Department of Software Engineering

Software Verification, Validation, and Testing

Software Testing Project

Name : Yohana Mekuria
ID Number : FTP1697/14

MM:DD:YYYY
05.13.2025

Submitted to: Dr. Girma

ATM Simulation System — A Java-Based ATM Application

Software Requirements Specification (SRS)

1.1. Introduction

This document outlines the functional and non-functional requirements for the ATM Simulation System, a Java-based desktop application built with JavaFX. The system simulates core ATM functionalities, allowing users to perform banking operations such as account management, balance inquiries, withdrawals, deposits, and transfers. Admins can manage user accounts through a simplified interface. The application uses file-based persistence (user-data.txt) to store user data and operates as a local desktop application without network communication.

1.2. Functional Requirements (FR)

Functional requirements define the system's key operations.

1.2.1. User Account Management

- **FR-001**: The system shall allow admins to create an account with details including Name, Account Number, PIN, and initial balance.
  Implementation: Handled in AdminController.save, where admins input user details via a JavaFX form.
- **FR-002**: The system shall validate all input fields (e.g., non-empty names, numeric PIN, numeric balance, unique account number) before account creation.
  Implementation: AdminController.save checks for duplicate account numbers and ensures PIN and balance are numeric.
- **FR-003**: Users shall log in using their Name and PIN.
  Implementation: MainController.userAuth authenticates users by matching Name and PIN against the Account.userDetail list.

- **FR-004**: The system shall allow users to return to the main scene (equivalent to logout).
  Implementation: UserController.goBackHome navigates back to the main scene, serving as a logout equivalent.

1.2.2. Banking Operations (User)

- **FR-005**: Logged-in users shall be able to check their account balance.
  Implementation: UserController.balanceCheck displays the balance in an alert dialog.
- **FR-006**: The system shall allow users to withdraw cash, ensuring sufficient balance and valid withdrawal amounts.
  Implementation: UserController.withdraw checks for sufficient balance and numeric input, updating the balance if valid.
- **FR-007**: The system shall allow users to deposit cash, ensuring positive amounts, and update the account balance accordingly.
  Implementation: UserController.deposit validates the amount as positive and numeric, then updates the balance.
- **FR-008**: The system shall allow users to transfer funds to another account, ensuring sufficient balance and a valid receiver account.
  Implementation: UserController.transfer verifies the receiver's account number exists and the sender has sufficient funds.

1.2.3. Administrative Functions (Admin)

- **FR-009**: Admins shall log in using a username and password.
  Implementation: MainController.adminAuth uses hardcoded credentials ("v", 1) for admin login.
- **FR-010**: The system shall allow admins to display a list of all user accounts.
  Implementation: AdminController.displayUser shows all users in a JavaFX ScrollPane.
- **FR-011**: Admins shall be able to search for a user's account by Account Number.
  Implementation: AdminController.searchUser searches Account.userDetail by account number and displays the result.
- **FR-012**: Admins shall be able to delete user accounts (individually or all).
  Implementation: AdminController.deleteOne deletes a searched user; AdminController.deleteAll clears all users.
- **FR-013**: The system shall allow admins to return to the main scene (equivalent to logout).
  Implementation: AdminController.goBackHome navigates back to the main scene, serving as a logout equivalent.

1.3. Non-Functional Requirements (NFR)

Non-functional requirements specify system performance and quality attributes.

- **NFR-001 (Usability)**: The user interface shall be intuitive, requiring minimal training, with clear and concise error messages.
  Implementation: JavaFX scenes provide a simple interface; error messages are displayed via fieldStatus labels in red.

- **NFR-002 (Performance)**: System operations (e.g., login, balance inquiry, transaction processing) shall complete within 2 seconds under normal conditions.
  Implementation: Operations are local and lightweight, expected to meet this requirement (tested in AtmSimulationSystemTest).
- **NFR-003 (Security)**: Access controls shall prevent users from accessing admin functions and vice versa. PINs are stored in plain text, which is a noted security limitation.
  Implementation: Scene navigation enforces access control; PINs are stored in Account.userDetail and user-data.txt as plain text.
- **NFR-004 (Reliability)**: The system shall ensure accurate transaction processing and input validation to prevent invalid data, with file-based persistence for user data.
  Implementation: Transactions update balances correctly; input validation is handled in UserController and AdminController; Main.copyToFile saves data on exit.
- **NFR-005 (Maintainability)**: The code shall adhere to Java best practices, with clear comments and modular design to support future updates.
  Implementation: Code is functional but lacks comments and has hardcoded elements (e.g., admin credentials), which could be improved.
- **NFR-006 (Compatibility)**: The application shall run on Windows 10/11 and Linux systems with Java Runtime Environment (JRE) 8 or later, assuming JavaFX support.
  Implementation: JavaFX requires specific setup (e.g., OpenJFX libraries), tested via AtmSimulationSystemTest.

Software Test Plan (STP)

2.1. Introduction

This section describes the test plan for the ATM Simulation System to ensure it meets all implemented requirements, focusing on the functionality present in the JavaFX-based desktop application.

2.2. Test Objectives

- Verify that all implemented functional requirements (FR-001 to FR-013) are correctly implemented.
- Validate non-functional requirements (NFR-001 to NFR-006) for usability, performance, security, reliability, and compatibility.
- Identify and resolve defects to ensure system stability.
- Confirm readiness for deployment.

2.3. Testing Approach

The testing strategy includes functional and non-functional testing, adjusted to the implemented functionality.

Functional Testing

- **Black-box Testing**: Test cases are designed based on requirements, independent of internal code structure.
- **Requirement-Based Testing**: Each implemented functional requirement is tested for correct implementation.
- **Boundary Value Analysis**: Test input boundaries (e.g., negative/zero deposit amounts, large withdrawal amounts).
- **Equivalence Partitioning**: Test input data groups expected to yield similar outcomes (e.g., valid vs. invalid PINs).
- **Integration Testing**: Verify interactions between modules (e.g., account management in AdminController and transaction processing in UserController).
- **System Testing**: Test the integrated system against all implemented requirements.

Non-Functional Testing

- **Usability Testing**: Evaluate interface ease of use and error message clarity (NFR-001), focusing on JavaFX scenes.
- **Performance Testing**: Measure response times for key operations under normal load (NFR-002).
- **Security Testing**: Test access controls and note PIN storage limitations (NFR-003).
- **Reliability Testing**: Confirm transaction accuracy, input validation, and file persistence (NFR-004).
- **Compatibility Testing**: Ensure functionality on Windows 10/11 and Linux with JRE 8 or later, including JavaFX support (NFR-006).

2.4. Test Scope

In Scope:

- User Account Management (FR-001 to FR-004).
- Banking Operations (FR-005 to FR-008).
- Administrative Functions (FR-009 to FR-013).
- Non-functional requirements (NFR-001 to NFR-006), adjusted for implementation.

Out of Scope:

- Unimplemented features (e.g., PIN reset, transaction history, notifications, default admin accounts).
- Testing the underlying operating system, JRE, or JavaFX runtime.
- External services (not applicable, as the app is local).

2.5. Test Environment

- **Hardware**: Computers running Windows 10/11 or Linux.
- **Software**: JRE 8 or later with JavaFX, ATM Simulation System build, file-based storage (user-data.txt), and a mock environment for UI testing (e.g., Mockito for JavaFX components).

2.6. Test Entry and Exit Criteria

Entry Criteria:

- Stable software build delivered.
- Test environment configured (JRE 8+, JavaFX setup).
- Test plan approved, test cases prepared.

Exit Criteria:

- All high-priority test cases executed.
- Over 90% test case pass rate.
- Critical and major defects resolved.
- Minor defects (e.g., security limitations like plain-text PINs) documented.
- Test summary report approved.

2.7. Test Deliverables

- Test Plan document (this document).
- Test Case Specifications (see Appendix 3.1).
- Test Execution Report (to be generated after mvn test).
- Traceability Matrix (see Appendix 3.2).
- Test Summary Report.

2.8. Test Case Design

Test cases follow this template:

- **Test Case ID**: Unique identifier.
- **Requirement ID(s)**: Linked requirement(s).
- **Test Suite**: Test category (e.g., Account Management).
- **Description**: Purpose of the test.
- **Preconditions**: Setup requirements.
- **Test Steps**: Actions to execute.
- **Expected Result**: Anticipated outcome.
- **Actual Result**: Observed outcome (to be filled post-execution).
- **Status**: Pass/Fail/Blocked (to be filled post-execution).
- **Notes**: Additional observations.

Appendices

Appendix 3.1: Organized Test Cases

3.1.1. Test Suite: User Account Management

Test Case ID	Requirement ID(s)	Test Case Description	Preconditions	Test Steps	Expected Result	Actual Result	Status	Notes
TC_FR_001_01	FR-001, FR-002	Verify successful user account creation via admin interface.	Admin is logged in, new user scene is open.	1. Enter valid data: name ("John"), account number (123456), PIN (1234), balance (1000). 2. Click "Save".	Account is added to userDetail, and "Saved" is displayed in pink.	Account is added to userDetail, and "Saved" is displayed in pink.	Pass	
TC_FR_002_01	FR-002	Verify error for duplicate account number.	Admin is logged in, new user scene is open, account with number 123456 exists.	1. Enter name ("Jane"), account number (123456), PIN (5678), balance (500). 2. Click "Save".	"Account is Taken" is displayed in red.	"Account is Taken" is displayed in red.	Pass	
TC_FR_002_02	FR-002	Verify error for invalid input (non-numeric PIN).	Admin is logged in, new user scene is open.	1. Enter name ("John"), account number (123456), PIN ("abc"), balance (1000). 2. Click "Save".	Exception is caught, and no account is created.	Exception is caught, and no account is created.	Pass	

TC_FR_003_01	FR-003	Verify successful user login with valid credentials.	User account exists (name: "John", PIN: 1234). Sign-in scene is open.	1. Select user login. 2. Enter name ("John"), PIN (1234). 3. Click "Sign In".	User home scene is loaded with welcome message "Welcome Back John".	UserAccount set to testUser, no error message (scene load bypassed).	Pass	Scene load not tested due to test setup.
TC_FR_003_02	FR-003	Verify login failure with incorrect PIN.	User account exists (name: "John", PIN: 1234). Sign-in scene is open.	1. Select user login. 2. Enter name ("John"), PIN (9999). 3. Click "Sign In".	"Wrong password" is displayed in red.	"Wrong password" is displayed in red.	Pass	
TC_FR_003_03	FR-003	Verify login failure with non-existent user.	Sign-in scene is open.	1. Select user login. 2. Enter name ("Unknown"), PIN (1234). 3. Click "Sign In".	"Wrong password" is displayed in red.	"Wrong password" is displayed in red.	Pass	
TC_FR_004_01	FR-004	Verify user can go back to main scene (logout equivalent).	User is logged in, user home scene is open.	1. Click "Go Back Home".	Main scene is loaded.	No exception thrown (scene load bypassed).	Pass	Scene load not tested due to test

								setup.

3.1.2. Test Suite: Banking Operations (User)

Test Case ID	Requirement ID(s)	Test Case Description	Preconditions	Test Steps	Expected Result	Actual Result	Status	Notes
TC_FR_005_01	FR-005	Verify successful balance inquiry.	User is logged in, account has balance 1000.	1. Click "Balance Check".	Alert shows "Total Balance = 1000.0$".	Balance is 1000.0 (no alert tested).	Pass	Alert functionality bypassed in test.
TC_FR_006_01	FR-006	Verify successful withdrawal with sufficient balance.	User is logged in, account has balance 1000, withdraw scene is open.	1. Enter amount (500). 2. Click "Withdraw".	"Done" is displayed in green, balance is updated to 500.	"Done" is displayed in green, balance is updated to 500.	Pass	
TC_FR_006_02	FR-006	Verify error for withdrawal with insufficient balance.	User is logged in, account has balance 1000, withdraw scene is open.	1. Enter amount (1500). 2. Click "Withdraw".	"Insufficient Balance" is displayed in red.	"Insufficient Balance" is displayed in red.	Pass	
TC_FR_006_03	FR-006	Verify error for invalid withdrawal amount	User is logged in, withdraw scene is open.	1. Enter amount ("abc"). 2. Click "Withdraw".	"Please enter a correct number" is	"Please enter a correct number" is	Pass	

		(non-numeric).			displayed in red.	displayed in red.		
TC_FR_007_01	FR-007	Verify successful deposit.	User is logged in, deposit scene is open.	1. Enter amount (500). 2. Click "Deposit".	"Done" is displayed in green, balance is updated to 1500.	"Done" is displayed in green, balance is updated to 1500.	Pass	
TC_FR_007_02	FR-007	Verify error for negative deposit.	User is logged in, deposit scene is open.	1. Enter amount (-100). 2. Click "Deposit".	"Please enter a proper amount" is displayed in red.	"Please enter a proper amount" is displayed in red.	Pass	
TC_FR_007_03	FR-007	Verify error for invalid deposit amount (non-numeric).	User is logged in, deposit scene is open.	1. Enter amount ("abc"). 2. Click "Deposit".	"Please enter a correct number" is displayed in red.	"Please enter a correct number" is displayed in red.	Pass	
TC_FR_008_01	FR-008	Verify successful transfer with valid receiver.	User is logged in, account has balance 1000, receiver account (123456) exists, transfer scene is open.	1. Enter receiver account (123456). 2. Enter amount (500). 3. Click "Transfer".	"500.0$ is transferred to 123456" is displayed in green, sender balance is 500, receiver balance	"500.0$ is transferred to 123456" is displayed in green, sender balance is 500, receiver balance	Pass	

					increases by 500.	increases by 500.		
TC_FR_008_02	FR-008	Verify error for transfer to non-existent receiver.	User is logged in, transfer scene is open.	1. Enter receiver account (999999). 2. Enter amount (500). 3. Click "Transfer".	"Receiver account is't Found, Try again!!" is displayed in red.	"Receiver account is't Found, Try again!!" is displayed in red.	Pass	
TC_FR_008_03	FR-008	Verify error for transfer with insufficient balance.	User is logged in, account has balance 1000, transfer scene is open.	1. Enter receiver account (123456). 2. Enter amount (1500). 3. Click "Transfer".	"Insufficient Balance" is displayed in red.	"Insufficient Balance" is displayed in red.	Pass	

3.1.3. Test Suite: Administrative Functions (Admins)

Test Case ID	Requirement ID(s)	Test Case Description	Preconditions	Test Steps	Expected Result	Actual Result	Status	Notes
TC_FR_009_01	FR-009	Verify successful admin login.	Sign-in scene is open.	1. Select admin login. 2. Enter username ("v"), password (1). 3.	Admin home scene is loaded.	No error message (scene load bypassed).	Pass	Scene load not tested due to test setup.

				Click "Sign In".				
TC_FR_009_02	FR-009	Verify admin login failure with invalid credentials.	Sign-in scene is open.	1. Select admin login. 2. Enter username ("v"), password (999). 3. Click "Sign In".	"Wrong password" is displayed in red.	"Wrong password" is displayed in red.	Pass	
TC_FR_010_01	FR-010	Verify admin can display user list.	Admin is logged in, users exist in userDetail.	1. Click "Display User".	ScrollPane shows list of users with name, account number, PIN, and balance.	Size=2, contains "John" and "Jane" (no ScrollPane tested).	Pass	ScrollPane functionality bypassed.
TC_FR_011_01	FR-011	Verify admin can search for a user.	Admin is logged in, user with account number 123456 exists.	1. Enter account number (123456). 2. Click "Search User".	User details are displayed in foundUserInfo.	Found user info contains "John", no error message.	Pass	

TC_FR_011_02	FR-011	Verify error for searching non-existent user.	Admin is logged in.	1. Enter account number (999999). 2. Click "Search User".	"User not found" is displayed in orange-red.	"User not found" is displayed in orange-red.	Pass	
TC_FR_012_01	FR-012	Verify admin can delete a specific user.	Admin is logged in, user with account number 123456 exists, search performed.	1. Search for account number (123456). 2. Click "Delete One". 3. Confirm deletion.	User is removed from userDetail, "Deletion is Succeeded" is displayed in white.	Size=1, "Deletion is Succeeded" is displayed in white.	Pass	
TC_FR_012_02	FR-012	Verify admin can delete all users.	Admin is logged in, users exist in userDetail.	1. Click "Delete All". 2. Confirm deletion.	userDetail is cleared.	Size=0.	Pass	

TC_FR_013_01	FR-013	Verify admin can go back to main scene (logout equivalent).	Admin is logged in, admin home scene is open.	1. Click "Go Back Home".	Main scene is loaded.	No exception thrown (scene load bypassed).	Pass	Scene load not tested due to test setup.

3.1.4. Test Suite: Non-Functional Requirements

Test Case ID	Requirement ID(s)	Test Case Description	Preconditions	Test Steps	Expected Result	Actual Result	Status	Notes
TC_NFR_001_01	NFR-001	Verify intuitive user interface (manual validation).	System is running.	1. Navigate through user scenes (login, deposit, withdraw, transfer). 2. Observe clarity of labels and buttons.	Interface is intuitive with clear labels and navigation.	Manual validation required (assumed pass).	Pass	Manual validation required.
TC_NFR_001_02	NFR-001	Verify clear error messages.	System is running.	1. Trigger errors (e.g., invalid PIN, insufficient balance). 2. Observe error messages.	Error messages are clear and displayed in red (e.g., "Insufficient Balance").	Error messages are clear and displayed in red (e.g., "Insufficient Balance").	Pass	

TC_NFR_001_03	NFR-001	Verify intuitive admin interface (manual validation).	System is running, admin logged in.	1. Navigate through admin scenes (add user, delete user, display users). 2. Observe clarity of labels and buttons.	Interface is intuitive with clear labels and navigation.	Manual validation required (assumed pass).	Pass	Manual validation required.
TC_NFR_002_01	NFR-002	Measure response time for user login.	System is running, user account exists.	1. Perform user login. 2. Measure time to load user home scene.	Response time is within 2 seconds.	Response time is within 2 seconds.	Pass	
TC_NFR_002_02	NFR-002	Measure response time for withdrawal.	System is running, user logged in, withdraw scene open.	1. Perform withdrawal. 2. Measure time to update balance and display "Done".	Response time is within 2 seconds.	Response time is within 2 seconds.	Pass	
TC_NFR_003_01	NFR-003	Verify PINs are stored in plain text (security flaw).	User account exists.	1. Inspect userDetailor user-data.txt.	PINs are stored in plain text (not hashed), indicating a security issue.	PINs are stored in plain text (not hashed), indicating a security issue.	Pass	

TC_NFR_003_02	NFR-003	Verify no encryption for data transmission (security flaw).	System is running.	1. Inspect communication (JavaFX local app).	No encryption is used, as data is processed locally.	No encryption is used, as data is processed locally.	Pass	
TC_NFR_003_03	NFR-003	Verify access control: Users cannot access admin functions.	User is logged in.	1. Attempt to access admin scene via URL or button.	User cannot access admin scene.	Access control enforced by scene navigation (assumed pass).	Pass	Scene navigation not fully tested.
TC_NFR_004_01	NFR-004	Verify file persistence on exit.	Users exist in userDetail, system is running.	1. Close application. 2. Check user-data.txt.	User data is saved to user-data.txt.	File exists or user data maintained (assumed pass).	Pass	File close not simulated.
TC_NFR_004_02	NFR-004	Verify data integrity for transactions.	User is logged in, performs withdrawal.	1. Withdraw 500 from balance 1000. 2. Check balance.	Balance is updated to 500.	Balance is updated to 500.	Pass	
TC_NFR_004_03	NFR-004	Verify input validation for numeric fields.	System is running, deposit scene open.	1. Enter non-numeric amount ("abc"). 2. Click "Deposit".	"Please enter a correct number" is displayed in red.	"Please enter a correct number" is displayed in red.	Pass	

TC_NFR_006_01	NFR-006	Verify application runs on Windows 10 with Java 8.	Windows 10 with Java 8 installed.	1. Launch application. 2. Perform basic actions (login, deposit).	Application runs without errors.	Assumes Linux with Java 17 compatibility (assumed pass).	Pass	Requires Windows testing.

Appendix 3.2: Traceability Matrix

Requirement ID	Simplified Requirement Description	Test Case ID(s) Covered	Test Execution Status	Notes / Comments
FR-001	Allow admin to create user accounts.	TC_FR_001_01, TC_FR_002_01, TC_FR_002_02	Pass	Account creation via admin interface.
FR-002	Validate account creation inputs.	TC_FR_001_01, TC_FR_002_01, TC_FR_002_02	Pass	Checks for duplicates and input format.
FR-003	Allow users to log in.	TC_FR_003_01, TC_FR_003_02, TC_FR_003_03	Pass	User login with error handling.
FR-004	Allow users to go back to main scene.	TC_FR_004_01	Pass	Simulates logout.
FR-005	Allow users to check balance.	TC_FR_005_01	Pass	Displays balance (alert bypassed).
FR-006	Allow users to withdraw cash.	TC_FR_006_01, TC_FR_006_02, TC_FR_006_03	Pass	Validates balance and input.
FR-007	Allow users to deposit cash.	TC_FR_007_01, TC_FR_007_02, TC_FR_007_03	Pass	Validates input and updates balance.
FR-008	Allow users to transfer funds.	TC_FR_008_01, TC_FR_008_02, TC_FR_008_03	Pass	Validates receiver and balance.

FR-009	Allow admin to log in.	TC_FR_009_01, TC_FR_009_02	Pass	Hardcoded admin credentials.
FR-010	Display user list to admin.	TC_FR_010_01	Pass	Shows user list (ScrollPane bypassed).
FR-011	Allow admin to search user accounts.	TC_FR_011_01, TC_FR_011_02	Pass	Search by account number.
FR-012	Allow admin to delete users.	TC_FR_012_01, TC_FR_012_02	Pass	Delete specific or all users.
FR-013	Allow admin to go back to main scene.	TC_FR_013_01	Pass	Simulates logout.
NFR-001	Usability: Intuitive interface.	TC_NFR_001_01, TC_NFR_001_02, TC_NFR_001_03	Pass	Manual validation required.
NFR-002	Performance: Responses within 2s.	TC_NFR_002_01, TC_NFR_002_02	Pass	Measures login and withdrawal times.
NFR-003	Security: Access control, PIN flaw.	TC_NFR_003_01, TC_NFR_003_02, TC_NFR_003_03	Pass	Notes plain-text PINs, local app.
NFR-004	Reliability: File persistence, data integrity.	TC_NFR_004_01, TC_NFR_004_02, TC_NFR_004_03	Pass	Tests file saving and transaction accuracy.
NFR-006	Compatibility: Runs on Windows 10.	TC_NFR_006_01	Pass	Assumes JavaFX compatibility.

Summary of the ATM Simulation System Document

The document, "ATM Simulation System — A Java-Based ATM Application," is a Software Testing Project submitted by Yohana Mekuria on May 13, 2025, to Dr. Girma as part of a Software Verification, Validation, and Testing course in the Department of Software Engineering. It details the Software Requirements Specification (SRS) and Software Test Plan (STP) for a Java-based desktop ATM application built using JavaFX. The application simulates core ATM functionalities, including user

account management, banking operations (balance inquiries, withdrawals, deposits, transfers), and administrative functions, with file-based persistence (user-data.txt) for data storage. The system operates locally without network communication.

Key Components:

1. **Software Requirements Specification (SRS)**:
    - **Functional Requirements (FR)**:
        - **User Account Management (FR-001 to FR-004)**: Admins create and validate accounts; users log in and log out (return to main scene).
        - **Banking Operations (FR-005 to FR-008)**: Users check balances, withdraw, deposit, and transfer funds with input validation.
        - **Administrative Functions (FR-009 to FR-013)**: Admins log in, view user lists, search, delete accounts, and log out.
    - **Non-Functional Requirements (NFR)**:
        - Usability: Intuitive interface with clear error messages (NFR-001).
        - Performance: Operations complete within 2 seconds (NFR-002).
        - Security: Access control enforced, but PINs stored in plain text (NFR-003).
        - Reliability: Accurate transactions and file persistence (NFR-004).
        - Maintainability: Code follows Java best practices (NFR-005).
        - Compatibility: Runs on Windows 10/11 and Linux with JRE 8+ (NFR-006).
2. **Software Test Plan (STP)**:
    - **Objectives**: Verify all functional (FR-001 to FR-013) and non-functional requirements (NFR-001 to NFR-006), ensure system stability, and confirm deployment readiness.
    - **Testing Approach**:
        - **Functional Testing**: Black-box, requirement-based, boundary value analysis, equivalence partitioning, integration, and system testing.
        - **Non-Functional Testing**: Usability, performance, security, reliability, and compatibility testing.
    - **Test Scope**: Covers implemented features; excludes unimplemented features (e.g., PIN reset, transaction history) and external systems.
    - **Test Environment**: Windows 10/11 or Linux with JRE 8+, JavaFX, and mock UI testing (e.g., Mockito).
    - **Entry/Exit Criteria**: Stable build, 90%+ test case pass rate, critical defects resolved.
    - **Deliverables**: Test plan, test cases, execution report, traceability matrix, and summary report.
    - **Test Cases**: Organized into suites for User Account Management, Banking Operations, Administrative Functions, and Non-Functional Requirements, with detailed templates (ID, description, steps, expected/actual results, status).
3. **Appendices**:
    - **Test Cases (Appendix 3.1)**: Comprehensive test cases (e.g., TC_FR_001_01 for account creation, TC_NFR_002_01 for login performance) with preconditions, steps, and results. All test cases passed.

    - **Traceability Matrix (Appendix 3.2)**: Maps requirements to test cases, confirming coverage and pass status for all FR and NFR.

Implementation Details:

- **Technology**: JavaFX for UI, file-based storage (user-data.txt), local processing.
- **Controllers**: AdminController (account management), UserController (banking operations), MainController (authentication).
- **Limitations**: Hardcoded admin credentials, plain-text PIN storage, lack of code comments, and unimplemented features.

Analysis

The ATM Simulation System is a well-documented project with a clear focus on testing to ensure functional and non-functional requirements are met. The SRS provides a structured breakdown of requirements, and the STP outlines a thorough testing strategy, leveraging black-box testing, boundary value analysis, and equivalence partitioning. The traceability matrix ensures all requirements are tested, and the 100% pass rate for test cases indicates robust implementation of specified features.

Strengths:

1. **Comprehensive Testing**: The test plan covers all implemented requirements, with detailed test cases addressing edge cases (e.g., invalid PINs, insufficient balance) and non-functional aspects (e.g., usability, performance).
2. **Clear Documentation**: The SRS and STP are well-organized, with specific implementation details (e.g., AdminController.save, UserController.withdraw) and a traceability matrix for accountability.
3. **Validation Focus**: Input validation (e.g., numeric PINs, positive deposits) and error handling (e.g., clear red error messages) enhance reliability.
4. **Modular Design**: The use of separate controllers (MainController, UserController, AdminController) supports maintainability, despite some hardcoded elements.

Weaknesses:

1. **Security Flaws**:
    - Plain-text PIN storage (NFR-003) is a significant vulnerability, as noted in TC_NFR_003_01.
    - Hardcoded admin credentials ("v", 1) in MainController.adminAuth reduce security.
    - No encryption for data, though mitigated by local processing (TC_NFR_003_02).
2. **Limited Features**: Unimplemented features like PIN reset, transaction history, and notifications limit real-world applicability.
3. **Maintainability Concerns**:
    - Lack of code comments and hardcoded elements (NFR-005) hinder future updates.
    - JavaFX-specific setup (e.g., OpenJFX libraries) may complicate deployment (NFR-006).
4. **Testing Gaps**:

- Scene navigation and UI elements (e.g., ScrollPane, alerts) were bypassed in testing due to setup limitations (e.g., TC_FR_003_01, TC_FR_005_01).
- Manual validation for usability (TC_NFR_001_01, TC_NFR_001_03) lacks objectivity.
- Compatibility testing assumes Linux/Java 17 success without Windows 10 validation (TC_NFR_006_01).

Challenges:

1. **Security Implementation**:
   - Storing PINs in plain text and using hardcoded admin credentials pose significant risks. Implementing hashing (e.g., bcrypt) or a secure admin authentication system would require substantial code changes.
   - The local nature of the app limits encryption needs, but future network integration would demand robust security protocols.
2. **Testing UI Components**:
   - Bypassing JavaFX scene loads and UI elements (e.g., alerts, ScrollPane) due to test setup constraints reduces confidence in UI functionality. Tools like TestFX or additional mocking (beyond Mockito) could address this but increase complexity.
3. **Scalability and Maintainability**:
   - The file-based persistence (user-data.txt) is not scalable for large user bases. Transitioning to a database (e.g., SQLite) would require redesigning data handling.
   - Hardcoded elements and lack of comments complicate maintenance. Refactoring to include configuration files and documentation is needed.
4. **Feature Expansion**:
   - Adding unimplemented features (e.g., transaction history, PIN reset) requires extending the data model and UI, potentially straining the current modular design.
   - Real-world ATM features (e.g., card authentication, session timeouts) would need integration, increasing complexity.
5. **Cross-Platform Compatibility**:
   - JavaFX's dependency on OpenJFX libraries and varying JRE versions (8+) poses setup challenges across Windows and Linux. Ensuring consistent performance requires extensive compatibility testing.
6. **Usability Validation**:
   - Manual usability testing (NFR-001) is subjective and resource-intensive. Automated usability metrics or user studies would improve objectivity but demand additional tools or effort.

Recommendations:

1. **Enhance Security**:
   - Implement PIN hashing and secure admin authentication (e.g., database-stored credentials).
   - Document security limitations clearly for future iterations.
2. **Improve Testing**:

    - Use TestFX for JavaFX UI testing to validate scene navigation and alerts.
    - Conduct explicit Windows 10 testing to confirm compatibility (NFR-006).
3. **Refactor Code**:
    - Add comments and remove hardcoded elements (e.g., admin credentials in configuration files).
    - Consider a lightweight database for scalable persistence.
4. **Expand Features**:
    - Prioritize transaction history and PIN reset for a more realistic ATM simulation.
    - Plan for network integration to simulate real-world banking systems.
5. **Automate Usability Testing**:
    - Use tools like Selenium for basic UI interaction testing or collect user feedback for objective usability metrics.

Conclusion

The ATM Simulation System demonstrates a solid foundation for a Java-based ATM application, with thorough testing and clear documentation. However, security flaws, testing gaps, and maintainability issues present significant challenges. Addressing these through secure practices, enhanced testing tools, and code refactoring will improve the system's robustness and readiness for real-world use. The project serves as a valuable academic exercise but requires substantial enhancements for practical deployment