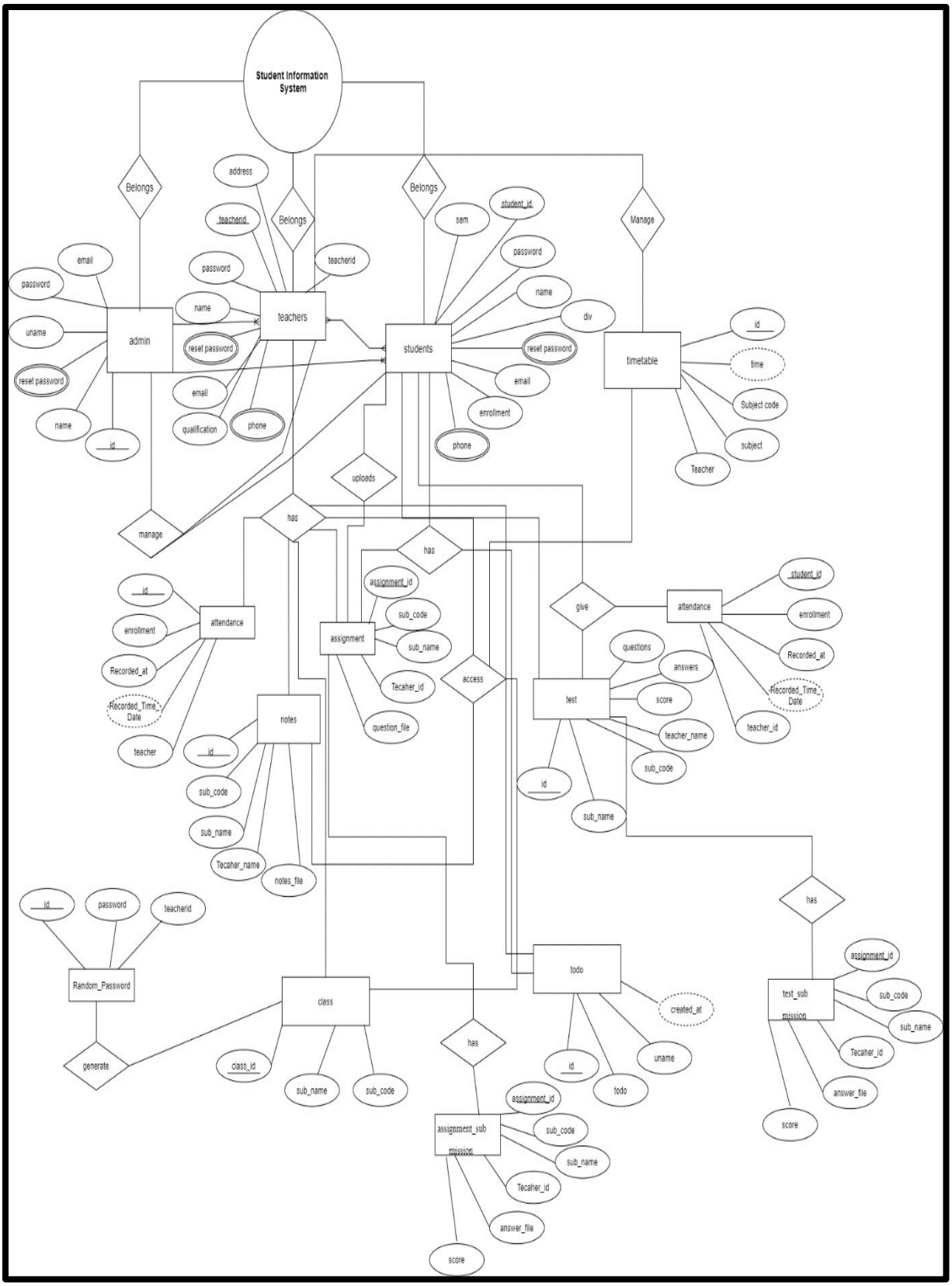


Figure 5.4.1. Teacher Side DFD: Manage Class 3nd Level

ER diagram example :



1.5.Description of E-R Diagram:

This ER (Entity Relationship) Diagram represents the model of Student Information System. The entity-relationship diagram of Student Information System shows all the visual instrument of database tables and the relations between Admin, Teacher, Students, Timetable, Attendance, to-do etc. It used structure data and to define the relationships between structured data groups of Student Information System functionalities.

This ER (Entity Relationship) Diagram represents the model of Student Information System. The entity-relationship diagram of Student Information System shows all the visual instrument of database tables and the relations between Admin, Teacher and Students. It used structure data and to define the relationships between structured data groups of Student Information System functionalities.

The main entities of the Student Information System are Admin, Teacher, Students, Timetable, Attendance, Assignment, Notes, Test, Class, Random_password, Todo etc.

Student Information System belongs to Admin, Teacher and Students entities. In which admin can manage teacher and student's entities and teacher has attendance, assignments, class, notes, test and todo. In which class can generate random_password. On the other hand students can upload assignment, it has a access on notes, todo and timetable, it can give test and attendance and student entity has assignment and todo list.

☑ Practical: 4 Project Planning and Estimation (W5HH & COCOMO) (Use the W5HH principle to plan a project (Who, What, When, Where, How, how much). Perform software effort estimation using Basic COCOMO Model)

Project Planning and Estimation – Online Food Ordering System

1. W5HH Analysis

W5HH is a simple framework to plan **who, what, when, where, how, and how much** for the project.

W5HH	Description for Online Food Ordering System
Who	- Project Manager: Overall coordination - Developers: Web/Mobile App, Backend API - UI/UX Designer: Interface design - Database Designer: DB design - Testers: Functional & performance testing - Admin: System management
What	- Develop Online Food Ordering System (Web & Mobile) - Features: Registration/Login, Browse Restaurants, Menu Management, Cart, Order, Payment, Delivery Tracking, Admin Panel
When	Estimated Timeline: 12 weeks - Week 1–2: Requirement Analysis & SRS - Week 3–4: UI/UX Design - Week 5–8: Development (Frontend + Backend + DB) - Week 9–10: Integration & Testing - Week 11–12: Deployment & Documentation
Where	Development: Office / Online Remote Deployment: Web Server / Cloud Platform (AWS / Azure) Access: Customer Mobile App / Web Browser
How	- SDLC Approach: Agile / Iterative - Tools: • Frontend: ReactJS / Android / iOS • Backend: NodeJS / Django / Java Spring • Database: MySQL / PostgreSQL • Version Control: GitHub - Testing: Unit, Integration, User Acceptance Testing
How Much	- Resources: 5–6 team members - Estimated Cost: (Approx based on man-hours & infrastructure) - Hardware/Software: Servers, development tools, cloud services

2. Project Estimation Using COCOMO Model

COCOMO (Constructive Cost Model) estimates **effort, time, and cost** for software projects based on project size in **KLOC (thousands of lines of code)**.

Step 1: Estimate Project Size

- Assume **Web + Mobile OFOS**:
  - Frontend: 2000 LOC
  - Backend: 3000 LOC
  - Database scripts: 500 LOC
  - **Total ≈ 5500 LOC = 5.5 KLOC**

Step 2: Choose COCOMO Model

- Use **Basic COCOMO** (for small-medium project)
- Project Type: **Organic** (small team, well-understood application)

Basic COCOMO Formula:

1. **Effort (Person-Months)** = a × (KLOC)^b
2. **Development Time (Months)** = c × (Effort)^d

Coefficients for Organic projects:

Parameter	Value
a	2.4
b	1.05
c	2.5
d	0.38

Step 3: Calculate Effort

$$\text{Effort (PM)} \approx 2.4 \times 6.00$$

Step-by-step calculation:

1. $5.5^{1.05} \approx 5.5 \times (5.5^{0.05})$
2. $5.5^{0.05} \approx 1.086$

3. $5.5^{1.05} \approx 5.5 \times 5.5^{0.05} \approx 5.5 \times 1.086$
4. **Effort ≈ 2.4 × 5.973 ≈ 14.33 Person-Months**

Step 4: Calculate Development Time

$$\text{Time (Months)} = 2.5 \times (14.33)^{0.38}$$

Step-by-step:

1. $14.33^{0.38} \approx 2.74$
2. **Time ≈ 2.5 × 2.74 ≈ 6.85 Months**

Approx 7 months for the full project, considering small team efficiency, Agile iterations, and overheads.

Step 5: Team Size Estimation

$$Team \frac{¿\, Effort}{Development\ Time} = \frac{14.33}{6.85} \approx 2.1 \approx 3 people$$

Recommended: 3–5 team members including frontend, backend, and database developers.

---

3. Summary Table – Project Planning

Parameter	Value
Project Size	5.5 KLOC
Model Used	COCOMO Basic (Organic)
Effort	14.33 Person-Months
Development Time	6.85 Months (~7 months)
Team Size	3–5 Members
W5HH Highlights	Who: Dev team, Admin What: OFOS features When: 12 weeks (practical plan) Where: Office/Remote How: Agile + Web & Mobile How Much: Cost depends on resources & infrastructure

☑ Key Notes:

- For practical submissions, you can **combine W5HH table with COCOMO table**.
- You can also **adjust KLOC estimates** based on detailed module sizing.
- For bigger projects, **Intermediate or Detailed COCOMO** can be used to include cost drivers.

Project Planning Document – Online Food Ordering System

1. Project Overview

Project Title: Online Food Ordering System (OFOS)
Objective: To design and implement a system for customers to order food online, restaurants to manage orders and menus, and admins to manage users and reports.
Project Type: Web + Mobile Application

---

2. W5HH Analysis

W5HH	Description
Who	Project Manager, Frontend Developer, Backend Developer, UI/UX Designer, Database Designer, Testers, Admin
What	Features: Registration/Login, Browse Restaurants, Menu Management, Cart, Place Order, Payment Integration, Delivery Tracking, Admin Panel
When	**Timeline:** 12 Weeks • Week 1–2: Requirement Analysis & SRS • Week 3–4: UI/UX Design • Week 5–8: Development (Frontend + Backend + DB) • Week 9–10: Integration & Testing • Week 11–12: Deployment & Documentation
Where	Development: Office / Remote Deployment: Cloud / Web Server Access: Mobile App & Web Browser
How	SDLC: Agile / Iterative Tools: ReactJS / Android / iOS, NodeJS / Django / Java Spring, MySQL / PostgreSQL, GitHub, Testing Tools
How Much	Team: 5–6 members Cost: Based on man-hours, development tools, server, cloud services

3. Project Estimation – COCOMO Model

Step 1: Project Size

- Frontend: 2000 LOC
- Backend: 3000 LOC
- Database scripts: 500 LOC
- **Total ≈ 5500 LOC = 5.5 KLOC**

Step 2: COCOMO (Basic, Organic)

Parameter	Value
a	2.4
b	1.05
c	2.5
d	0.38

Step 3: Effort Calculation

$$\text{Effort (PM)} = 2.4 \times (5.5)^{1.05} \approx 14.33 \text{ Person-Months}$$

Step 4: Development Time

$$\text{Time (Months)} = 2.5 \times (14.33)^{0.38} \approx 6.85 \approx 7 \text{ Months}$$

Step 5: Team Size

$$\text{Team} \frac{¿\ \text{Effort}}{\text{Development Time}} = \frac{14.33}{6.85} \approx 3\text{–}5 \text{ members}$$

4. Gantt Chart – Project Timeline (12 Weeks)

Week	Activity
1–2	Requirement Analysis & SRS Preparation
3–4	UI/UX Design, Prototype Screens
5–6	Frontend Development (Customer Module, Restaurant Module)
7–8	Backend Development & Database Integration
9	Integration Testing
10	System Testing & Bug Fixes

Week	Activity
11	Deployment Preparation
12	Deployment & Documentation Submission

Note: Each week represents **one iteration/sprint** in Agile approach.

5. Team Allocation

Team Member	Role	Responsibilities
Project Manager	PM	Plan, coordinate, track project progress
Frontend Developer	Developer	Develop web/mobile UI, integrate APIs
Backend Developer	Developer	API development, database connectivity, server-side logic
UI/UX Designer	Designer	Design app interface, user experience workflow
Database Designer	DBA	Design schema, manage database, optimize queries
Tester	QA	Unit, Integration, System, and User Acceptance Testing
Admin	System Admin	Deploy system, monitor performance, maintain servers

6. Resource Requirements

Resource Type	Description
Hardware	Developer PCs, Server/Cloud instance for deployment
Software	IDEs (VS Code, Android Studio), Database (MySQL/PostgreSQL), Version Control (GitHub)
Network	Internet connection, VPN (if remote work)
Miscellaneous	Project documentation tools, Testing tools

7. Cost Estimation (Optional)

Item	Cost Estimate (USD)
Human Resources	5–6 members × 7 months × avg salary per PM ≈ 20,000–25,000
Servers / Cloud	Deployment & hosting ≈ 200–500
Tools / Software	IDEs, Database, Testing Tools ≈ 300
Miscellaneous	Documentation, contingency ≈ 200
Total Estimate	~21,000–26,000

8. Summary

- **W5HH** ensures project scope, team, methods, and resources are well-defined.
- **COCOMO estimation** provides effort, time, and team size for planning.
- **Gantt chart** outlines weekly tasks for scheduling.
- **Team allocation** defines responsibilities and accountability.
- Ready for development, testing, and deployment with clear timeline and resources.

☑ Practical: 5 Risk Management and Quality Planning Report (Identify at least 5 major project risks, analyze their impact and probability, and propose mitigation strategies. Include a short quality plan covering QA and QC activities).

Risk Management and Quality Planning Report – Online Food Ordering System

1. Introduction

The **Online Food Ordering System (OFOS)** allows customers to browse restaurants, place orders, make payments, and track deliveries. Restaurants can manage menus and orders, while admins manage users and reports.

This report identifies **project risks**, evaluates their impact, and defines **quality planning** to ensure a reliable, maintainable, and efficient system.

2. Risk Management

Risk management involves **identifying, analyzing, and mitigating risks** throughout the software development lifecycle.

2.1 Risk Identification

Risk ID	Risk Description	Category
R1	Delay in requirement gathering	Schedule Risk
R2	Incorrect or incomplete SRS	Requirement Risk
R3	Development delays due to unfamiliar technology	Technical Risk
R4	Server downtime / hosting issues	Operational Risk
R5	Payment gateway integration failure	Technical / Financial Risk
R6	Security vulnerabilities (data breach, SQL injection)	Security Risk

Risk ID	Risk Description	Category
R7	Poor UI/UX leading to low adoption	User Acceptance Risk
R8	Data loss due to database failure	Operational Risk
R9	Communication gaps in team (remote work)	Management Risk
R10	Regulatory compliance issues (PCI DSS, GDPR)	Legal / Compliance Risk

2.2 Risk Analysis and Mitigation

Risk ID	Probability	Impact	Mitigation Strategy
R1	Medium	High	Conduct regular meetings, clear timelines, monitor progress
R2	Medium	High	Perform peer reviews of SRS, involve stakeholders
R3	Medium	Medium	Provide training, adopt Agile iterative development
R4	Low	High	Use reliable cloud servers, maintain backup servers
R5	Medium	High	Test payment integration thoroughly, use sandbox mode
R6	Medium	High	Implement secure coding practices, encryption, regular security testing
R7	Medium	Medium	Conduct usability testing, prototype feedback
R8	Low	High	Implement regular database backups, replication
R9	Medium	Medium	Use project management tools (Jira, Trello), daily standups
R10	Low	High	Consult legal experts, follow compliance guidelines

2.3 Risk Monitoring

- Maintain a **Risk Register** for tracking risks, status, and mitigation.
- Conduct **weekly risk review meetings**.

- Assign **risk owners** for each critical risk.

---

3. Quality Planning

Quality planning ensures the software meets **functional and non-functional requirements**, is reliable, maintainable, and satisfies end-users.

3.1 Quality Objectives

Objective	Description
Functionality	All features (registration, menu management, ordering, payment, tracking) must work correctly.
Reliability	System should handle 100+ concurrent users with minimal downtime.
Usability	Intuitive UI/UX, responsive design for web and mobile.
Security	Secure login, encrypted data storage, secure payment transactions.
Performance	Fast response time (<3 seconds for key actions like browsing menu or placing order).
Maintainability	Modular design using MVC/OOA approach, clear code documentation.
Compliance	Follow PCI DSS for payments, GDPR for user data privacy.

3.2 Quality Assurance (QA) Activities

QA Activity	Description
Code Review	Peer review of all code modules to detect errors early.
Unit Testing	Test individual components (frontend, backend, database) separately.
Integration Testing	Verify combined modules interact correctly.
System Testing	Full system test for end-to-end functionality.
User Acceptance Testing (UAT)	End-users test the system for usability and functionality.
Security Testing	Test for vulnerabilities, penetration testing.

QA Activity	Description
Performance Testing	Load and stress testing to measure system performance.

3.3 Quality Standards and Metrics

Metric	Target
Defect Density	< 1 defect per 1000 LOC
Test Coverage	≥ 90% code coverage in unit tests
Response Time	≤ 3 seconds for main actions
Availability	≥ 99% uptime for web/mobile
User Satisfaction	≥ 85% positive feedback from UAT

3.4 Tools for Quality Assurance

- **Version Control:** GitHub / GitLab
- **Testing Tools:** Selenium, JUnit, Postman
- **Project Management:** Jira / Trello
- **Security Testing:** OWASP ZAP, Burp Suite
- **Performance Testing:** Apache JMeter

---

4. Conclusion

- Proper **risk management** minimizes potential delays, cost overruns, and technical failures.
- Effective **quality planning** ensures the OFOS system meets user expectations, is secure, reliable, and maintainable.
- Combining **risk monitoring and quality assurance** creates a robust software development process ready for deployment.

☑ Practical: 6 Use Case Modeling and Diagrams (Prepare Use Case Descriptions and corresponding Use Case Diagrams for the SRS developed earlier. Identify main actors, use cases, and relationships).

Here's a **complete Use Case Modeling and Diagrams package** for your **Online Food Ordering System (OFOS)**, combining **use case descriptions** and a **graphical UML Use Case Diagram**.

---

1. Main Actors

Actor	Description
Customer	End-user who browses restaurants, places orders, makes payments, and tracks delivery.
Restaurant Staff	Manages menu, receives and processes orders.
Delivery Person	Updates order status during delivery.
Admin	Manages system users, restaurants, and generates reports.

2. Use Case Descriptions

UC01 – User Registration

Field	Description
Use Case ID	UC01
Actor	Customer
Pre-condition	User does not have an account
Post-condition	New account created
Description	Customer enters personal details to register for the

Field	Description
	system
Normal Flow	Open registration → enter details → validate → create account → confirmation displayed
Alternate Flow	Email already exists → show error
Exception Flow	Network failure → registration fails

UC02 – User Login

Field	Description
Use Case ID	UC02
Actor	Customer
Pre-condition	Account exists
Post-condition	User authenticated
Normal Flow	Enter email/password → system validates → login successful
Alternate Flow	Wrong password → show error
Exception Flow	Account locked after multiple failed attempts

UC03 – Browse Restaurants & Menu

Field	Description
Use Case ID	UC03
Actor	Customer
Pre-condition	Logged in
Post-condition	Customer views restaurant and menu

Field	Description
Normal Flow	Select restaurant → view menu → apply filters
Alternate Flow	No restaurants available → show message

UC04 – Add Item to Cart

Field	Description
Use Case ID	UC04
Actor	Customer
Pre-condition	Menu loaded
Post-condition	Item added to cart
Normal Flow	Select item → choose quantity → add to cart
Alternate Flow	Item unavailable → show message

UC05 – Place Order & Payment

Field	Description
Use Case ID	UC05
Actor	Customer
Pre-condition	Cart contains items
Post-condition	Order placed and recorded
Normal Flow	Review cart → choose payment → confirm order → system records order → sent to restaurant
Alternate Flow	Payment fails → retry
Exception	Restaurant offline → order canceled

Field	Description
Flow	

UC06 – Accept/Reject Order

Field	Description
Use Case ID	UC06
Actor	Restaurant Staff
Pre-condition	Order submitted
Post-condition	Order status updated
Normal Flow	Receive order → accept/reject → system updates status
Alternate Flow	Rejected → notify customer

UC07 – Update Order Status (Delivery)

Field	Description
Use Case ID	UC07
Actor	Delivery Person
Pre-condition	Order accepted
Post-condition	Status updated
Normal Flow	Pick up order → update status (Out for delivery → Delivered)
Exception Flow	Delivery failed → mark as failed

UC08 – Manage Menu

Field	Description
Use Case ID	UC08

Field	Description
Actor	Restaurant Staff
Pre-condition	Restaurant active
Post-condition	Menu updated
Normal Flow	Add/update/delete menu items → save changes
Exception Flow	Invalid input → show error

UC09 – Admin Operations

Field	Description
Use Case ID	UC09
Actor	Admin
Pre-condition	Admin logged in
Post-condition	System data updated
Normal Flow	Manage users/restaurants → generate reports
Exception Flow	Unauthorized access blocked

UC10 – Track Order

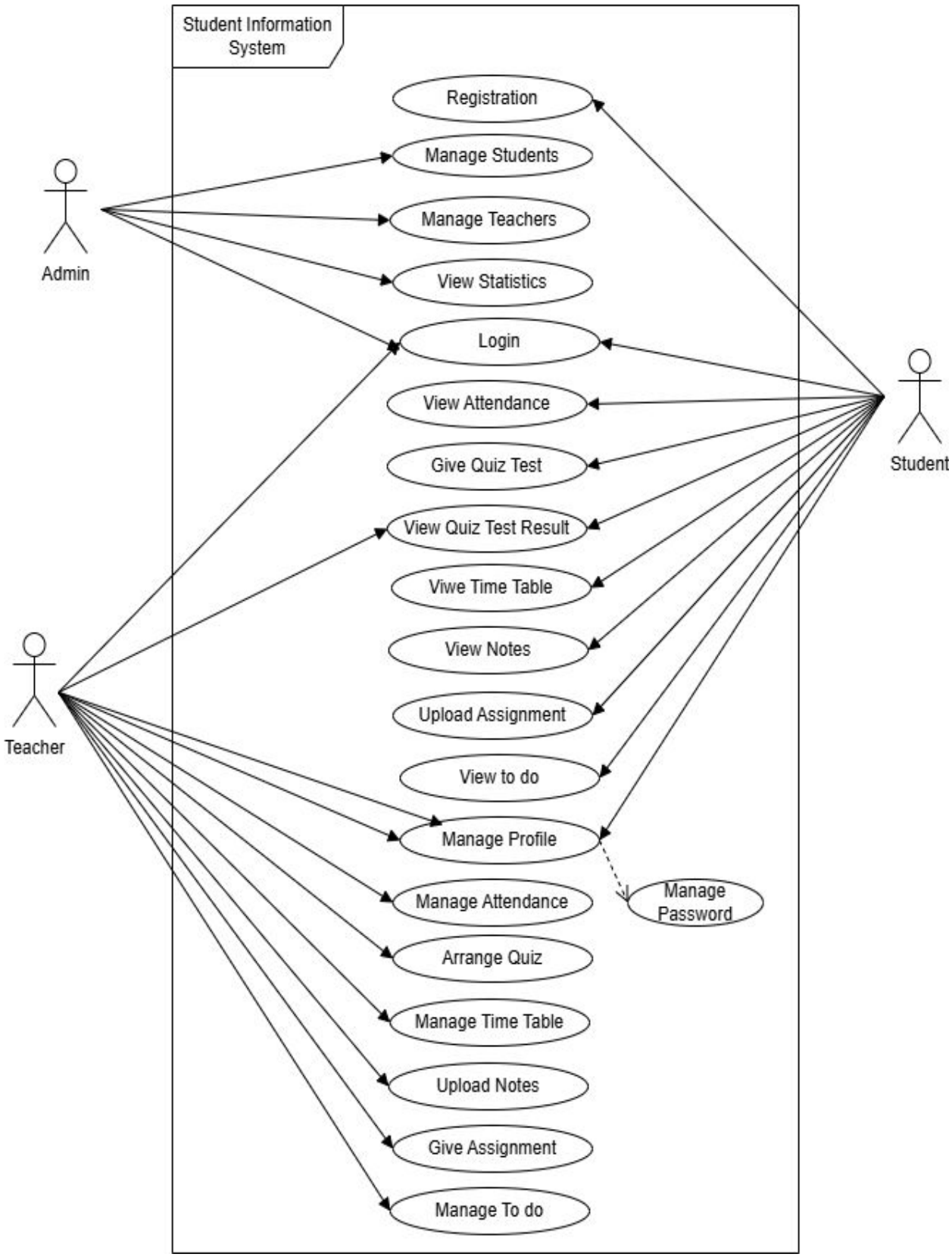
Field	Description
Use Case ID	UC10
Actor	Customer
Pre-condition	Customer has active order
Post-condition	Customer views order status

Field	Description
Normal Flow	Open “Track Order” → system fetches status → display
Exception Flow	Network issue → cannot fetch status

3. Use Case Diagram (Graphical)

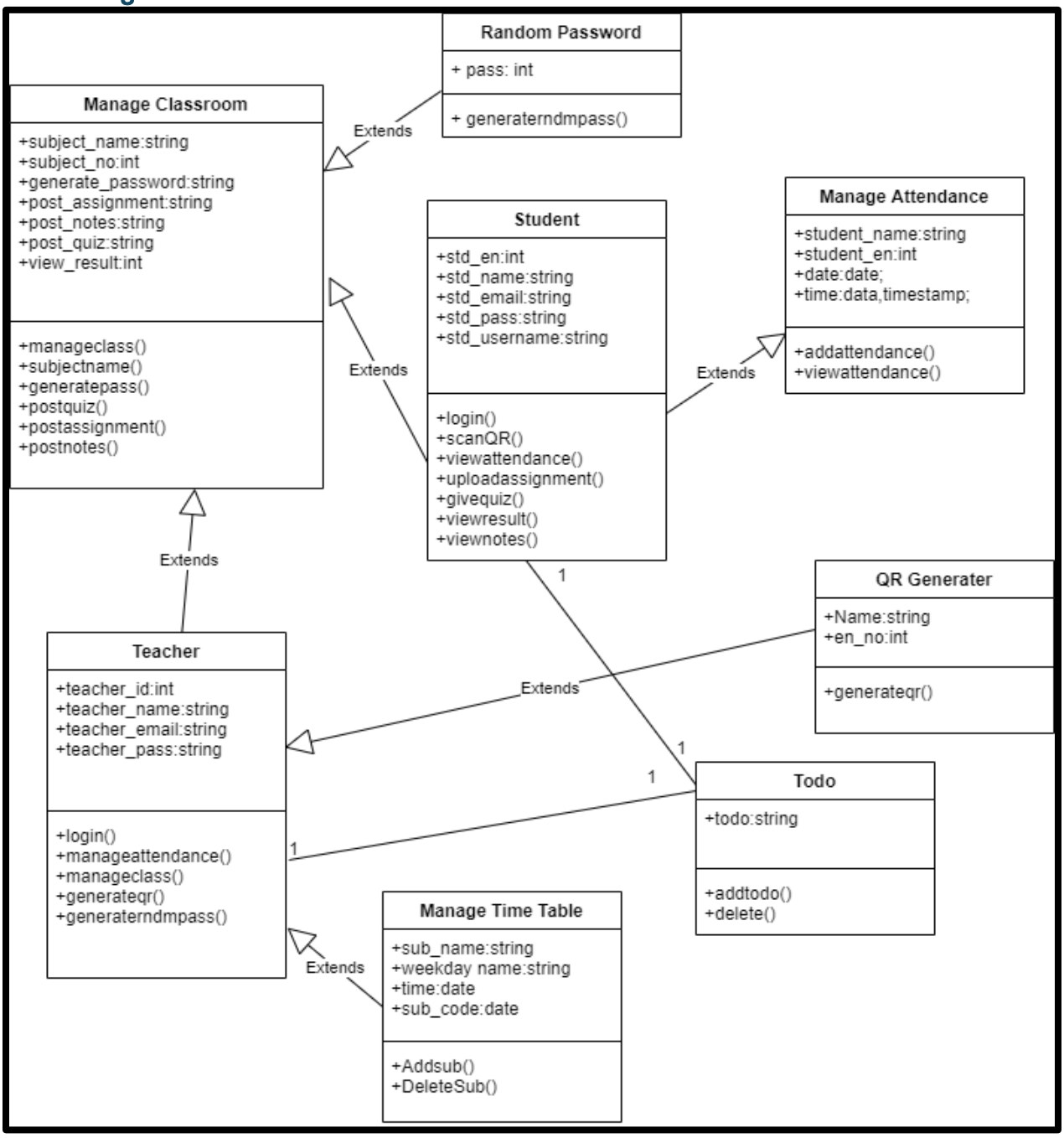
Diagram Description:

- Actors: Customer, Restaurant Staff, Delivery Person, Admin
- Customer → Registration/Login, Browse Menu, Add to Cart, Place Order, Payment, Track Order
- Restaurant Staff → Accept/Reject Order, Manage Menu
- Delivery Person → Update Order Status
- Admin → Manage Users/Restaurants, Generate Reports

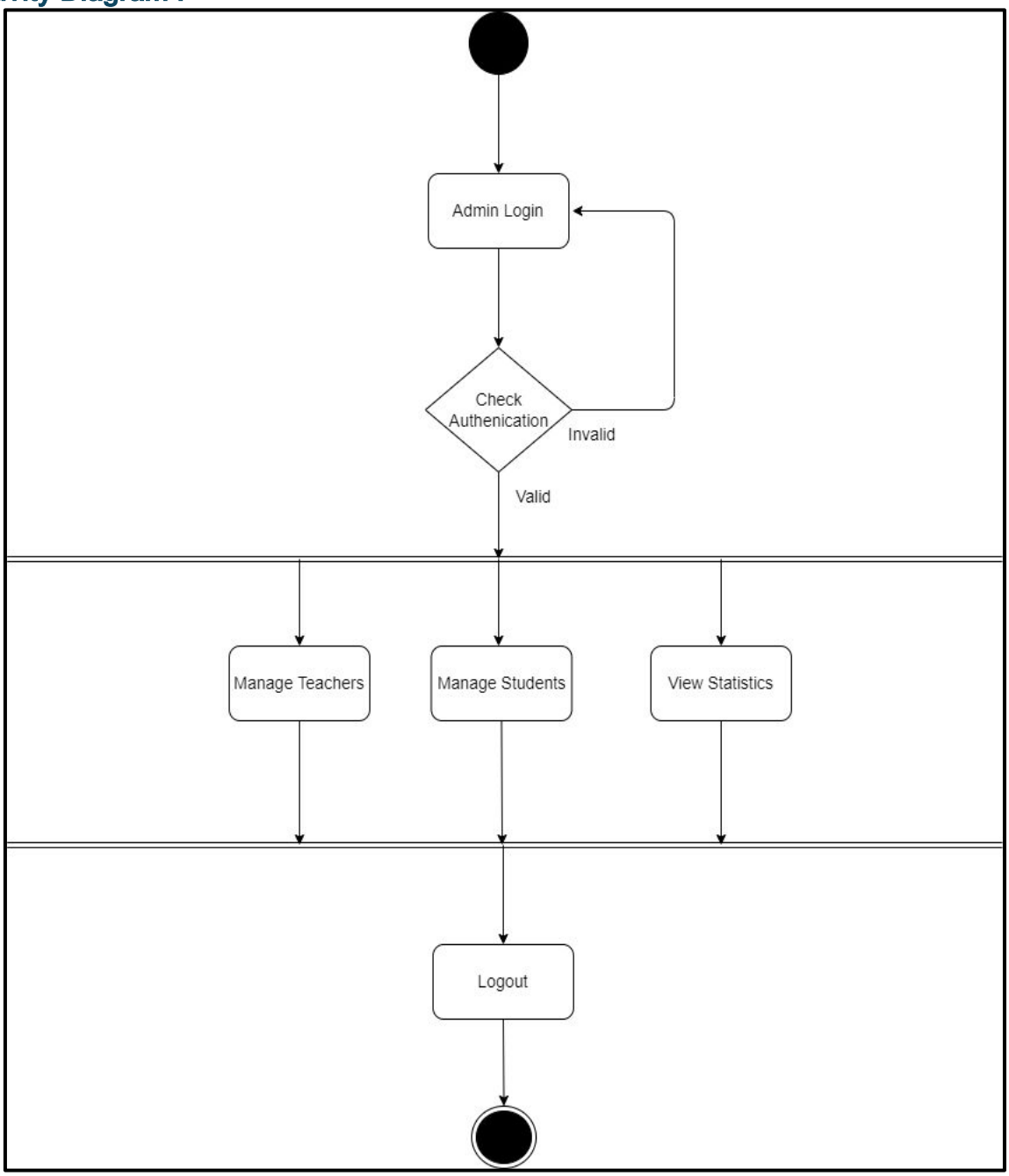

Student Information System
Registration
Manage Students
Manage Teachers
View Statistics
Login
View Attendance
Give Quiz Test
View Quiz Test Result
Viwe Time Table
View Notes
Upload Assignment
View to do
Manage Profile
Manage Password
Manage Attendance
Arrange Quiz
Manage Time Table
Upload Notes
Give Assignment
Manage To do
Admin
Teacher
Student

☑ Practical: 7 Design UML Diagrams (Class, Sequence, and Activity).

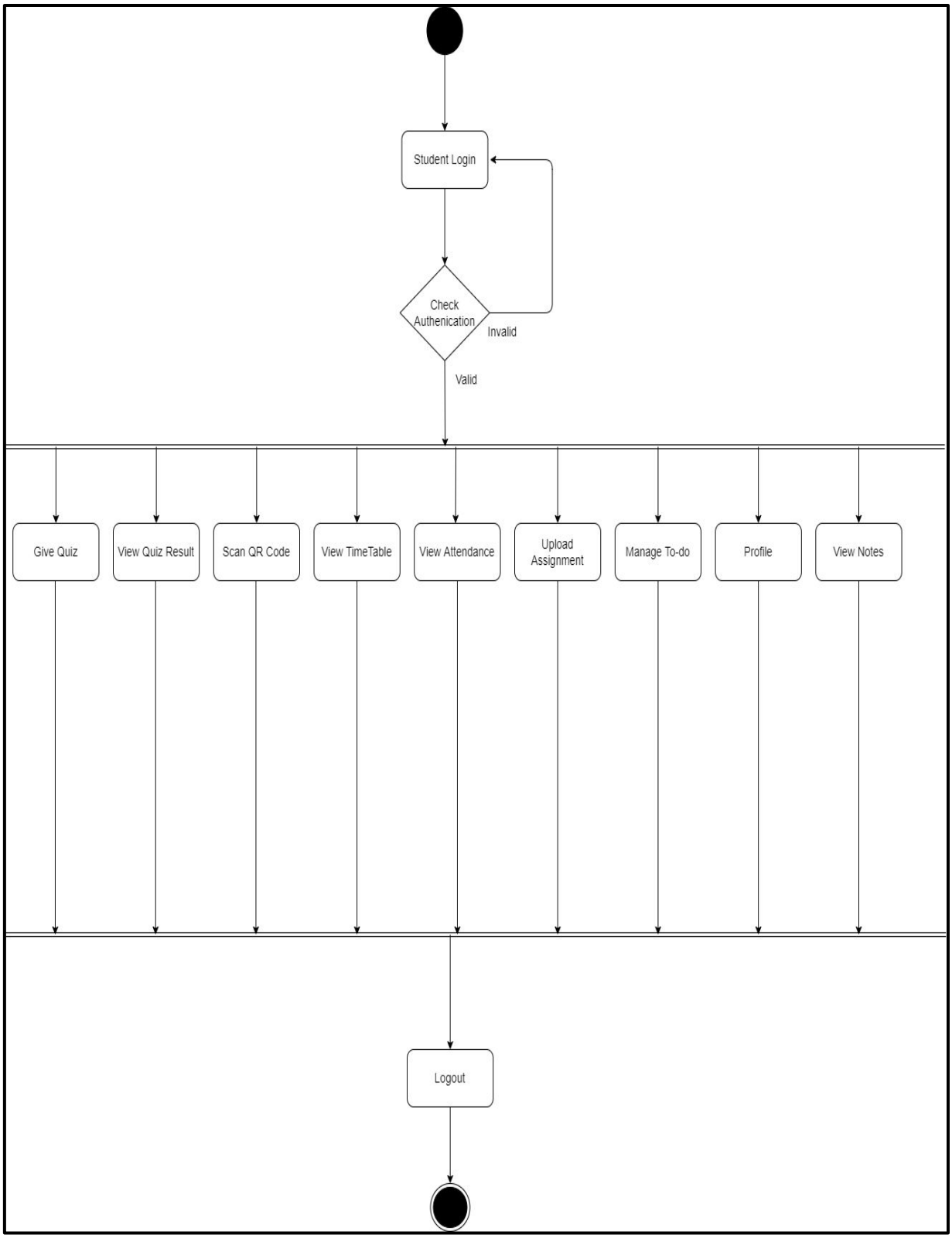
Class Diagram:



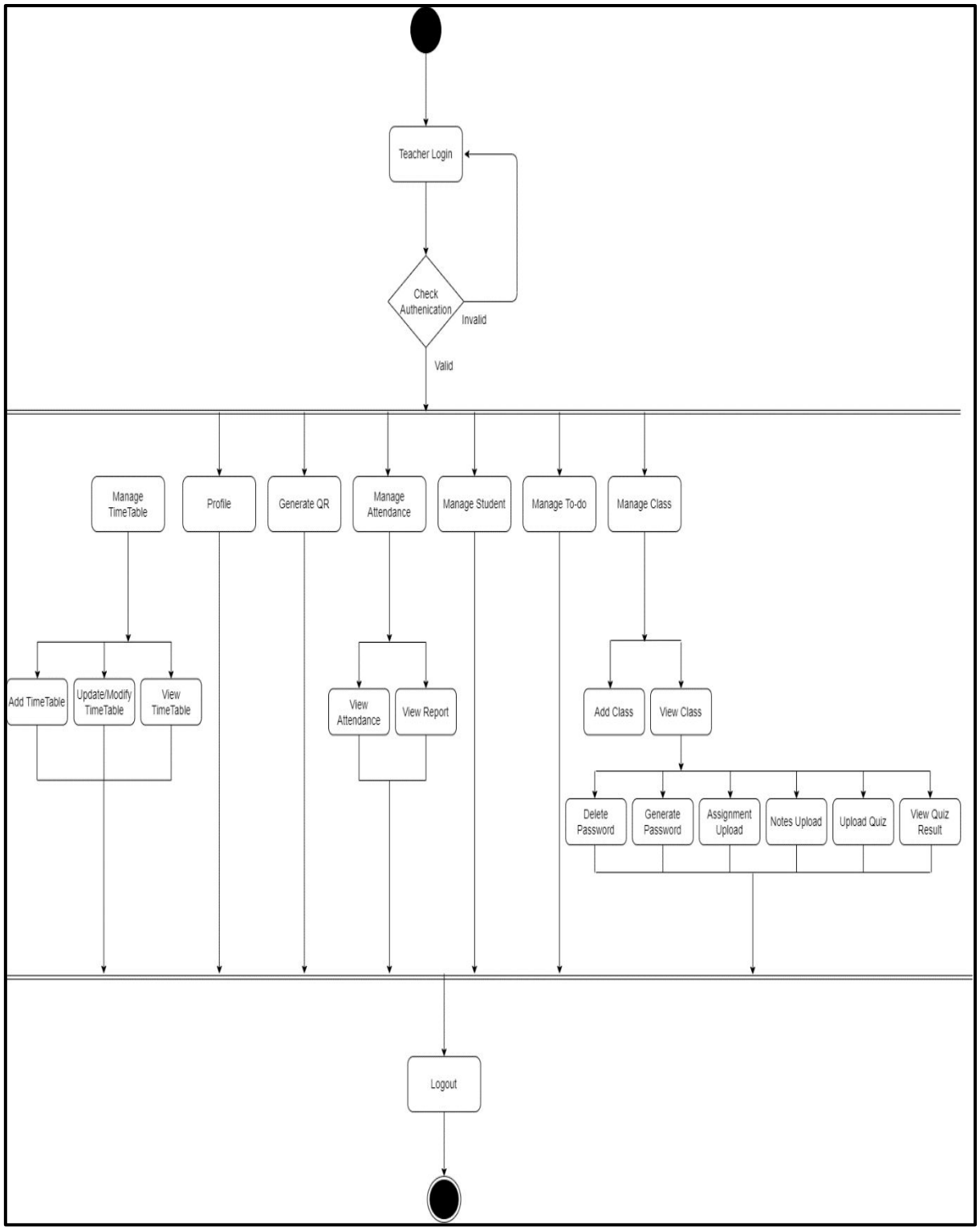
Activity Diagram :



Admin Activity Diagram

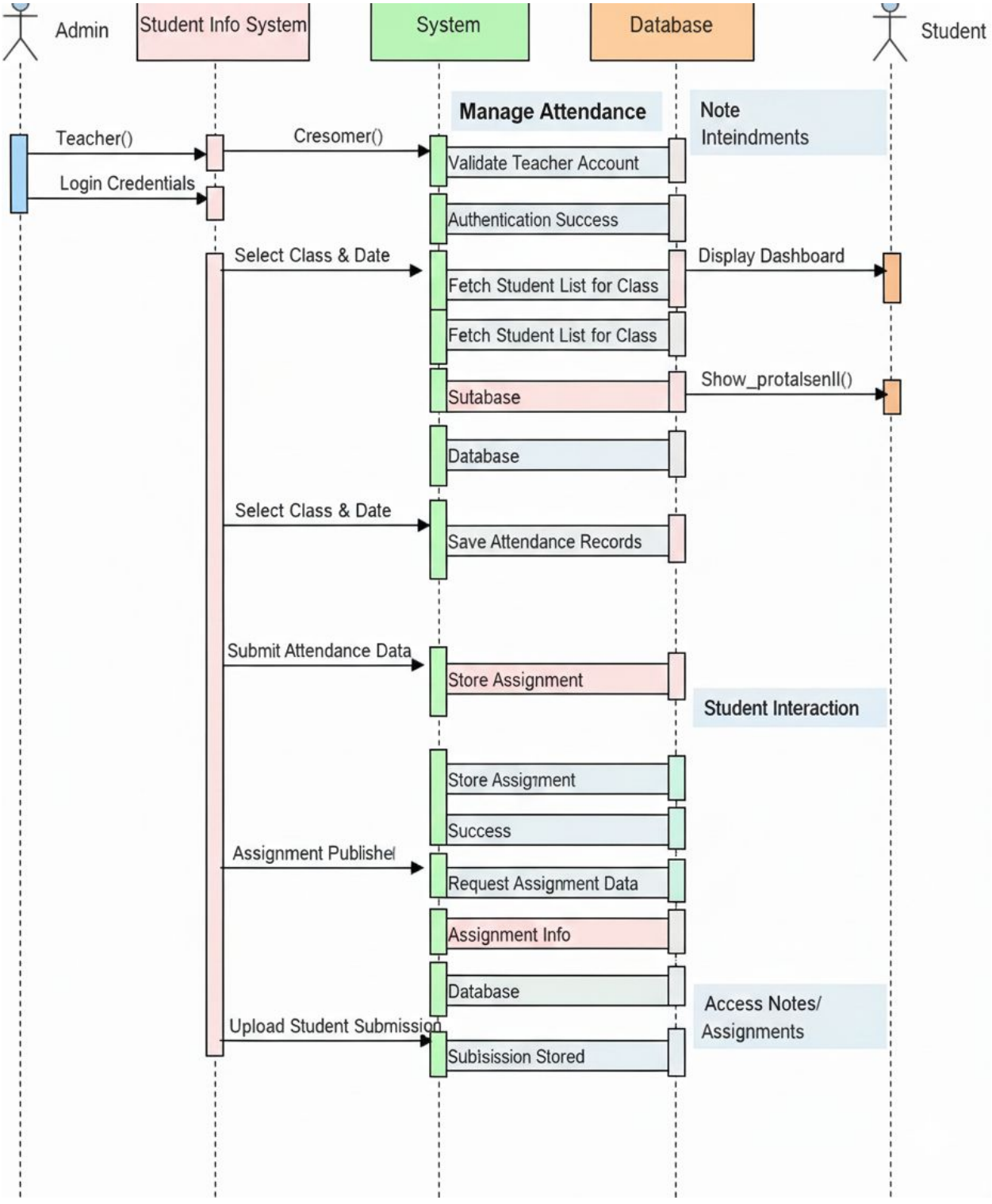


Student Activity Diagram



Teacher Activity Diagram

Sequence Diagram :



☑ Practical: 8 Software Testing – Test Case Design and Verification (Write at least 10 test cases for one module using Black Box and White Box techniques. Indicate input, expected output, and actual output.)

Module: *Login Function for an Online Banking System*
Functionality: The user enters a username and password to log in. If the credentials are correct, access is granted; otherwise, an error message is displayed.

Black Box Test Cases (Functional Testing)

Here, we focus on **inputs and outputs**, without considering the internal code.

TC No	Test Case Description	Input	Expected Output	Actual Output	Pass / Fail
1	Valid login credentials	Username: user1, Password: Pass@123	Login successful, redirect to dashboard	-	-
2	Invalid password	Username: user1, Password: wrongpass	Error message: “Invalid username or password”	-	-
3	Invalid username	Username: wronguser, Password: Pass@123	Error message: “Invalid username or password”	-	-
4	Empty username	Username: , Password: Pass@123	Error message: “Username cannot be empty”	-	-
5	Empty password	Username: user1, Password:	Error message: “Password cannot be empty”	-	-
6	SQL Injection	Username: ' OR	Error message:	-	-

TC No	Test Case Description	Input	Expected Output	Actual Output	Pass / Fail
	attempt	1=1 -- , Password: any	"Invalid username or password"		
7	Special characters in username	Username: user! @#, Password: Pass@123	Error message: "Invalid characters in username"	-	-
8	Password case sensitivity	Username: user1, Password: pass@123	Error message: "Invalid username or password"	-	-
9	Maximum length username	Username: 50-char string, Password: Pass@123	Login successful or error if max limit exceeded	-	-
10	Maximum length password	Username: user1, Password: 50-char string	Login successful or error if max limit exceeded	-	-

White Box Test Cases (Structural / Code-Based Testing)

Here, we test **internal logic, branches, and loops**. Assume the login module code contains:

```
if(username == "") { error("Username cannot be empty"); }
else if(password == "") { error("Password cannot be empty"); }
else if(validateUser(username, password)) { redirectToDashboard(); }
else { error("Invalid username or password"); }
```

TC No	Test Case Description	Input	Expected Output	Path Tested
1	Empty username	Username: , Password: Pass@123	Error: Username cannot be empty	`if(username=="")`
2	Empty password	Username: user1, Password:	Error: Password cannot be empty	`else if(password=="")`

TC No	Test Case Description	Input	Expected Output	Path Tested
3	Valid credentials	Username: user1, Password: Pass@123	Redirect to dashboard	`else if(validateUser)`
4	Invalid username	Username: wrong, Password: Pass@123	Error: Invalid username or password	`else` branch
5	Invalid password	Username: user1, Password: wrong	Error: Invalid username or password	`else` branch
6	Both username and password empty	Username: , Password:	Error: Username cannot be empty	First `if` takes precedence
7	SQL Injection attempt	Username: ' OR 1=1 -- , Password: any	Error: Invalid username or password	`validateUser()` branch
8	Case sensitivity test	Username: User1, Password: pass@123	Error: Invalid username or password	`validateUser()` branch
9	Boundary test for username length	Username: 50-char string, Password: Pass@123	Pass or error based on max length	Input validation path
10	Boundary test for password length	Username: user1, Password: 50-char string	Pass or error based on max length	Input validation path

☑ Notes:

- **Black Box** focuses on expected behavior regardless of code paths.
- **White Box** focuses on testing all possible **branches and paths** in the code.
- Actual output should be filled after executing the test cases.

☑ Practical: 9 Software Testing Levels and Execution Report (Demonstrate different testing levels (Unit, Integration, System, Acceptance) conceptually using one module. Prepare a short testing report).

Module: Login Function for Online Banking System

Functionality: User enters a username and password. If correct, access is granted; otherwise, an error message is displayed.

---

1. Testing Levels

1. Unit Testing

- **Purpose:** Test individual components or functions of the module.
- **Scope in Login Module:**
  - `validateUsername(username)` → **Checks if username is non-empty and valid.**
  - `validatePassword(password)` → **Checks password constraints.**
  - `authenticateUser(username, password)` → **Checks credentials in the database.**
- **Example Test Cases:**
  - **Username empty → should return error.**
  - **Password empty → should return error.**
  - **Correct username and password → should authenticate successfully.**

---

2. Integration Testing

- **Purpose:** Test interactions between integrated modules.
- **Scope in Login Module:**
  - Login module integrated with **Database Module**.
  - Verify correct fetching and matching of user credentials.

- **Example Test Cases:**
  - Enter valid credentials → system retrieves user data from DB → login succeeds.
  - Enter invalid credentials → system checks DB → returns error.

---

3. System Testing

- **Purpose:** Test the complete system to ensure it meets requirements.
- **Scope in Login Module:**
  - Full banking system login workflow including UI, backend authentication, and security checks.
- **Example Test Cases:**
  - Login with valid credentials → redirect to dashboard.
  - Login with SQL injection attempt → system rejects input.
  - Login with incorrect credentials → displays proper error message.

---

4. Acceptance Testing

- **Purpose:** Ensure system meets business requirements and is ready for end users.
- **Scope in Login Module:**
  - Conducted by **end users or stakeholders**.
- **Example Test Cases:**
  - Users can log in successfully with correct credentials.
  - System prevents unauthorized access.
  - System displays user-friendly error messages for invalid inputs.

---

2. Sample Testing Execution Report (Short Format)

Test Level	Test Case Description	Input	Expected Output	Actual Output	Status
Unit	Empty username	Username: , Password: Pass@123	Error: Username cannot be empty	Error: Username cannot be empty	Pass

Test Level	Test Case Description	Input	Expected Output	Actual Output	Status
Unit	Empty password	Username: user1, Password:	Error: Password cannot be empty	Error: Password cannot be empty	Pass
Integration	Valid credentials, DB connected	Username: user1, Password: Pass@123	Login successful	Login successful	Pass
Integration	Invalid credentials, DB connected	Username: user1, Password: wrong	Error: Invalid username or password	Error: Invalid username or password	Pass
System	SQL injection attempt	Username: ' OR 1=1 -- , Password: any	Error: Invalid username or password	Error: Invalid username or password	Pass
System	Valid credentials	Username: user1, Password: Pass@123	Redirect to dashboard	Redirect to dashboard	Pass
Acceptance	End user login	Username: user1, Password: Pass@123	User can access dashboard	User can access dashboard	Pass
Acceptance	Error handling	Username: , Password:	User-friendly error message	User-friendly error message	Pass

✅ Notes:

- **Execution Report** shows each test case's **expected vs actual output**.
- Status is marked as **Pass/Fail** after testing.
- This format is concise and suitable for documentation purposes.

☑ Practical: 10 Demonstration of CASE Tools (Explore a CASE tool such as StarUML, Visual Paradigm, or Rational Rose. Demonstrate how it supports software design (UML creation, code generation, or documentation).

---

1. Introduction to CASE Tools

CASE (Computer-Aided Software Engineering) tools help software developers **model, design, and document software systems efficiently**. They provide:

- **UML diagram creation** (Class, Sequence, Activity, etc.)
- **Code generation** (from UML to programming languages)
- **Documentation support** (export diagrams, reports)
- **Consistency checks** (ensures model correctness)

Popular CASE tools include: **StarUML, Visual Paradigm, Rational Rose, Enterprise Architect**.

---

2. Demonstration Using StarUML

Step 1: Create a New Project

1. Open StarUML.
2. Click **File → New Project → UML**.
3. Choose the **UML 2.x** model.
4. Name your project, e.g., `OnlineBankingSystem`.

Step 2: Create UML Diagrams

A. Class Diagram

1. Add a **Class Diagram** to the model.
2. Create classes like `User`, `Login`, `Account`.
3. Add attributes and methods:
    - `User`: `username`, `password`

- `Login`: `validateUser()`, `authenticateUser()`
- `Account`: `accountNumber`, `balance`

4. Define relationships:
   - Association between `User` and `Account`.
   - Dependency between `Login` and `User`.

B. Sequence Diagram

1. Add a **Sequence Diagram**.
2. Illustrate the login process:
   - **User → Login → Database → Login → User**
3. Show messages like `enterCredentials()`, `validateUser()`, `displayDashboard()`.

C. Activity Diagram

1. Add an **Activity Diagram** for login flow.
2. Define activities:
   - **Enter username & password → Validate → Authenticate → Success/Error message.**
3. Use **decision nodes** for valid/invalid credentials.

Step 3: Code Generation

1. Select a class (e.g., `User`) → **Right-click → Generate Code → Choose Language (Java, C#, Python, etc.)**
2. StarUML generates skeleton code with class names, attributes, and methods.
3. Developers can then add logic to these generated classes.

Step 4: Documentation and Export

1. Export diagrams:
   - **File → Export → Diagram as Image (PNG, SVG)** for reports.
2. Generate HTML or PDF reports for project documentation.
3. Helps maintain **traceability** between requirements, design, and implementation.

---

3. Benefits of Using StarUML (or any CASE tool)

- Speeds up software design with **drag-and-drop UML modeling**.
- Maintains **consistency** between diagrams and code.
- Supports **team collaboration** by sharing models.
- Useful for **educational, professional, and documentation purposes**.

1. Project Name: `OnlineBankingSystem`

---

2. UML Diagrams to Include

A. Class Diagram

Classes and Attributes/Methods:

Class	Attributes	Methods
User	username, password, email	getUsername(), getPassword()
Login	-	enterCredentials(), validateUser(), authenticateUser()
Account	accountNumber, balance	getBalance(), deposit(), withdraw()

Relationships:

- Association: User → Account
- Dependency: Login → User

B. Sequence Diagram (Login Process)

Participants: User, Login, Database

Flow:

1. User → Login: enterCredentials()
2. Login → User: validateUser()
3. Login → Database: authenticateUser()
4. Database → Login: authenticationResult
5. Login → User: displayDashboard() / displayError()

C. Activity Diagram (Login Flow)

Activities & Decisions:

- **Start → Enter Username & Password → Decision: Are credentials valid?**
  - **Yes → Redirect to Dashboard → End**
  - **No → Display Error → End**

---

3. Code Generation (Optional Demo)

- From the **Class Diagram**, generate skeleton code in **Java/Python/C#**.
- Example: `User.java` with class, attributes, and empty methods.

---

4. Documentation & Export

- Export each diagram as **PNG or SVG** for reports.
- Generate **HTML or PDF report** from StarUML to show project structure.

---

5. Deliverables for Submission

1. StarUML **.mdj project file** (editable in StarUML).
2. PNG/SVG **diagrams** (Class, Sequence, Activity).
3. Optional: **Generated skeleton code**.
4. Optional: **HTML/PDF documentation** exported from StarUML.
