# Software Engineering

(CSSE-3113)

## Current Banking Account System

**Rana Muhammad Daniyal(G1F23UBSCS051)**
**Ali Akbar Gondal(G1F23UBSCS061)**
**Abdullah Mazhar(G1F23UBSCS066)**
**Muhammad Usman Khalid Mughal(G1F23UBSCS097)**
**Abu Umair(G1F23UBSCS306)**


**SESSION 2023-2027**

**Submitted to**

**Prof. Obaid Ullah Obaid**

**DEPARTMENT OF COMPUTER SCIENCE**

**UNIVERSITY OF CENTRAL PUNJAB,**

**GUJRANWALA.**

# Table Of Contents

# 1. Introduction

## 1.1 Purpose

The purpose of this document is to define the software requirements for the **Current Account Banking System**. This system is being developed as part of the **Software Engineering (CSSE3113)** course. This document covers the functional and non-functional requirements required to build a secure, web-based banking platform that manages current accounts, ensuring zero-interest handling, overdraft facilities, and real-time transaction processing.

## 1.2 Document Conventions

- **Bold text** is used to highlight specific roles (e.g., **Admin**, **Teller**) or critical system states.
- **FR** denotes Functional Requirements (e.g., FR1.1).
- **NFR** denotes Non-Functional Requirements (e.g., NFR1.1).
- Priorities are assumed to be **High** for all core banking transactions (Deposits, Transfers) unless otherwise stated.

## 1.3 Intended Audience

- **Project Supervisor (Prof. Obaid Ullah Obaid):** To evaluate the scope and quality of the requirements.

- **Development Team:** To understand the specific logic for modules such as the "Overdraft Limit" and "0% Interest" logic.

- **Quality Assurance (QA):** To design test cases based on the inputs and outputs defined in Section 4.

## 1.4 Product Scope

The **Current Account Banking System** is a web-based software solution designed to replace manual ledgers with a real-time digital balance database. The system aims to:

Provide role-based access for Admins, Tellers, and Customers.

Manage "Current Accounts" specifically, ensuring **0% interest** is applied.

Facilitate fund transfers, bill payments, and beneficiary managements

Provide unique features like **Chequebook Management** and **Stop Payment** services.

# 2. Overall Description

## 2.1 Product Perspective

This is a new, self-contained product. It is a web-based application designed to run on modern web browsers (Chrome, Edge, Safari). It connects to a centralized cloud database to ensure data consistency and allows for 24/7 access for customers.

## 2.2 Product Functions

The system supports the following major functions:

- **User Authentication:** Secure login with Role-Based Access Control (RBAC).
- **Account Management:** Creating accounts and updating profiles.
- **Transaction Processing:** Cash deposits, fund transfers, and transaction reversals.
- **Cheque Operations:** Requesting chequebooks and stopping payments on specific cheque numbers.
- **Reporting:** Generating mini-statements and monthly PDF statements.
- **Bill Payments:** Utility bill payments and mobile top-ups.

## 2.3 User Classes and Characteristics

| User Class | Description | Characteristics |
|---|---|---|
| **Admin** | Bank managers or IT staff. | Technically competent. Has full access to account creation, limit setting, and user management. |

| | | |
|---|---|---|
| **Teller (Cashier)** | Bank employees at the counter. | Focused on speed and accuracy. Has access to deposit, withdrawal, and cheque clearance modules. |
| **Customer** | The account holder. | Non-technical. Accesses the system via personal dashboard to view balances, transfer funds, and pay bills. |

### 2.4 Operating Environment

- **Client Side:** Any device with a web browser (Laptop, Tablet, Mobile).
- **Server Side:** Cloud-based server supporting Python/Node.js (or selected backend).
- **Database:** Relational Database (SQL) to ensure ACID compliance.
- **Network:** Requires active Internet connection (HTTPS).

### 2.5 Design and Implementation Constraints

- **Zero Interest Rule:** The system must strictly enforce a 0% interest rate on all Current Accounts.
- **Encryption:** Sensitive data (CNIC, Address) must be encrypted using **AES-256** standards.
- **Password Policy:** Passwords must be at least 8 characters with mixed case, numbers, and special characters.
- **Session Timeout:** Users must be logged out after 5 minutes of inactivity.

---

## 3. External Interface Requirements

### 3.1 User Interfaces

The system will feature a responsive web interface (HTML/CSS/React/Flutter Web) that adjusts to screen size.

- **Dashboard:** Displays Account Balance, Mini-Statement (Last 10 transactions), and Quick Actions.
- **Feedback Mechanism:** Error messages will be financial and clear (e.g., "Insufficient Funds" rather than "Error 404").

### 3.2 Hardware Interfaces

- **Printers:** The system shall interface with standard office printers to generate transaction slips and PDF statements.
- **Server:** The system requires a backend server capable of handling 100 transactions per second.

### 3.3 Software Interfaces

- **Database:** The system will interface with a SQL database to store financial ledgers and perform daily backups.
- **Export Tools:** The system will interface with libraries to export data to **Excel (.xlsx)** and **PDF** formats.

### 3.4 Communications Interfaces

- **Protocol:** HTTPS for all client-server communication.
- **Notification Service:** The system will interface with an SMS/Email gateway to send Low Balance Alerts.

## 4. Functional Requirements

**FR1: User Authentication & Access Control**
The system must ensure secure access and differentiate between roles (e.g., Admin,

Teller/Cashier, Customer).

- **FR1.1: User Login:**
  The system shall allow users to log in using a unique Username/Account Number and Password.
- **FR1.2: Role-Based Access Control (RBAC):**
  - **FR1.2.1:** Admins shall have full access to account creation, limit setting, and user management.
  - **FR1.2.2:** Tellers shall have access to deposit, withdrawal, and cheque clearance modules.
  - **FR1.2.3:** Customers shall only have access to their personal dashboard, transfer interfaces, and statements.
- **FR1.3: Profile Management:**
  Users shall be able to update their passwords and personal contact information (email/phone) upon verification.

**FR2: Account Fund Management**

This module replaces manual ledgers with a real-time digital balance database.

- **FR2.1: Deposit Entry:**
  The system shall allow Tellers to process cash deposits by entering: Account Number, Amount, Depositor Name, and Source of Funds.
- **FR2.2: Real-time Updates:**
  The system shall automatically increment or decrement the account balance immediately upon a completed transaction.
- **FR2.3: Low Balance Alerts:**
  The system shall trigger a dashboard notification or SMS when the account balance falls below the minimum requirement.
- **FR2.4: Balance Adjustment:**
  The system shall allow authorized Admins to manually reverse a transaction (e.g., for error correction) with a mandatory reason entry.

**FR3: Transaction Processing & Transfers**

This replaces manual cheque clearing and physical cash handling for transfers.

**FR3.1: Account Validation:**
The system shall validate the beneficiary account number before allowing a transfer to proceed.

**FR3.2: Automated Calculation:**

- **FR3.2.1:** The system shall calculate the transfer amount and deduct any applicable service charges/tax automatically.
- **FR3.2.2:** The system shall ensure the "Current Account" does not apply any interest (0% interest rate) on the balance.

**FR3.3: Receipt Generation:**
The system shall generate and print a professional digital receipt or PDF transaction slip for every successful transfer.

**FR3.4: Transaction Reversal:**

The system shall handle failed transactions by automatically rolling back funds to the sender's account to prevent data inconsistency.

**FR4: Chequebook & Overdraft Management**

This addresses the critical features specific to Current Accounts.

- **FR4.1: Chequebook Request:**
  The system shall allow customers to request a new chequebook (10, 25, or 50 leaves) through their portal.
- **FR4.2: Overdraft Usage:**
  The system shall allow the account balance to go negative up to a specific "Overdraft Limit" assigned by the bank admin.
- **FR4.3: Stop Payment:**
  The system shall allow a customer to instantly block a specific cheque number to prevent it from being cashed.

**FR5: Beneficiary & Payee Management**

This module tracks the people the customer frequently sends money to.

- **FR5.1: Beneficiary Database:**
  The system shall maintain records of saved beneficiaries (Name, Account Number, Bank Name) for quick transfers.
- **FR5.2: Limit Management:**
  The system shall enforce daily transfer limits per beneficiary as defined by bank policy.

**FR6: Reporting & Statements**

This provides the data for customer tracking and bank auditing.

- **FR6.1: Mini-Statement:**
  The system shall generate a quick view of the last 10 transactions on the dashboard.
- **FR6.2: Monthly Statement:**
  The system shall generate a detailed PDF statement showing opening balance, all debits/credits, and closing balance.
- **FR6.3: Export Functionality:**
  The system shall allow users to export their transaction history in Excel or PDF formats.

**FR7: Search & Filter Capabilities**

To replace the slow process of finding old records.

- **FR7.1: Global Search:**
  The system shall provide a search bar to find transactions by Transaction ID, Cheque Number, or Beneficiary Name.
- **FR7.2: Advanced Filtering:**
  Users shall be able to filter transactions by:
  - Date Range (Start Date - End Date)
  - Transaction Type (Debit/Credit)
  - Amount Range

**FR8: System Administration & Maintenance**

These requirements ensure the software remains healthy and secure.

- **FR8.1: Database Backup:**
  - **FR8.1.1:** The system shall perform an automatic daily backup of all financial ledgers.

    - **FR8.1.2:** The system shall encrypt these backups before storing them on the cloud.
- **FR8.2: Audit Logs:**
  The system shall record every action taken (who viewed an account, who authorized a large transfer) to prevent internal fraud.

**FR9: Debit Card Management** This module allows customers to manage the physical card linked to their current account.

- **FR9.1: Card Request:** The system shall allow customers to apply for a new Debit Card (Classic, Gold, or Platinum) via the portal.
- **FR9.2: PIN Management:**
    - **FR9.2.1:** The system shall allow users to set or change their 4-digit ATM PIN securely.
    - **FR9.2.2:** The system shall lock the card temporarily after 3 incorrect PIN attempts.
- **FR9.3: Card Blocking:** The system shall provide a "Panic Button" to instantly freeze the debit card in case of theft or loss.
- **FR9.4: Usage Limits:** The system shall allow users to customize their daily spending limits for Online transactions versus ATM withdrawals.

**FR10: Bill Payments & Utilities** This replaces the need for customers to visit physical utility offices.

- **FR10.1: Biller Management:** The system shall allow users to add utility consumer numbers (Electricity, Gas, Internet, Mobile Postpaid) to their profile.
- **FR10.2: Payment Processing:** The system shall fetch the latest bill amount due and allow the user to pay it instantly from their account balance.
- **FR10.3: Mobile Top-ups:** The system shall support prepaid mobile recharges for all major network operators.

**FR11: Standing Instructions (Scheduled Payments)** This is essential for Current Accounts used for business or rent payments.

- **FR11.1: Schedule Creation:** The system shall allow users to set up recurring transfers (e.g., "Pay 50,000 to Landlord on the 5th of every month").
- **FR11.2: Frequency Options:** The system shall support various frequencies: Weekly, Monthly, Quarterly, or Annually.
- **FR11.3: Auto-Execution:** The system shall automatically attempt to execute these transfers on the specified date without manual user intervention.

**FR12: Customer Support & Ticketing**

- **FR12.1: Complaint Logging:** The system shall allow users to submit complaints regarding failed transactions or service issues.
- **FR12.2: Status Tracking:** The system shall assign a unique Ticket ID to every complaint and allow the user to track its status (Open/In-Progress/Resolved).

# 5. Non-Functional Requirements

**NFR1: Security & Compliance**

- NFR1.1 Data Privacy:
  The system must encrypt sensitive customer data (CNIC, Address) using AES-256 encryption standards.
- NFR1.2 Password Complexity:
  The system shall enforce a strong password policy (minimum 8 characters, including upper case, lower case, number, and special character).
- NFR1.3 Session Security:
  The system shall automatically log out the user after 5 minutes of inactivity to prevent unauthorized access.

**NFR2: Performance & Efficiency**

- NFR2.1 Throughput:
  The system must be capable of processing at least 100 transactions per second during peak banking hours.
- NFR2.2 Response Time:
  The balance update must reflect on the user dashboard within 2 seconds of a successful transaction.

**NFR3: Reliability & Fault Tolerance**

- NFR3.1 ACID Compliance:
  The system must use "Atomic Transactions." If a network failure occurs during a money transfer, the money must not be deducted from the sender if it hasn't reached the receiver.
- NFR3.2 Availability:
  The system must ensure 99.9% uptime, allowing customers to access their funds 24/7.

**NFR4: Usability & Human Factors**

- NFR4.1 Error Messaging:
  The system must provide clear, financial error messages (e.g., "Insufficient Funds for this transaction" instead of "Error Code 404").
- NFR4.2 Input Validation:
  The system must prevent users from entering negative numbers in the "Amount" field.

**NFR5: Maintainability & Scalability**

- NFR5.1 Data Growth:
  The system should be able to store 10 years of transaction history without performance degradation.
- NFR5.2 Scalability:
  The backend must support adding new servers to handle increased user load without rewriting the code.

**NFR6: Portability**

- NFR6.1 Browser Compatibility:

  The system must function identically across all major browsers (Chrome, Firefox, Safari, Edge) without requiring specific plugins.

- NFR6.2 Device Independence:

  The layout must automatically adjust (responsive design) to fit screens of laptops, tablets, and mobile phones.

# 6. System Architecture & Other Requirements

## 6.1 Database Requirements

- The system shall use an easily scalable, ACID-compliant relational database management system (e.g., PostgreSQL) capable of handling high-frequency financial ledgers and up to 100 concurrent transactions per second during peak banking hours.
- The database must support advanced encryption extensions to securely store and query AES-256 encrypted columns for sensitive customer data (CNIC, PIN hashes).

## 6.2 Legal and Regulatory Requirements

- **KYC & AML:** The system shall comply with national central bank regulations regarding "Know Your Customer" (KYC) and Anti-Money Laundering (AML) policies, mandating the collection and verification of identity documents before account creation.
- **Islamic/Current Account Compliance:** The system must strictly enforce legal banking definitions of a Current Account, ensuring zero percent (0%) interest is applied to deposited funds at all times.
- **PCI-DSS:** The system shall not store complete debit card numbers or unhashed PINs locally; all sensitive card management data must be handled in compliance with PCI-DSS (Payment Card Industry Data Security Standard).

## 6.3 Technology Stack Constraints

- **Mobile Application (Customer):** The mobile frontend for customers shall be developed using a cross-platform framework like Flutter (Dart) to ensure concurrent, secure deployment on iOS and Android from a single codebase.

- **Web Dashboard (Admin/Teller):** The internal bank portal for Admins and Tellers shall be built using a modern, strictly typed web framework like React to handle complex data tables and transaction interfaces efficiently.
- **Backend API:** The core banking server logic shall be implemented using a high-performance, secure Python framework (such as FastAPI) to ensure rapid response times, secure token-based authentication, and efficient handling of simultaneous ledger updates.

# 7. Software Testing Strategy

To ensure the reliability, security, and performance of the Current Account Banking System, a comprehensive testing strategy will be implemented throughout the development lifecycle.

## 7.1 Testing Techniques (The "Box" Approaches)

We will utilize three main perspectives to test the system:

- **Black Box Testing:** This focuses entirely on inputs and outputs without looking at the internal code. For example, QA testers will input a "Deposit Amount" into the UI and verify if the "Success Receipt" is generated, validating **FR2.1**.
- **White Box Testing:** This involves testing the internal logic, structure, and code pathways. Developers will examine the backend code to ensure the if/else statements for the **0% Interest Rule** and the **ACID Compliance (NFR3.1)** transaction rollbacks execute properly at the code level.
- **Grey Box Testing:** A hybrid approach where testers have partial knowledge of the internal structure. For instance, a tester will initiate a "Fund Transfer" on the front-end interface, and then manually query the SQL Database to ensure the exact balance was properly deducted and credited.

## 7.2 Levels of Testing

The system will be tested in progressive stages:

- **Unit Testing:** The smallest pieces of code (individual functions) will be tested in isolation. For example, testing the specific function that validates if an inputted transfer amount is a negative number (**NFR4.2**).
- **Integration Testing:** Testing how different modules communicate with each other (e.g., verifying that the Login Module correctly passes the User Role to the

Dashboard Module).

- **System Testing:** Testing the fully assembled software as a complete whole to ensure it meets all the requirements defined in Section 4.
- **Acceptance Testing:** The final phase where the end-user (or simulated Bank Admin/Customer) tests the system to ensure it is ready for real-world deployment.

## 7.3 Integration Testing Strategies

To assemble and test the modules of the banking system, we will evaluate the following integration approaches:

- **Top-Down Integration:** Testing starts from the main UI (Dashboard) and moves down to the backend database. We will use dummy code called "Stubs" to simulate the database before the backend is fully built.
- **Bottom-Up Integration:** Testing starts at the lowest level (SQL Database and core transaction logic) and moves up to the UI. We will use "Drivers" to simulate user inputs before the front-end is complete.
- **Sandwich (Hybrid) Integration:** Combining both Top-Down and Bottom-Up simultaneously. The UI and the Database are tested at the same time and meet in the middle (the API/Logic layer). This is the optimal strategy for our team to work in parallel.
- **Big Bang Integration:** Waiting until every single module is 100% coded and throwing them all together at once to test. (Note: This approach will be avoided as it makes finding the source of transaction errors very difficult).

## 7.4 Static vs. Dynamic Testing

To ensure quality throughout the entire lifecycle, we will use both approaches:

- **Static Testing (Verification):** This involves reviewing documents, code walkthroughs, and inspections *before* the code is actually executed. For example, reviewing this SRS document to ensure no requirements contradict the "0% Interest Rule."
- **Dynamic Testing (Validation):** This is the actual execution of the programmed code to validate that the Current Account system behaves as expected under various conditions.

## 7.5 Functional vs. Non-Functional Testing

Our testing will not just cover what the system does, but *how well* it does it.

- **Functional Testing:** Ensuring features like "Deposit Cash" and "Block Card" work according to the FRs.
- **Non-Functional Testing:** Testing the system's operational capabilities (NFRs), specifically:
    - **Performance/Load Testing:** Simulating high traffic to ensure the system can handle the required **100 transactions per second (NFR2.1)** without crashing.
    - **Security Testing:** Attempting unauthorized access and verifying that passwords meet complexity rules and data is encrypted using **AES-256 (NFR1.1)**.
    - **Usability Testing:** Ensuring the Customer Dashboard is intuitive and responsive across mobile and desktop devices **(NFR6.2)**.

## 7.6 Regression Testing

Whenever a new feature is added (e.g., adding "Bill Payments" after the "Transfer" module is finished) or a bug is fixed, **Regression Testing** will be performed. This ensures that the new changes did not accidentally break any previously working parts of the banking system.

## 7.7 Sample Test Cases

To demonstrate our testing approach, below is a sample test case for the Fund Transfer module:

| Test Case ID | Feature Tested | Pre-Condition | Steps to Execute | Expected Result | Pass/Fail |
|---|---|---|---|---|---|
| **TC-01** | FR3.1: Transfer | User is logged in. | 1. Enter Beneficiary Acc No.<br>2. Enter Amount | System validates account, deducts $500, shows | Pending |

| | | | | | |
|---|---|---|---|---|---|
| | | | ($500).<br><br>3. Click Transfer. | Success Receipt. | |
| **TC-02** | NFR4.2: Validation | User is on Transfer Page. | 1. Enter Amount: "-100"<br><br>2. Click Transfer. | System blocks transfer, shows error: "Invalid negative amount." | Pending |
| **TC-03** | FR4.2: Overdraft | Balance is $0. Limit is $1000. | 1. Transfer $500.<br><br>2. Check Balance. | Transfer succeeds. Balance shows -$500. | Pending |

# Appendix A: Glossary

- **ACID (Atomicity, Consistency, Isolation, Durability):** A set of database properties that guarantee reliable processing of financial transactions, ensuring that no partial fund transfers occur during network failures.
- **AES-256:** Advanced Encryption Standard with a 256-bit key size. A top-tier cryptographic algorithm used to secure sensitive customer data and passwords.
- **AML (Anti-Money Laundering):** International and national laws and regulations designed to stop the practice of generating income through illegal actions.
- **CNIC:** Computerized National Identity Card. The primary identity document used for customer verification in the system.
- **COCOMO (Constructive Cost Model):** An algorithmic software cost estimation model used to predict project effort, schedule, and team size.
- **Current Account:** A type of bank account designed for frequent, daily

transactions, explicitly operating with a 0% interest rate to comply with specific banking principles.
- **ERD (Entity-Relationship Diagram):** A structural blueprint of the system's database showing tables, columns, and relationships.
- **KLOC:** Kilo Lines of Code (Thousands of lines of code). A metric used to estimate the size of a software project.

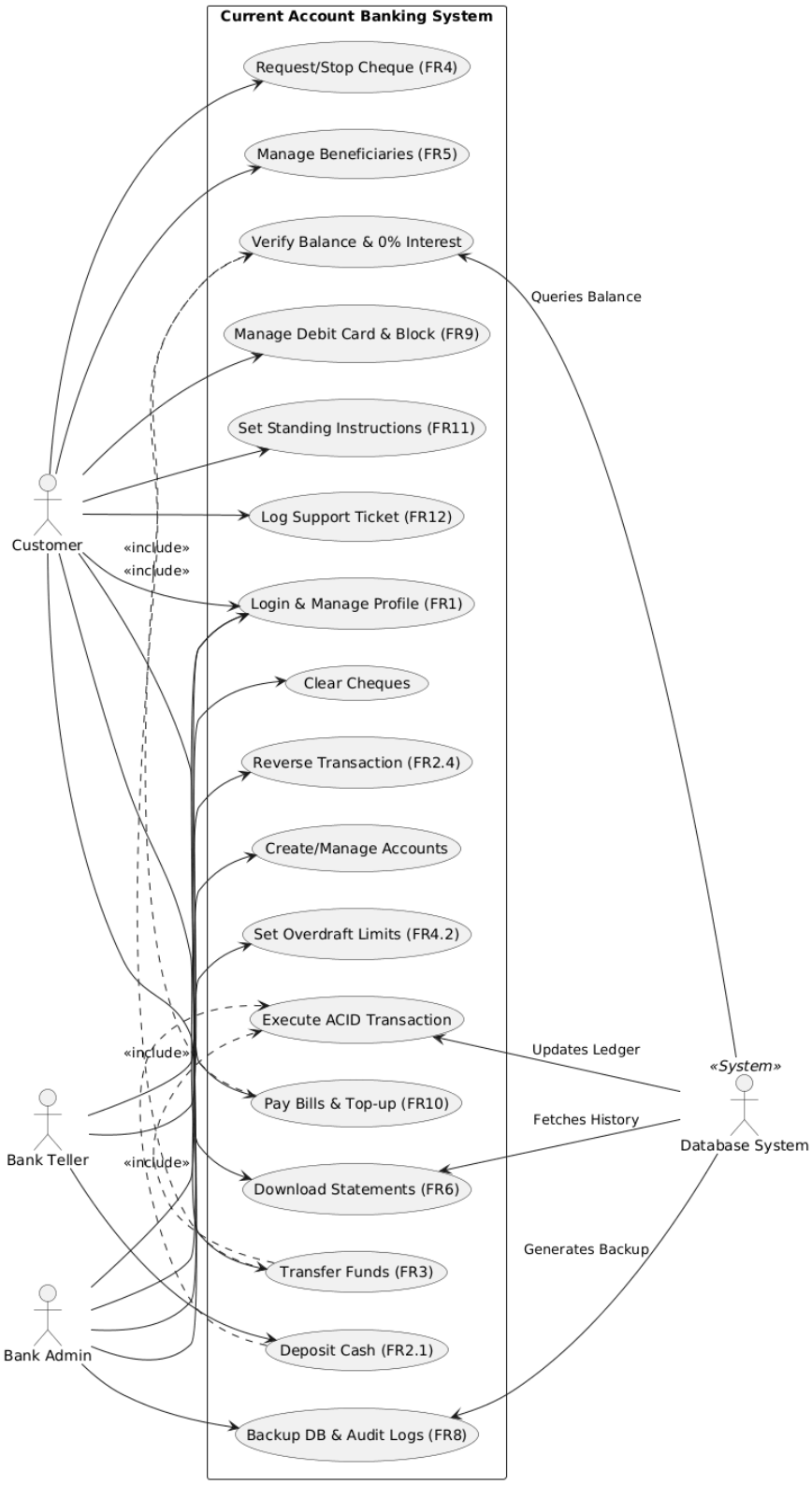
- **KYC (Know Your Customer):** The mandatory regulatory process of identifying and verifying the client's identity when opening a bank account.
- **Overdraft:** A credit facility that allows an account holder to temporarily withdraw more money than their current balance holds, up to a bank-approved limit.
- **PCI-DSS (Payment Card Industry Data Security Standard):** Information security standards ensuring that all systems storing or transmitting debit card information maintain a secure environment.
- **RBAC (Role-Based Access Control):** A security paradigm that restricts system access to authorized users based on their specific roles (e.g., separating Admin privileges from Customer portals).
- **Standing Instruction:** An automated, recurring fund transfer scheduled by the customer to execute on specific dates (e.g., for monthly rent or utility bills).

# Appendix B: Analysis Models (The Diagrams)

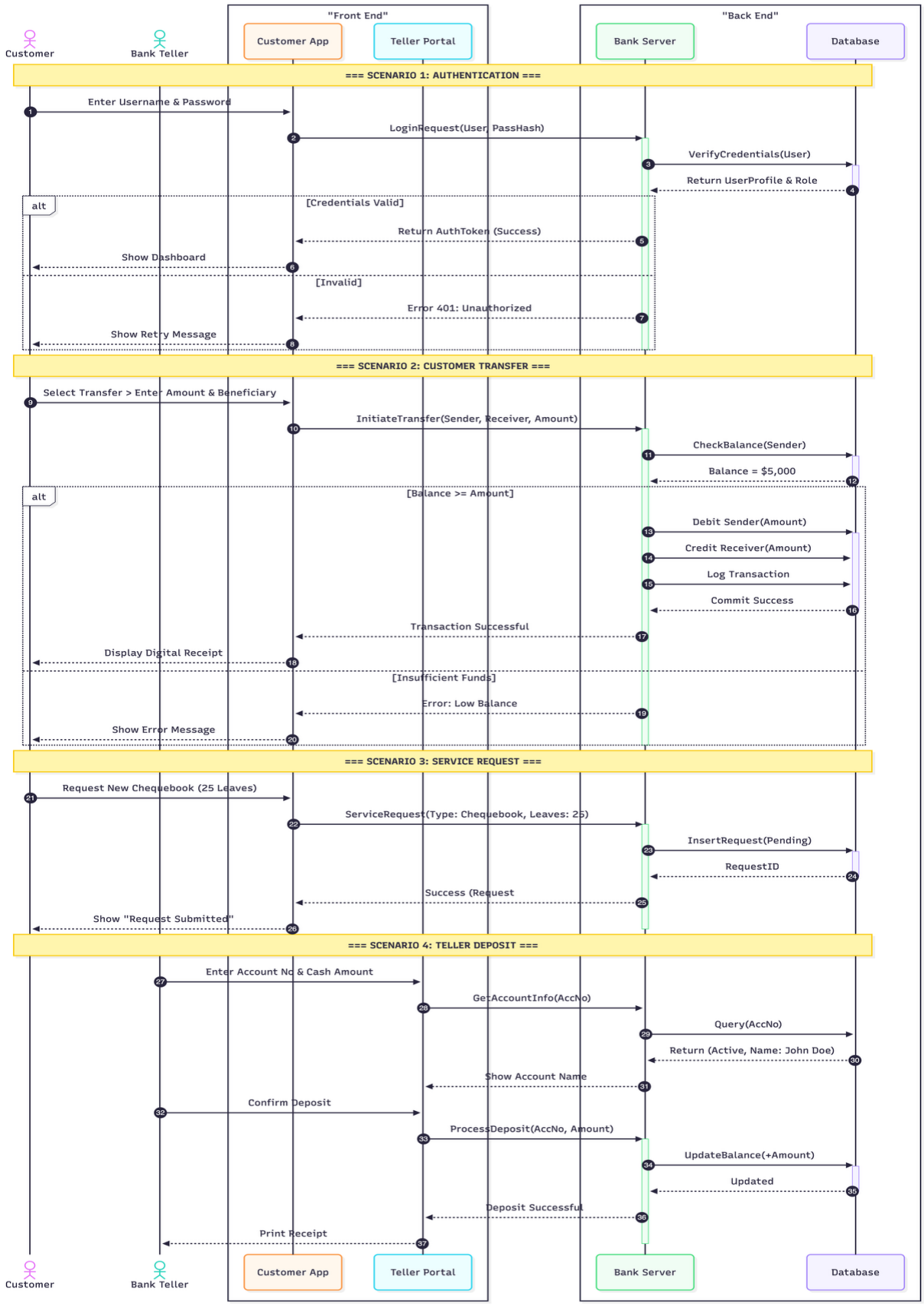
Below are the architectural, flow, and UML diagrams designed for the Current Account Banking System.

- **B.1 Use Case Diagram (Page 16):** Illustrates the interactions between the primary actors (Customer, Bank Teller, Bank Admin), the system boundary, and the secondary external actor (Database System).
- **B.2 System Sequence Diagram (Page 17):** Maps the chronological sequence of messages and data exchanges between the User, the Banking System API, and the Database during core transactions, such as secure fund transfers and balance inquiries.
- **B.3 Activity Diagram (Page 18):** Visualizes the dynamic flow of control and business logic across different architectural swimlanes (Customer Interface, Banking System, Database), tracking the process from authentication to the final digital receipt.
- **B.4 Data Flow Diagrams (DFD):**
    - **Level 0 Context Diagram (Page 19):** Shows the entire Current Account Banking System as a single overarching process and maps its high-level data

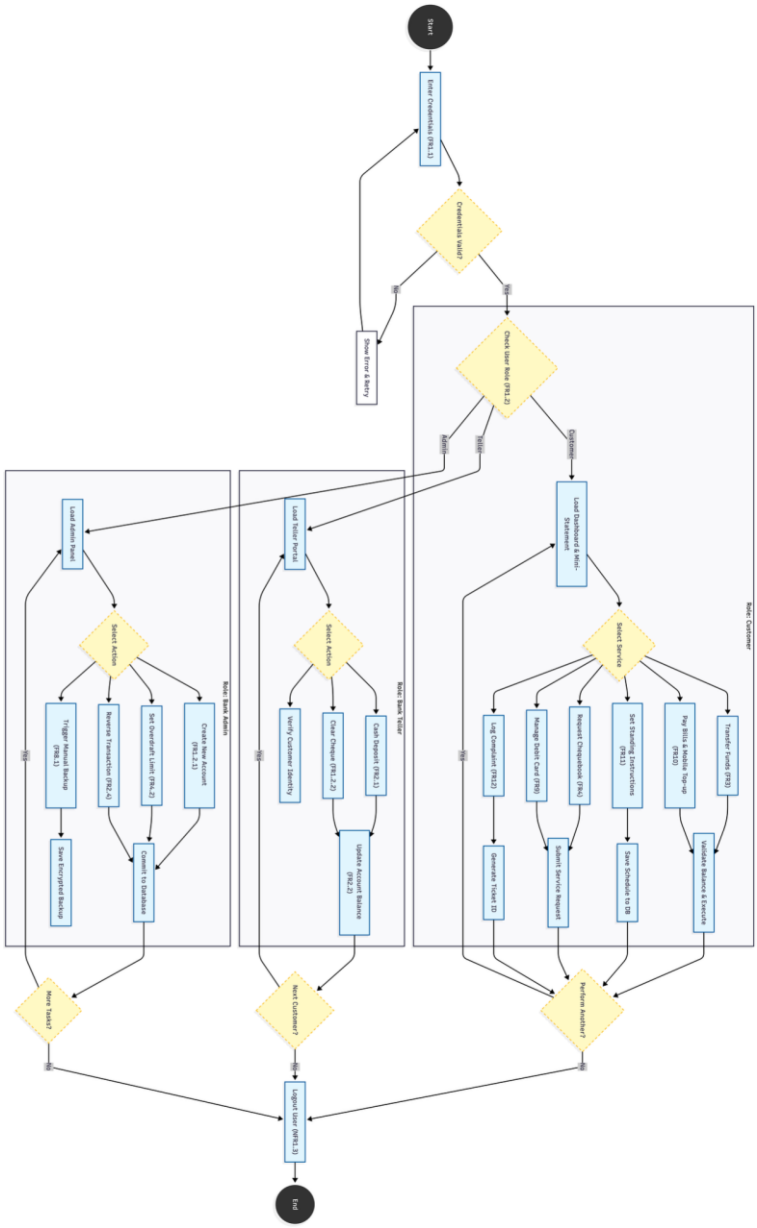
exchanges with all external entities.
    - **Level 1 Detailed DFD (Page 20):** Breaks down the main system into detailed core processes (e.g., Authentication, Transfers, Cheque Management, Bill Payments) and maps their read/write interactions with the data stores.
    - **Level 2 Transfers DFD (Page 21):** Zooms into the specific logic of the fund transfer process, detailing the step-by-step data flow of balance validation, 0% interest enforcement, and ACID compliance rollbacks.
- **B.5 Entity-Relationship Diagram (Page 22):** Defines the strict relational database architecture, mapping out the primary keys, foreign keys, and relationships across tables including Users, Accounts, Transactions, Beneficiaries, Chequebooks, Debit Cards, and Audit Logs.




# Use Case Diagram

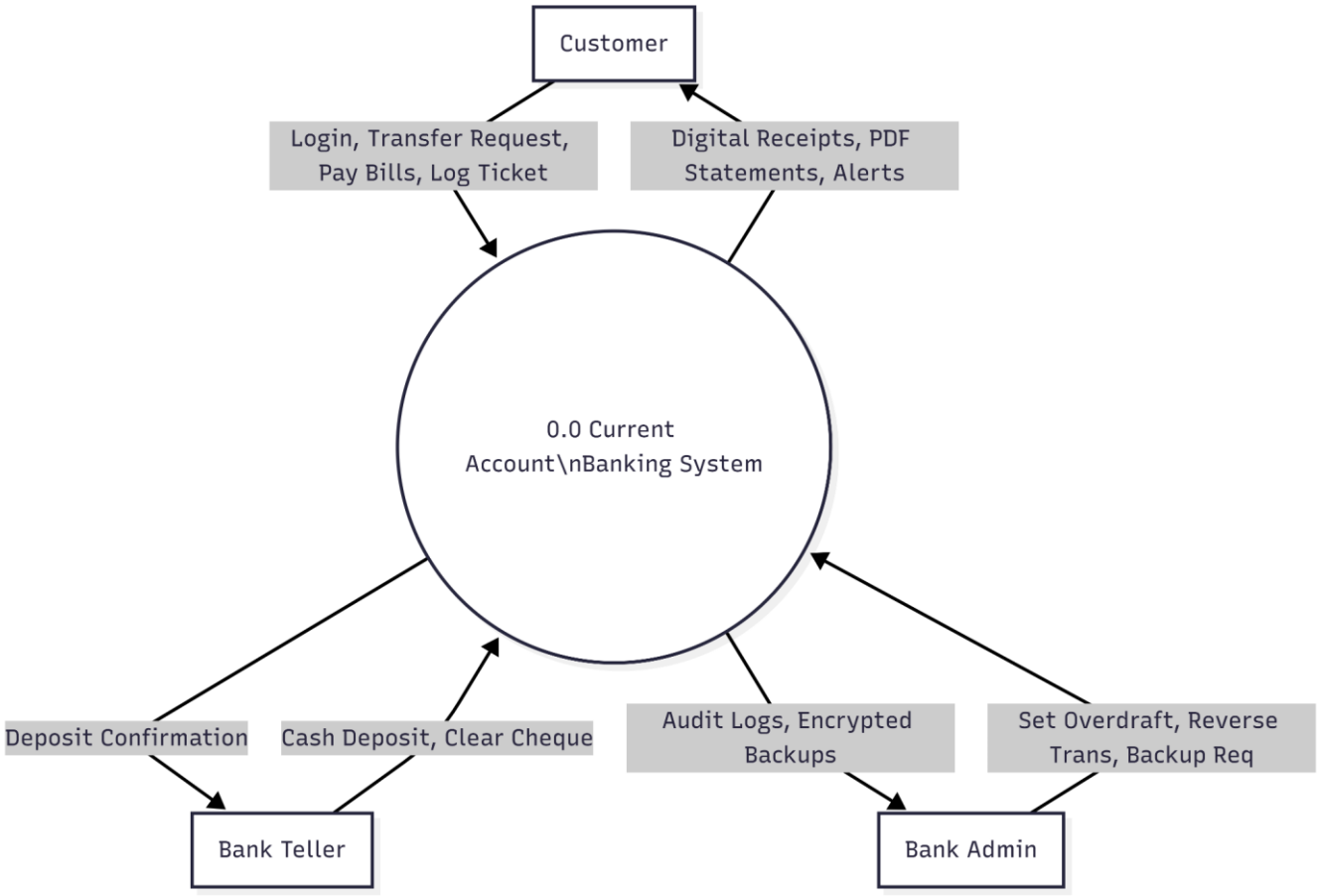



# Sequence Diagram

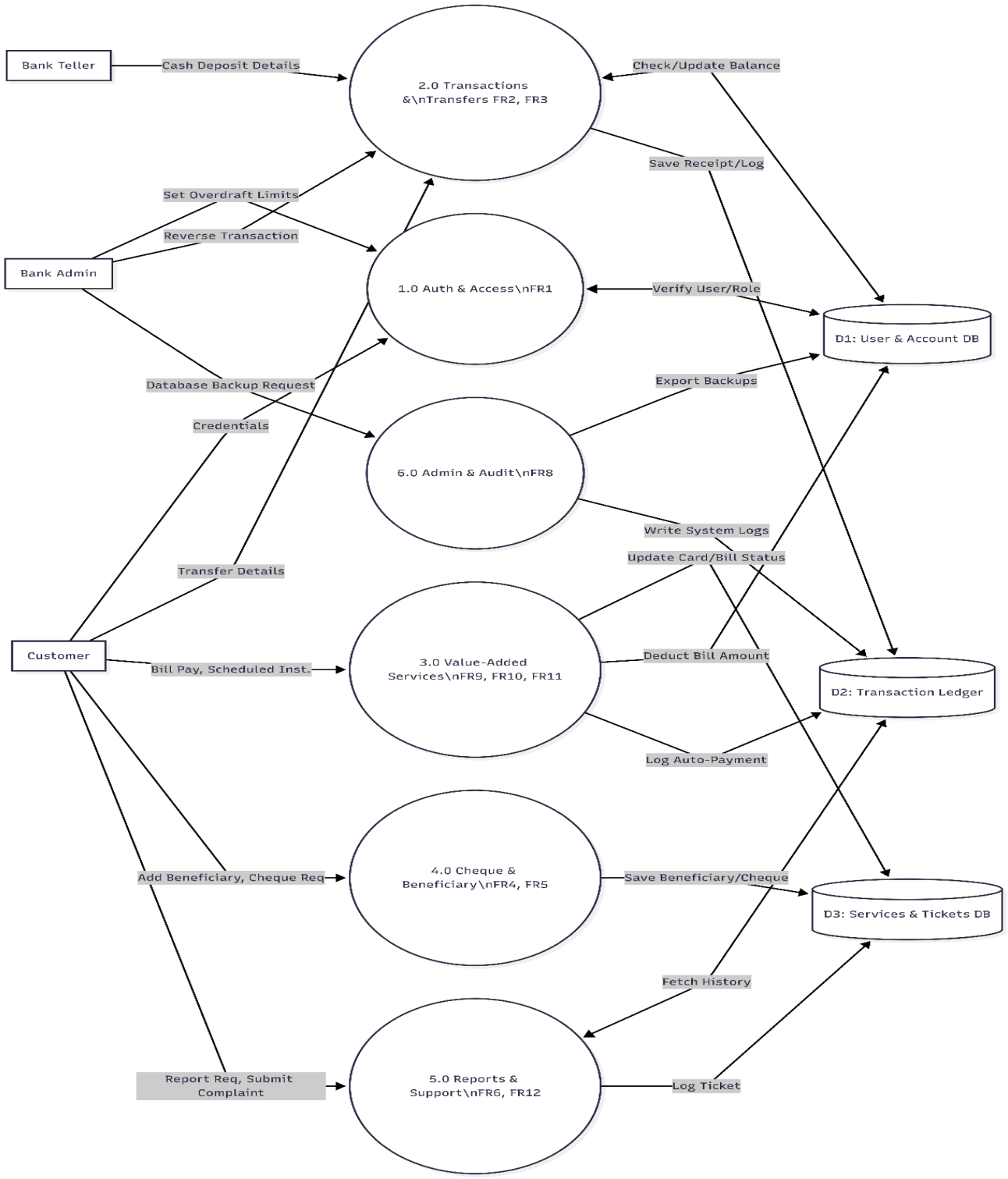



# Activity Diagram

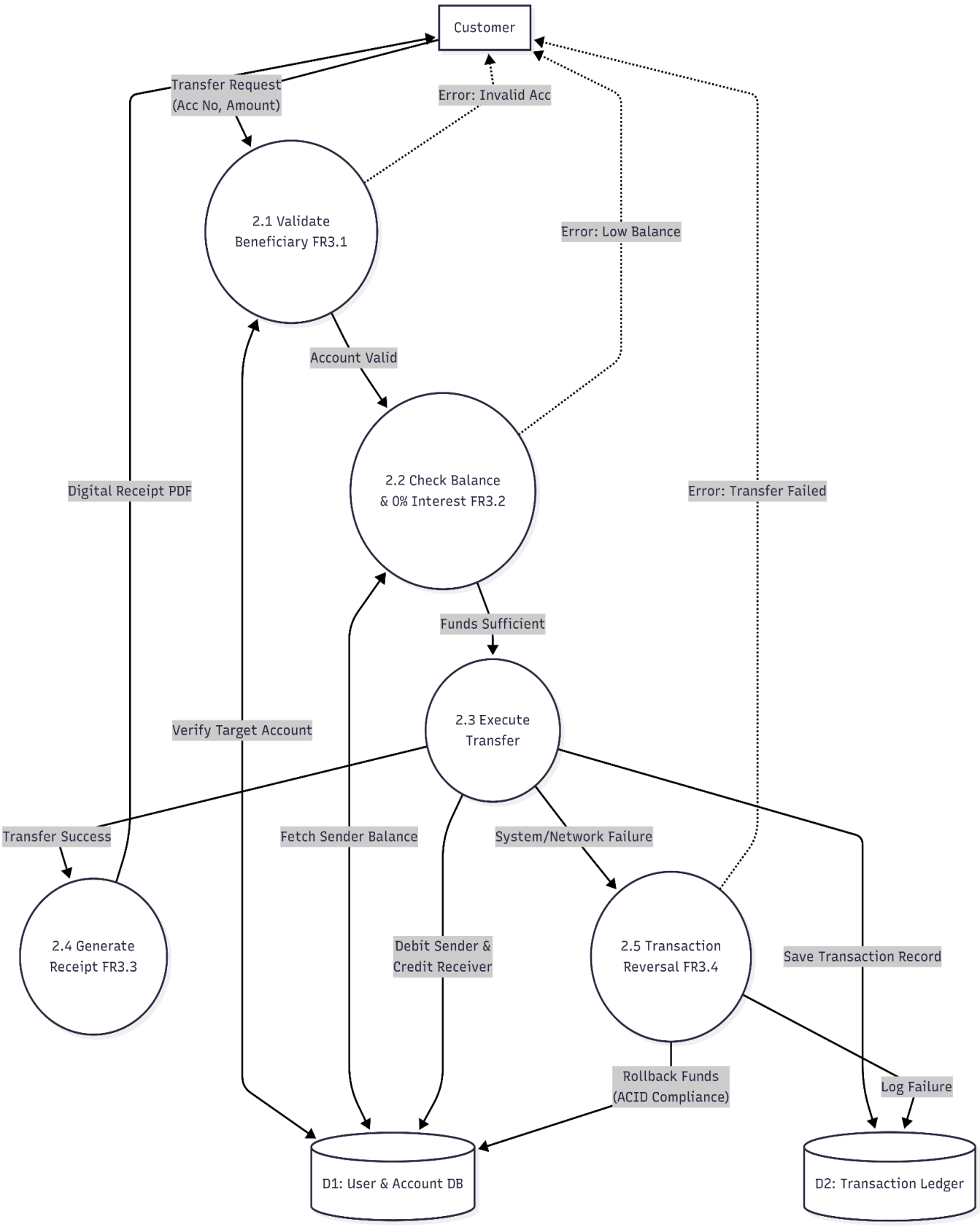
# Data Flow Diagrams



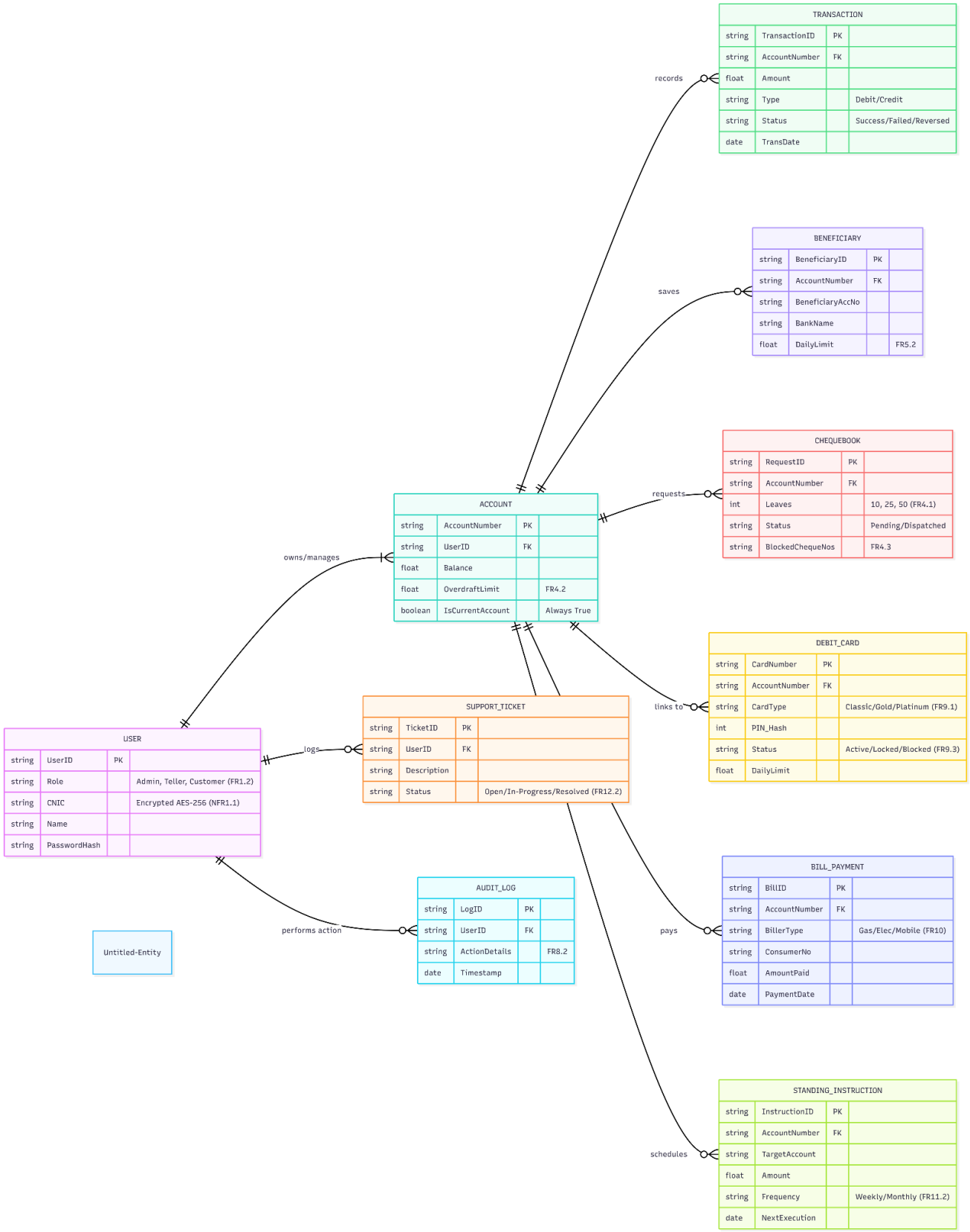

## DFD Level 0


Bank Teller
Cash Deposit Details
2.0 Transactions &\nTransfers FR2, FR3
Check/Update Balance
Save Receipt/Log
Set Overdraft Limits
Reverse Transaction
Bank Admin
1.0 Auth & Access\nFR1
Verify User/Role
D1: User & Account DB
Database Backup Request
Export Backups
Credentials
6.0 Admin & Audit\nFR8
Write System Logs
Update Card/Bill Status
Transfer Details
Customer
Bill Pay, Scheduled Inst.
3.0 Value-Added Services\nFR9, FR10, FR11
Deduct Bill Amount
D2: Transaction Ledger
Log Auto-Payment
Add Beneficiary, Cheque Req
4.0 Cheque & Beneficiary\nFR4, FR5
Save Beneficiary/Cheque
D3: Services & Tickets DB
Fetch History
Report Req, Submit Complaint
5.0 Reports & Support\nFR6, FR12
Log Ticket

# DFD Level 1


Customer
Transfer Request
(Acc No, Amount)
Error: Invalid Acc
2.1 Validate
Beneficiary FR3.1
Error: Low Balance
Account Valid
2.2 Check Balance
& 0% Interest FR3.2
Digital Receipt PDF
Error: Transfer Failed
Funds Sufficient
2.3 Execute
Transfer
Verify Target Account
Transfer Success
Fetch Sender Balance
System/Network Failure
2.4 Generate
Receipt FR3.3
Debit Sender &
Credit Receiver
2.5 Transaction
Reversal FR3.4
Save Transaction Record
Rollback Funds
(ACID Compliance)
Log Failure
D1: User & Account DB
D2: Transaction Ledger

# DFD level 2




# Class Diagram

# Appendix C: Project Estimations (Cocomo Model)

To provide a comprehensive analysis of the project scope, we have performed estimations using both the **Basic COCOMO** and the **Intermediate (Advanced) COCOMO** models as defined by Boehm (1981).

### 1. Project Classification

Based on the project characteristics defined in the course materials, we classify this system as **Organic Mode**:

- **Project Size:** Small (Typically 2–50 KLOC).
- **Team Experience:** Small team with experience in a familiar environment (Web Development).
- **Innovation:** Little to Medium.

### 2. Basic COCOMO Estimation (Initial Rough Estimate)

The Basic model provides a quick estimate based solely on the size of the code.

**Assumptions:**

- **Estimated Size (KLOC):** 3,000 Lines of Code (3 KLOC).
- **Coefficients (Organic Mode):**
  - a_b = 2.4
  - b_b = 1.05
  - c_b = 2.5
  - d_b = 0.38

**A. Effort Calculation (E)** Formula: E = a_b × (KLOC)^b_b  E = 2.4 × (3)^1.05 E = 2.4 × 3.17 **E ≈ 7.61 Person-Months**

**B. Development Time (D)** Formula: D = c_b × (E)^d_b  D = 2.5 × (7.61)^0.38 **D ≈ 5.4 Months**

### 3. Intermediate COCOMO Estimation (Advanced Refinement)

The Intermediate model is more accurate because it introduces an **Effort Adjustment Factor (EAF)** based on Cost Drivers.

**A. Cost Driver Analysis (EAF Calculation)**

We have selected the following ratings based on the **Current Account Banking System** requirements:

- **Required Software Reliability (RELY): High (1.15)**
    - *Reason:* Banking systems require high reliability to prevent financial loss.
- **Product Complexity (CPLX): High (1.15)**
    - *Reason:* Features like ACID compliance and encryption add complexity.
- **Modern Programming Practices (MODP): High (0.91)**
    - *Reason:* We are using modern frameworks (React/Node.js) which increases efficiency.
- **All other drivers: Nominal (1.00)**

*Formula for EAF:* 1.15 × 1.15 × 0.91

**EAF = 1.20**

**B. Revised Effort Calculation** For Intermediate Organic Mode, the coefficient *a_i* changes to **3.2**.

## Formula:

E = a_i × (KLOC)^b_i × EAF

E = 3.2 × (3)^1.05 × 1.20

E = 3.2 × 3.17 × 1.20

**E ≈ 12.17 Person-Months**

**C. Revised Development Time** Formula: D = c_i × (E)^d_i  D = 2.5 × (12.17)^0.38 **D ≈ 6.4 Months**

### 4. Comparative Analysis & Final Schedule

| Parameter | Basic Model | Intermediate Model |
|---|---|---|
| **Effort** | 7.61 PM | 12.17 PM |
| **Duration** | 5.4 Months | 6.4 Months |
| **Staff Size** | 1.4 Persons | ~2 Persons |
| **Productivity** | 394 LOC/PM | 246 LOC/PM |

# Conclusion & Feasibility Statement

The Intermediate COCOMO Model provides a significantly more realistic projection for this project because it incorporates critical Cost Drivers (EAF), specifically the high reliability (RELY) and operational complexity (CPLX) inherent to secure financial software.

- **Final Effort Estimate:** Based on the Intermediate calculations, the total required work amounts to **12.17 Person-Months**.
- **Team Allocation & Schedule Compression:** While a single developer would require over a year to complete this system, our dedicated development team consists of **5 members**. By distributing the modules evenly and utilizing parallel development strategies (such as the Sandwich Integration testing approach), the calendar schedule compresses down to approximately **2.4 to 2.5 months**.
- **Final Verdict:** This estimated duration fits perfectly within our academic semester timeline. Therefore, the Current Account Banking System is officially deemed both technically and practically feasible for our group to successfully deliver.