Practical-9

Aim: Develop Activity diagram for following problem:

a) **Banking management system**
b) **College management system**

What is an Activity Diagram?

An Activity Diagram is a type of UML (Unified Modeling Language) diagram that illustrates the dynamic aspects of a system. It shows the flow of control or data from one activity to another and is particularly useful for modeling the workflow of a system or business process.

Purpose of an Activity Diagram:

- To represent sequential and parallel flows in a system.
- To understand the logic of complex operations or business processes.
- To describe the control flow from one activity to another.
- To model use case workflows.
- To aid in system design, process documentation, and project planning.

Components of Activity Diagrams:

Element	Description
Initial Node	Denotes the start of the process.
Activity/Action	Represents a task or function.
Decision Node	A diamond shape that splits flow based on conditions (yes/no, true/false).
Merge Node	Combines multiple flows into one.
Fork Node	Splits into parallel activities.
Join Node	Synchronizes parallel flows.
Final Node	Marks the end of the process.

a) Banking Management System

Objective:

To model the workflow and activities in a Banking Management System to understand user interactions such as login, balance enquiry, money transfer, and cheque-related operations.

Overview:

The diagram contains **three swimlanes** for:

- **Customer**
- **Account & Transaction Details**
- **Cheque Details**

Process Flow:

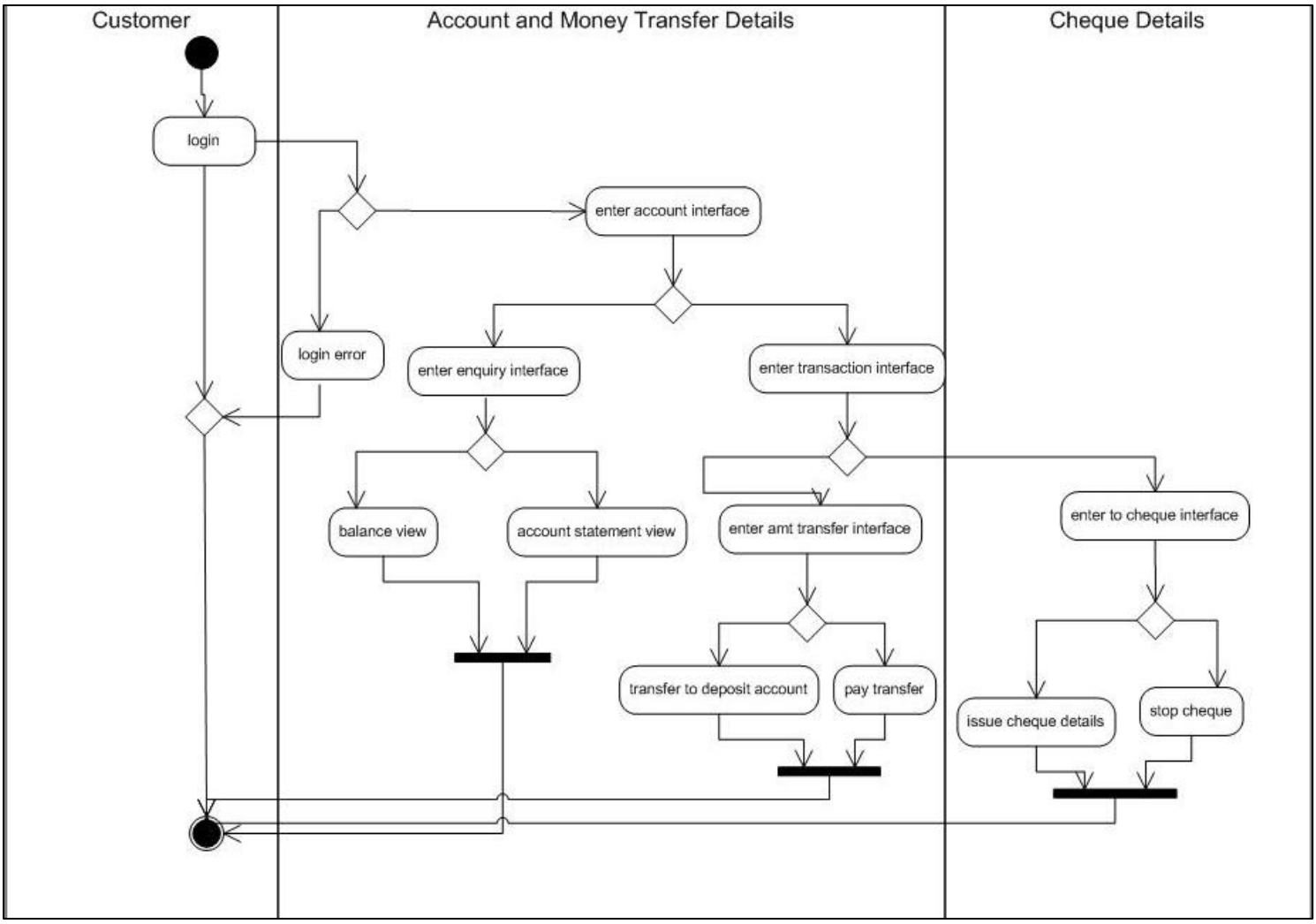
1. **Customer Login:**
    - User inputs credentials.
    - If credentials are invalid → Login Error.
    - If valid → Proceed to main interface.
2. **Enquiry Interface:**
    - View Balance
    - View Account Statement
3. **Transaction Interface:**
    - Initiate Money Transfer
    - Choose between:
        - Transfer to Deposit Account
        - Pay Transfer
4. **Cheque Interface:**
    - Issue Cheque
    - Option to Stop Cheque

Diagram Features:

- Decision nodes check login validity.
- Fork/Join nodes used to process parallel interfaces.
- Swimlanes clearly separate responsibilities.

Conclusion:

The diagram effectively models how a customer interacts with different banking services. It allows developers to plan interfaces and backend workflows according to user interactions and requirements.



b) College Management System

Objective:

To illustrate the major roles (student, teacher, administrator) and their activities in a college environment using an activity diagram.

Overview:

This system includes:

- **Login Validation**

- **Role-based Activity Distribution**

Actors & Roles:

1. **Student Activities:**
    - Access learning materials
    - Participate in activities
    - Track academic progress
    - Give feedback
2. **Teacher Activities:**
    - Create/manage courses
    - Evaluate assignments/quizzes
    - Interact with students
    - Give personalized guidance
3. **Administrator Activities:**
    - Manage user accounts
    - Monitor course content
    - Generate reports
    - Implement feedback and improvements

Flow Highlights:

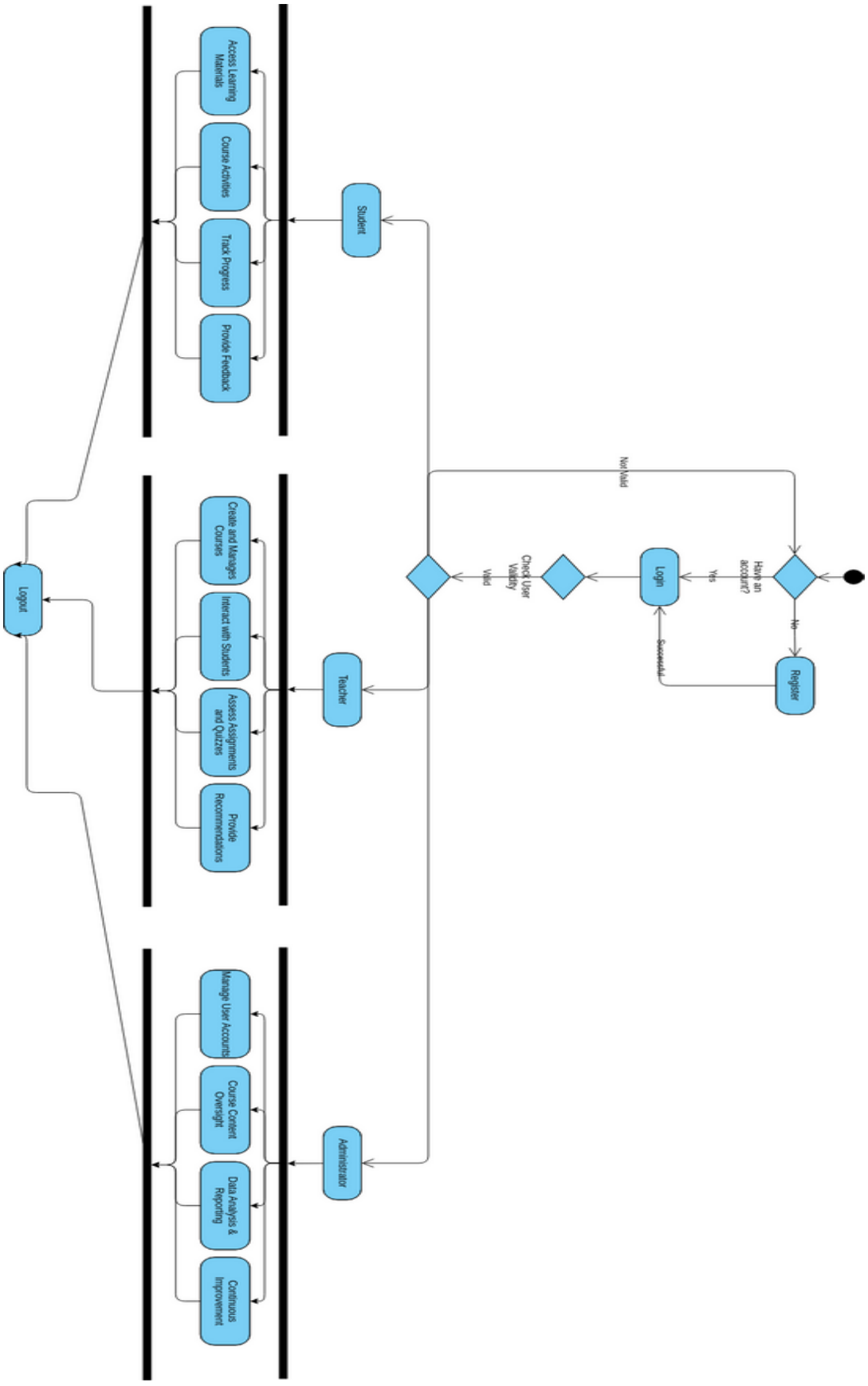
- A single login system branches out into role-specific activities.
- Activity paths are mutually exclusive based on the user type.
- Merge node consolidates all roles before reaching the end.

Conclusion:

The activity diagram models an efficient, organized system where each user type has a well-defined path and role. It's helpful in planning system modules and access control mechanisms.

Final Conclusion: Activity diagrams serve as a visual blueprint for understanding, designing, and improving system workflows. Whether it's a bank or a college system, activity diagrams

break down complex behavior into simple, understandable steps, enhancing communication among stakeholders and aiding in development.



Practical-10

Aim: Develop requirements specification for following problem:

c) **Banking management system**
d) **College management system**

Introduction

In software engineering, requirements engineering is the cornerstone of building a reliable, scalable, and functional software product. It involves identifying, analyzing, documenting, and validating the needs and expectations of stakeholders. A clear understanding of software requirements ensures that the final product meets user expectations and complies with regulatory and operational constraints.

This practical file outlines the key requirements for two commonly developed systems: a Banking Management System and a College Automation System. These systems serve different domains—financial services and educational institutions respectively—but both share the necessity for robust, user-centric design and careful attention to performance, security, and scalability.

Each system is analyzed from three critical perspectives:

1. **Domain Requirements: High-Level Business Needs and System Context**
   This perspective focuses on understanding why the system is being developed in the first place. It identifies the overarching business goals, stakeholder expectations, and the environment in which the system will operate. Domain requirements often include regulations, organizational policies, and broad objectives that the system must support. They help establish the system's role within the larger business or operational context and ensure alignment with strategic priorities.

2. **Functional Requirements: What the System Should Do**
   These are the specific behaviors, tasks, or services the system must perform. Functional requirements detail the interactions between the system and its users, including inputs, outputs, data handling, and user interface behavior. They define the core features of the system—for example, "The system must allow users to register and log in" or "The system should generate monthly performance reports." Capturing these requirements accurately is essential for guiding development and testing.

3. **Non-Functional Requirements: How the System Should Perform**
   Unlike functional requirements, which describe *what* the system does, non-functional requirements describe *how well* it should perform under certain conditions. These include

aspects such as performance, scalability, security, usability, maintainability, and reliability. For instance, a system may be required to handle 10,000 concurrent users with a response time under 2 seconds or comply with data privacy standards such as GDPR. These requirements are crucial for ensuring the system's quality and user satisfaction.

a) Banking Management System (BMS)

1. Domain Requirements

The Banking Management System operates within the financial services domain, serving commercial banks, cooperative banks, and financial institutions. The system should facilitate the seamless functioning of day-to-day banking operations, including customer account management, fund transfers, loan processing, and compliance reporting.

The banking domain is complex, requiring compliance with international financial standards such as KYC, AML, BASEL norms, and data security laws like GDPR or local equivalents. The BMS must integrate with third-party systems including credit rating agencies, national ID verification services, interbank transfer networks (NEFT/RTGS/IMPS), and taxation portals.

Other domain-specific needs include:

1. Supporting multiple branches and currencies.

2. Managing high transaction volumes.

3. Providing a unified customer view across savings, loans, investments, and cards.

4. Adapting to modern banking trends like UPI, digital wallets, chatbot support, and AI-driven analytics.

5. Moreover, the system must be built to support multi-tier architecture with multi-factor authentication and real-time fraud detection.

2. Functional Requirements

Here are the major functionalities the BMS must support:

1. User Authentication & Roles: Role-based access for customers, tellers, branch managers, and administrators with secure login protocols.

2. Account Operations: Open/close/manage accounts, view balances, account statements, and transaction summaries.

3. Transaction Processing: Handle intra-bank and inter-bank fund transfers, mobile payments, standing instructions, and real-time updates.

4. Loan Services: Loan application, approval workflow, EMI calculator, disbursal tracking, and status monitoring.

5. Customer Support: Ticket management, live chat, knowledge base, and escalation tracking.

6. Notification System: Email/SMS alerts for key events like login attempts, large transactions, and suspicious activity.

7. Internal Dashboards: Reporting for account growth, loan defaults, transaction volume, and staff performance.

8. Audit Logging: Maintain logs of all user activities for compliance audits and dispute resolution.

Each function must be transaction-safe and implement ACID properties to ensure data integrity.

3. Non-Functional Requirements

The BMS must meet the following non-functional standards:

1. Security: Encrypted communication (TLS/SSL), secure APIs, firewall integration, intrusion detection, and role-based access control.

2. Availability: Must ensure 24/7 uptime with automatic failover systems and backup power sources. Aim for 99.99% uptime.

3. Scalability: Should support a growing user base without performance loss. Implement load balancing and sharding for databases.

4. Performance: Transactions should complete in under 2 seconds, with batch jobs optimized to run off-peak hours.

5. Usability: Interfaces should be responsive, multilingual, and intuitive with minimal user training.

6. Maintainability: Modular codebase with documentation for easy updates. Includes APIs for third-party integration.

7. Data Retention & Recovery: Regular database backups, secure cloud storage, and disaster recovery mechanisms must be in place.

8. Compliance & Auditing: Should generate reports for regulators and maintain logs as per legal mandates.

b) College Automation System (CAS)

1. Domain Requirements

The College Automation System is designed to digitize the academic and administrative operations of educational institutions. It serves multiple stakeholders—students, faculty, admin staff, and management—by integrating processes such as enrollment, attendance, grading, library usage, and communication.

CAS must align with regulatory frameworks like NAAC, UGC, AICTE, and the emerging NEP 2020 standards. It should support multi-campus institutions and handle diverse programs—undergraduate, postgraduate, vocational, etc.

Other domain-specific expectations:

1. Integration with external systems like Digilocker, AISHE, and e-learning platforms (Moodle, Google Classroom).

2. Digitalization of academic records, transcripts, and mark sheets.

3. Handling semester-based or credit-based course structures.

4. Hostel and transport management.

5. Examination invigilation and secure result publishing.

6. A cloud-based architecture is preferred to allow remote learning and administrative access across multiple branches.

2. Functional Requirements

Core functionalities of CAS include:

1. User Onboarding & Roles: Admins, faculty, students, parents, and staff should access tailored dashboards.

2. Student Lifecycle Management: Admission, course registration, fee payment, graduation records.

3. Class Scheduling: Generate conflict-free timetables, faculty assignments, and room management.

4. Attendance System: Biometric or RFID-based attendance tracking with alerts for defaulters.

5. Grading System: Grade entry by faculty, GPA calculations, result publishing, and transcript generation.

6. Fee & Finance: Manage fee categories, concessions, online payments, receipts, and overdue tracking.

7. Library Management: Book inventory, lending logs, overdue alerts, and digital resources access.

8. Notices & Communication: Messaging portal for circulars, alerts, and feedback between stakeholders.

9. Hostel & Transport Modules: Student room allocation, bus route planning, and fee tracking.

10. The system should also provide data analytics for performance monitoring, placement stats, and academic progress.

3. Non-Functional Requirements

The CAS must adhere to these performance and quality attributes:

1. Security: Role-based permissions, encrypted data storage, and secure user authentication (OTP/email verification).

2. Scalability: Must handle peak traffic during admissions, results, and examinations without downtime.

3. Reliability: Backup mechanisms and uptime monitoring should be in place. Target is 99.9% availability.

4. User Experience: Interfaces must be intuitive, clean, and mobile-friendly. Support for local languages is desirable.

5. Integration Ready: Should provide REST APIs or XML integration with government databases, banks, and ERP systems.

6. Maintainability: Regular system updates, patch management, and documentation for easy maintenance.

7. Data Protection: Compliant with FERPA-like laws; student data should be stored securely with access logging.

8. Accessibility: Compliance with accessibility standards (like WCAG) for visually impaired users.