Gemini Chat

SE_Practic...tions.docx
PDF

give all the answer of this

Here are the answers to all the questions from the provided document.

---

1. SRS Outline for Online Library Management System

Here is a sample Software Requirements Specification (SRS) outline based on the IEEE 830 format, with key requirements highlighted.

1. Introduction * 1.1 Purpose * 1.2 Scope (System will manage book inventory, member accounts, and book lending) * 1.3 Definitions, Acronyms, and Abbreviations * 1.4 References * 1.5 Overview

2. Overall Description * 2.1 Product Perspective (Web-based system, replacing manual R-card system) * 2.2 Product Functions (User Registration, Login, Book Search, Issue, Return, Fine Calculation) * 2.3 User Characteristics (Librarian/Admin, Student/Member) * 2.4 Constraints (e.g., Must run on standard web browsers, data stored in SQL database) * 2.5 Assumptions and Dependencies

3. Specific Requirements * 3.1 External Interface Requirements * 3.1.1 User Interfaces (Web portal must be responsive and accessible) * 3.1.2 Hardware Interfaces (Barcode scanner for book issue/return) * 3.1.3 Software Interfaces (Email gateway for sending notifications) * **3.2 Functional Requirements** * 3.2.1 **User Authentication** * REQ-F1: The system shall allow members and librarians to log in with a unique ID and password. * REQ-F2: Librarians shall have access to an admin dashboard. * 3.2.2 **Book Management (Admin)** * REQ-F3: The librarian shall be able to add, update, and remove book titles and copies from the inventory. * 3.2.3 **Member Management (Admin)** * REQ-F4: The librarian shall be able to register new members and deactivate existing members. * 3.2.4 **Book Search (Member)** * REQ-F5: Members shall be able to search for books by title, author, or ISBN. * 3.2.5 **Book Lending** * REQ-F6: The system shall allow a librarian to issue a book to a member, updating the book's status to "On Loan." * REQ-F7: The system shall enforce a borrowing limit (e.g., max 3 books per member). * REQ-F8: The system shall allow a librarian to process a book return. * 3.2.6 **Fine Calculation** * REQ-F9: The system shall automatically calculate and apply fines for overdue books. * **3.3 Non-Functional Requirements** * 3.3.1 **Performance** * REQ-NF1: Book search results must be displayed within 3 seconds. * REQ-NF2: The system must support 100 concurrent users without significant performance degradation. * 3.3.2 **Security** * REQ-NF3: All user passwords must be stored in a hashed and salted format. * REQ-NF4: The system must be protected against basic SQL injection attacks. * 3.3.3 **Usability** * REQ-NF5: The user interface must be intuitive, allowing a new member to find and reserve a book without training. * 3.R.4 **Reliability** * REQ-NF6: The system shall have an uptime of 99.9%.

4. Appendices

---

2. Structured Analysis and Student Attendance DFDs

Steps to Perform Structured Analysis using DFDs

Structured Analysis is a top-down method for modeling system requirements. The key steps involving Data Flow Diagrams (DFDs) are:

1. **Create a Context Diagram (Level 0 DFD):** This is the highest-level view. It shows the **entire system as a single process** (labeled "0"). It identifies all **external entities** (people, departments, or other systems) that interact with the system and the major **data flows** that go in and out.
2. **Decompose to Level 1 DFD:** The single process from Level 0 is "exploded" or decomposed into its main sub-processes. This diagram shows:
   - The major **processes** within the system (e.g., "1.0 Mark Attendance," "2.0 Generate Reports").
   - The **data flows** between these processes and the external entities.
   - **Data stores** (e.g., "D1: Attendance_Records") that are introduced for the first time.
3. **Balance the DFDs:** A crucial rule is "balancing." The data flows going *into* and *out of* a process on a higher-level diagram (e.g., Process 0) must exactly match the data flows going *into* and *out of* the entire lower-level diagram (e.g., the Level 1 DFD).
4. **Decompose to Lower Levels (Level 2, 3...):** Any complex process on the Level 1 DFD (e.g., "2.0 Generate Reports") can be further decomposed into a Level 2 DFD (with processes like "2.1 Compile Class Report," "2.2 Generate Student Report"). This continues until each process represents a single, simple function.

DFDs for Student Attendance System

Level 0 DFD (Context Diagram)

- **External Entities:** `Student`, `Faculty`, `Admin`
- **Process:** `0. Student Attendance System`
- **Data Flows:**
  - `Faculty` -> `Attendance Data` -> Process 0
  - `Faculty` -> `Report Request` -> Process 0
  - Process 0 -> `Attendance Report` -> `Faculty`
  - `Admin` -> `Student/Course Info` -> Process 0
  - Process 0 -> `Defaulter List` -> `Admin`
  - `Student` -> `View Attendance` -> Process 0
  - Process 0 -> `Personal Report` -> `Student`

Level 1 DFD

- **External Entities:** `Student`, `Faculty`, `Admin`
- **Processes:**
  - `1.0 Record Attendance`
  - `2.0 Manage System Data`
  - `3.0 Generate Reports`
- **Data Stores:**
  - `D1: Attendance_Records`

    - `D2: Student_Data`
    - `D3: Course_Data`
- **Data Flows (Sample):**
    - `Faculty` -> `Attendance Data` -> `1.0 Record Attendance`
    - `1.0 Record Attendance` -> `New Record` -> `D1: Attendance_Records`
    - `Admin` -> `Student/Course Info` -> `2.0 Manage System Data`
    - `2.0 Manage System Data` -> `Student Updates` -> `D2: Student_Data`
    - `2.0 Manage System Data` -> `Course Updates` -> `D3: Course_Data`
    - `Faculty` -> `Report Request` -> `3.0 Generate Reports`
    - `3.0 Generate Reports` (reads from `D1`, `D2`, `D3`) -> `Attendance Report` -> `Faculty`
    - `Student` -> `View Attendance` -> `3.0 Generate Reports`
    - `3.0 Generate Reports` -> `Personal Report` -> `Student`

---

3. DFDs for Online Food Ordering System

Level 0 DFD (Context Diagram)

- **External Entities:** `Customer`, `Restaurant`, `Delivery Partner`
- **Process:** `0. Online Food Ordering System`
- **Data Flows:**
    - `Customer` -> `Order Request`, `Payment` -> Process 0
    - Process 0 -> `Order Confirmation`, `Status Update` -> `Customer`
    - Process 0 -> `New Order` -> `Restaurant`
    - `Restaurant` -> `Order Status` (e.g., "Accepted," "Ready") -> Process 0
    - Process 0 -> `Delivery Assignment` -> `Delivery Partner`
    - `Delivery Partner` -> `Delivery Status` (e.g., "Picked Up," "Delivered") -> Process 0

Level 1 DFD

- **External Entities:** `Customer`, `Restaurant`, `Delivery Partner`
- **Processes:**
    - `1.0 Manage Orders`
    - `2.0 Handle Payments`
    - `3.0 Manage Restaurants`
    - `4.0 Coordinate Delivery`
- **Data Stores:**
    - `D1: Orders`
    - `D2: Menu`
    - `D3: Customer_Accounts`
    - `D4: Restaurant_Data`
- **Data Flows (Sample):**
    - `Customer` -> `Order Request` -> `1.0 Manage Orders`
    - `1.0 Manage Orders` (reads from `D2: Menu`, `D3: Customer_Accounts`)

  - `1.0 Manage Orders` -> `Payment Details` -> `2.0 Handle Payments`
  - `Customer` -> `Payment` -> `2.0 Handle Payments`
  - `2.0 Handle Payments` -> `Payment Confirmation` -> `1.0 Manage Orders`
  - `1.0 Manage Orders` -> `Confirmed Order` -> `D1: Orders`
  - `1.0 Manage Orders` -> `New Order` -> `Restaurant`
  - `Restaurant` -> `Order Status` -> `1.0 Manage Orders`
  - `1.0 Manage Orders` -> `Updated Status` -> `D1: Orders`
  - `1.0 Manage Orders` -> `Ready for Pickup` -> `4.0 Coordinate Delivery`
  - `4.0 Coordinate Delivery` -> `Delivery Assignment` -> `Delivery Partner`
  - `Delivery Partner` -> `Delivery Status` -> `4.0 Coordinate Delivery`
  - `4.0 Coordinate Delivery` -> `Status Update` -> `1.0 Manage Orders`
  - `1.0 Manage Orders` -> `Status Update` -> `Customer`
  - `Restaurant` -> `Menu Updates` -> `3.0 Manage Restaurants`
  - `3.0 Manage Restaurants` -> `Updated Menu` -> `D2: Menu`

---

4. Level 0 DFD for Online Banking System

In a DFD, `Account` and `Transaction` are typically **Data Stores**, not external entities. The entities are the actors that interact with the system.

Level 0 DFD (Context Diagram)

- **External Entities:** `Customer`, `Bank Admin`, `External_Financial_Network` (e.g., for inter-bank transfers)
- **Process:** `0. Online Banking System`
- **Data Flows:**
  - `Customer` -> `Login Credentials`, `Transfer Request`, `View Balance`, `Deposit/Withdrawal Info` -> Process 0
  - Process 0 -> `Account Statement`, `Login Confirmation`, `Transaction Status` -> `Customer`
  - `Bank Admin` -> `Manage Customer Account`, `Update Rates` -> Process 0
  - Process 0 -> `Fraud Alerts`, `System Reports` -> `Bank Admin`
  - Process 0 -> `Fund Transfer Request` -> `External_Financial_Network`
  - `External_Financial_Network` -> `Transfer Confirmation` -> Process 0

(Note: The `Account` and `Transaction` data would be shown as Data Stores in the Level 1 DFD, internal to the system).

---

5. DFDs for E-Commerce Order Processing System

Level 0 DFD (Context Diagram)

- **External Entities:** `Customer`, `Warehouse`, `Shipping Carrier`, `Payment Gateway`
- **Process:** `0. E-Commerce Order Processing System`
- **Data Flows:**
  - `Customer` -> `New Order` -> Process 0

    - Process 0 -> `Order Confirmation`, `Shipping Status` -> `Customer`
    - Process 0 -> `Payment Request` -> `Payment Gateway`
    - `Payment Gateway` -> `Payment Confirmation` -> Process 0
    - Process 0 -> `Picking List` -> `Warehouse`
    - `Warehouse` -> `Packed Order Notification` -> Process 0
    - Process 0 -> `Shipping Request` -> `Shipping Carrier`
    - `Shipping Carrier` -> `Tracking ID` -> Process 0

Level 1 DFD

- **External Entities:** `Customer`, `Warehouse`, `Shipping Carrier`, `Payment Gateway`
- **Processes:**
    - `1.0 Validate Order`
    - `2.0 Process Payment`
    - `3.0 Manage Inventory`
    - `4.0 Coordinate Shipping`
- **Data Stores:**
    - `D1: Customer_Orders`
    - `D2: Product_Inventory`
    - `D3: Customer_Data`
- **Data Flows (Sample):**
    - `Customer` -> `New Order` -> `1.0 Validate Order`
    - `1.0 Validate Order` (reads from `D2: Product_Inventory`, `D3: Customer_Data`)
    - `1.0 Validate Order` -> `Order Details` -> `D1: Customer_Orders`
    - `1.0 Validate Order` -> `Payment Request` -> `2.0 Process Payment`
    - `2.0 Process Payment` -> `Payment Request` -> `Payment Gateway`
    - `Payment Gateway` -> `Payment Confirmation` -> `2.0 Process Payment`
    - `2.0 Process Payment` -> `Payment OK` -> `3.0 Manage Inventory`
    - `3.0 Manage Inventory` -> `Stock Update` -> `D2: Product_Inventory`
    - `3.0 Manage Inventory` -> `Picking List` -> `Warehouse`
    - `Warehouse` -> `Packed Order Notification` -> `4.0 Coordinate Shipping`
    - `4.0 Coordinate Shipping` -> `Shipping Request` -> `Shipping Carrier`
    - `Shipping Carrier` -> `Tracking ID` -> `4.0 Coordinate Shipping`
    - `4.0 Coordinate Shipping` -> `Shipping Status` -> `D1: Customer_Orders`
    - `1.0 Validate Order` -> `Order Confirmation` -> `Customer`
    - `4.0 Coordinate Shipping` -> `Shipping Status` -> `Customer`

---

6. DFDs for Inventory Management System

This DFD shows the interaction between stock (as a data store), purchase (as a process), and supplier (as an entity).

Level 0 DFD (Context Diagram)

- **External Entities:** `Supplier`, `Purchase Dept`, `Warehouse`, `Sales Dept`
- **Process:** `0. Inventory Management System`
- **Data Flows:**
    - `Sales Dept` -> `Sales Data` -> Process 0
    - Process 0 -> `Stock Levels` -> `Purchase Dept`
    - `Purchase Dept` -> `Purchase Order` -> Process 0
    - Process 0 -> `PO` -> `Supplier`
    - `Supplier` -> `Shipment/Invoice` -> Process 0
    - Process 0 -> `Stock Update` -> `Warehouse`
    - `Warehouse` -> `Stock Adjustment` -> Process 0

Level 1 DFD

- **External Entities:** `Supplier`, `Purchase Dept`, `Warehouse`, `Sales Dept`
- **Processes:**
    - `1.0 Track Stock` (The "Stock" module)
    - `2.0 Manage Purchases` (The "Purchase" module)
    - `3.0 Manage Suppliers` (The "Supplier" module)
- **Data Stores:**
    - `D1: Stock_Inventory`
    - `D2: Supplier_Data`
    - `D3: Purchase_Orders`
- **Data Flows (Sample):**
    - `Sales Dept` -> `Sales Data` -> `1.0 Track Stock`
    - `Warehouse` -> `Stock Adjustment` -> `1.0 Track Stock`
    - `1.0 Track Stock` -> `Stock Record` -> `D1: Stock_Inventory`
    - `1.0 Track Stock` -> `Re-order Alert` -> `2.0 Manage Purchases`
    - `1.0 Track Stock` -> `Stock Levels` -> `Purchase Dept`
    - `Purchase Dept` -> `Purchase Order` -> `2.0 Manage Purchases`
    - `2.0 Manage Purchases` (reads from `D2: Supplier_Data`)
    - `2.0 Manage Purchases` -> `PO` -> `Supplier`
    - `2.0 Manage Purchases` -> `PO Record` -> `D3: Purchase_Orders`
    - `Supplier` -> `Shipment/Invoice` -> `1.0 Track Stock` (to update inventory)
    - `Supplier` -> `Supplier Info` -> `3.0 Manage Suppliers`
    - `3.0 Manage Suppliers` -> `Supplier Updates` -> `D2: Supplier_Data`

---

7. DFDs for Student Marks Processing

Level 0 DFD (Context Diagram)

- **External Entities:** `Faculty`, `Student`, `Admin`
- **Process:** `0. Marks Management System`
- **Data Flows:**

  - `Faculty` -> `Entered Marks` -> Process 0
  - `Admin` -> `Exam Rules` -> Process 0
  - Process 0 -> `Final Report` -> `Admin`
  - Process 0 -> `Grade Sheet` -> `Student`
  - Process 0 -> `Class Roster` -> `Faculty`

Level 1 DFD

- **External Entities:** `Faculty`, `Student`, `Admin`
- **Processes:**
  - `1.0 Entry Marks`
  - `2.0 Process Results`
  - `3.0 Generate Reports`
- **Data Stores:**
  - `D1: Marks_Data`
  - `D2: Student_Data`
  - `D3: Exam_Config`
- **Data Flows (Sample):**
  - `Faculty` -> `Entered Marks` -> `1.0 Entry Marks`
  - `1.0 Entry Marks` (reads from `D2: Student_Data`)
  - `1.0 Entry Marks` -> `Raw Marks` -> `D1: Marks_Data`
  - `Admin` -> `Exam Rules` -> `2.0 Process Results`
  - `2.0 Process Results` (reads rules from `D3: Exam_Config`)
  - `2.0 Process Results` (reads from `D1: Marks_Data`)
  - `2.0 Process Results` -> `Processed Grades` -> `D1: Marks_Data`
  - `3.0 Generate Reports` (reads from `D1`, `D2`)
  - `3.0 Generate Reports` -> `Grade Sheet` -> `Student`
  - `3.0 Generate Reports` -> `Final Report` -> `Admin`
  - `3.0 Generate Reports` -> `Class Roster` -> `Faculty`

---

8. Project Scheduling with a Gantt Chart

A **Gantt chart** is a project management tool that provides a visual representation of a project's schedule. It's a type of horizontal bar chart where:

- The **vertical (Y) axis** lists the tasks or activities that need to be completed.
- The **horizontal (X) axis** represents time (e.g., days, weeks, or months).
- Each **bar** on the chart represents a single task. The position and length of the bar show its **start date**, **end date**, and **duration**.
- **Dependencies** between tasks can be shown with arrows (e.g., Task B cannot start until Task A is finished).
- **Milestones** (important checkpoints) are often marked with a special symbol, like a diamond.

How it Helps in Project Tracking

Gantt charts are extremely effective for tracking project progress:

1. **At-a-Glance View:** A project manager can instantly see the entire project timeline, what tasks should be in progress, and what's coming next.
2. **Progress Tracking:** A "Today" line can be drawn vertically on the chart. By shading the portion of a task bar that is complete, you can quickly see if a task is **on schedule**, **ahead**, or **delayed**.
3. **Identifying Bottlenecks:** If one task is delayed, the chart clearly shows the "knock-on effect" on all dependent tasks (the critical path), allowing the manager to reallocate resources.
4. **Resource Management:** It helps visualize task overlaps, preventing overallocation of a single resource (e.g., one developer assigned to two full-time tasks at the same time).
5. **Baseline vs. Actual:** A "baseline" (the original plan) can be saved. The actual progress is then plotted against it, making it easy to report on variances to stakeholders.

---

9. Equivalence Partitioning and Boundary Value Analysis

Equivalence Partitioning (EP) is a black-box testing technique where input data is divided into partitions of equivalent data. The system is expected to handle all values within a partition in the same way. Test cases are then designed to pick *one* value from each partition.

Boundary Value Analysis (BVA) is a refinement of EP. It focuses on testing the "boundaries" or "edges" of these partitions, as this is where errors (like off-by-one errors) are most likely to occur.

Example: Password Validation Module

Rule: "Password must be 8-12 characters long."

1. Equivalence Partitioning (EP) We identify three partitions:

- **Partition 1 (Invalid, Too Short):** {1 to 7 characters}
  - *Test Case:* `pass` (4 characters)
- **Partition 2 (Valid):** {8 to 12 characters}
  - *Test Case:* `password10` (10 characters)
- **Partition 3 (Invalid, Too Long):** {13+ characters}
  - *Test Case:* `thispasswordistoolong` (22 characters)

2. Boundary Value Analysis (BVA) We test the boundaries of the *valid* partition (8 and 12):

- **Min - 1 (Invalid):** 7 characters (`passwrd`)
- **Min (Valid):** 8 characters (`password`)
- **Min + 1 (Valid):** 9 characters (`password9`)
- **Max - 1 (Valid):** 11 characters (`password123`)
- **Max (Valid):** 12 characters (`password1234`)
- **Max + 1 (Invalid):** 13 characters (`password12345`)

Using BVA gives you a more robust set of test cases than EP alone.

---

10. Login Module: Pseudo-code and Test Cases

Pseudo-code

```
FUNCTION Login(username, password)
  1.  START
  2.    IF username IS blank OR password IS blank THEN
  3.      RETURN "Username or password cannot be empty"
  4.    END IF

  5.    user = FindUserInDatabase(username)

  6.    IF user IS NOT found THEN
  7.      RETURN "Invalid username"
  8.    END IF

  9.    IF user.password == Hash(password) THEN
  10.     RETURN "Login successful"
  11.   ELSE
  12.     RETURN "Invalid password"
  13.   END IF
  14. END
```

Decision Points

A decision point is any statement that can result in a `TRUE` or `FALSE` outcome, (e.g., `IF` statements).

- **D1:** `IF username IS blank OR password IS blank` (Line 2)
- **D2:** `IF user IS NOT found` (Line 6)
- **D3:** `IF user.password == Hash(password)` (Line 9)

White Box Test Cases (Path Coverage)

To achieve path coverage, we must execute every possible path from start to end. This also ensures decision coverage (every T/F branch of each `IF` is taken).

- **Path 1:** `1 -> 2(T) -> 3 -> 14` (Empty input)
  - **Test Case 1:** `username=""`, `password="123"`
  - **Expected Outcome:** "Username or password cannot be empty"
- **Path 2:** `1 -> 2(F) -> 4 -> 5 -> 6(T) -> 7 -> 14` (User not found)
  - **Test Case 2:** `username="nouser"`, `password="123"`
  - **Expected Outcome:** "Invalid username"
- **Path 3:** `1 -> 2(F) -> 4 -> 5 -> 6(F) -> 8 -> 9(F) -> 11 -> 12 -> 13 -> 14` (Wrong password)
  - **Test Case 3:** `username="validuser"`, `password="wrongpass"`
  - **Expected Outcome:** "Invalid password"
- **Path 4:** `1 -> 2(F) -> 4 -> 5 -> 6(F) -> 8 -> 9(T) -> 10 -> 13 -> 14` (Successful login)
  - **Test Case 4:** `username="validuser"`, `password="correctpass"`
  - **Expected Outcome:** "Login successful"

---

11. ATM Withdrawal: White Box Test Cases

Pseudo-code (Inferred Logic)

```
FUNCTION Withdraw(account_balance, withdrawal_amount, daily_limit_used, daily_limit)
  1.  START
  2.    IF withdrawal_amount > account_balance THEN
  3.      RETURN "Error: Insufficient funds"
  4.    END IF

  5.    remaining_limit = daily_limit - daily_limit_used
  6.    IF withdrawal_amount > remaining_limit THEN
  7.      RETURN "Error: Daily limit exceeded"
  8.    END IF

  9.    new_balance = account_balance - withdrawal_amount
  10.   PRINT "Balance: " + new_balance
  11.   RETURN "Success"
  12. END
```

Test Cases for Branch Coverage

Branch coverage requires that every branch (T/F) of each decision point (`IF` statement) is executed at least once.

- **Decision 1:** `IF withdrawal_amount > account_balance` (Line 2)
- **Decision 2:** `IF withdrawal_amount > remaining_limit` (Line 6)

Test Case	Inputs	Expected Path	D1 (L2)	D2 (L6)	Expected Outcome
TC1	`balance=100`, `amount=200`, `used=0`, `limit=1000`	`1-2(T)-3-12`	**True**	-	"Error: Insufficient funds"
TC2	`balance=1000`, `amount=300`, `used=800`, `limit=1000`	`1-2(F)-4-5-6(T)-7-12`	**False**	**True**	"Error: Daily limit exceeded"
TC3	`balance=1000`, `amount=200`, `used=0`, `limit=1000`	`1-2(F)-4-5-6(F)-8-9-10-11-12`	**False**	**False**	"Balance: 800", "Success"

This set of 3 test cases covers all T/F branches for both decisions.

12. Cart Discount: CFG and Path Test Cases

Control Flow Graph (CFG)

First, let's represent the logic:

1. `START`
2. `discount = 0`
3. `IF cart_value > 10000`
4. `discount = 0.20`
5. `ELSE IF cart_value > 5000`
6. `discount = 0.10`
7. `END IF`
8. `final_price = cart_value * (1 - discount)`

9. `END`

The graph would look like this:

- Node 1/2 -> Node 3 (Decision: >10000)
- Node 3 -> Node 4 (Process: 20%) -> Node 8
- Node 3 -> Node 5 (Decision: >5000)
- Node 5 -> Node 6 (Process: 10%) -> Node 8
- Node 5 -> Node 7 (Implicit join) -> Node 8
- Node 8 -> Node 9 (End)

Independent Paths

The Cyclomatic Complexity is 3 (3 possible paths).

- **Path 1:** 20% Discount (Node 1-2 -> 3(T) -> 4 -> 8 -> 9)
- **Path 2:** 10% Discount (Node 1-2 -> 3(F) -> 5(T) -> 6 -> 8 -> 9)
- **Path 3:** No Discount (Node 1-2 -> 3(F) -> 5(F) -> 7 -> 8 -> 9)

Test Cases for Independent Paths

Test Case	Path	Input (cart_value)	Expected Outcome (discount)
TC1	Path 1	`₹11000`	20% (final price ₹8800)
TC2	Path 2	`₹7000`	10% (final price ₹6300)
TC3	Path 3	`₹4000`	0% (final price ₹4000)

---

13. Registration Form: Condition Coverage Test Cases

Logic

The validation logic is: `IF (name_filled) AND (email_filled) AND (password_filled) AND (email_contains_"@") THEN Valid`

Conditions

- **A:** `name_filled`
- **B:** `email_filled`
- **C:** `password_filled`
- **D:** `email_contains_"@"`

Test Cases for Condition Coverage

Condition coverage requires that each condition (A, B, C, D) is evaluated as `TRUE` at least once and `FALSE` at least once.

Test Case	Inputs (Name, Email, Password)	A (Name)	B (Email)	C (Pass)	D (@)	Expected Outcome
TC1	`("John", "john@ex.com", "123")`	**True**	**True**	**True**	**True**	**Valid**
TC2	`("", "john.com", "")`	**False**	**True**	**False**	**False**	Invalid

This minimal set of two test cases achieves condition coverage:

- **A (name_filled):** True (TC1), False (TC2)
- **B (email_filled):** True (TC1), True (TC2) - *Wait, this is not covered.* We need a B=False.
- **C (password_filled):** True (TC1), False (TC2)
- **D (email_contains_"@"):** True (TC1), False (TC2)

Let's refine the set:

Test Case	Inputs (Name, Email, Password)	A (Name)	B (Email)	C (Pass)	D (@)	Expected Outcome
TC1	`("John", "john@ex.com", "123")`	**True**	**True**	**True**	**True**	**Valid**
TC2	`("", "", "")`	**False**	**False**	**False**	**False**	Invalid

This set (TC1, TC2) covers T/F for all four conditions.

- **A:** True (TC1), False (TC2)
- **B:** True (TC1), False (TC2)
- **C:** True (TC1), False (TC2)
- **D:** True (TC1), False (TC2)

---

14. Library Book Issue: Test Cases

Logic

The logic is: `IF (ID_valid) AND (book_available) AND (member_limit < 3) THEN Issue_Book`

Conditions

- **A:** `member_ID_is_valid`
- **B:** `book_is_available`
- **C:** `member_has_not_exceeded_limit` (books < 3)

Test Cases (Covering all logical conditions/branches)

We need to test all 8 combinations (2^3) for full logical coverage (also known as Multiple Condition Coverage).

TC	A (ID Valid)	B (Book Avail)	C (Limit OK)	Expected Outcome
1	**True**	**True**	**True**	**Issue Book**
2	True	True	**False**	Deny (Limit Exceeded)
3	True	**False**	True	Deny (Book Unavailable)
4	True	**False**	**False**	Deny (Book Unavailable)
5	**False**	True	True	Deny (Invalid Member ID)
6	**False**	True	**False**	Deny (Invalid Member ID)
7S	**False**	**False**	True	Deny (Invalid Member ID)
8	**False**	**False**	**False**	Deny (Invalid Member ID)

(Note: Due to short-circuiting in an `AND` operation, test cases 5, 6, 7, and 8 will all fail at the first condition. Similarly, TC 3 and 4 will fail at the second. This table covers all logical combinations.)

---

15. E-commerce Coupon Validation: Test Cases

Logic

The logic is: `IF (coupon_code_valid) AND (user_has_not_used_it) AND (cart_value >= 1000) THEN Apply_Coupon`

Conditions

- **A:** `Coupon_code_is_valid`
- **B:** `User_has_not_used_it_before`
- **C:** `Minimum_cart_value >= 1000`

Test Cases (Covering all True/False combinations)

We need to test all 8 combinations (2^3).

TC	A (Code Valid)	B (Not Used)	C (Cart >= 1000)	Inputs (Code, Used, Cart)	Expected Outcome
1	**True**	**True**	**True**	`("VALID", No, 1500)`	**Apply Coupon**
2	True	True	**False**	`("VALID", No, 500)`	Deny (Min. cart)
3	True	**False**	True	`("VALID", Yes, 1500)`	Deny (Already used)
4	True	**False**	**False**	`("VALID", Yes, 500)`	Deny (Already used)
5	**False**	True	True	`("INVALID", No, 1500)`	Deny (Invalid code)
6	**False**	True	**False**	`("INVALID", No, 500)`	Deny (Invalid code)
7	**False**	**False**	True	`("INVALID", Yes, 1500)`	Deny (Invalid code)
8	**False**	**False**	**False**	`("INVALID", Yes, 500)`	Deny (Invalid code)

---

16. Risk Management and RMMM Plan

What is Risk Management?

Risk Management is the systematic process of identifying, analyzing, and responding to potential risks that could negatively affect a software project. The goal is not to eliminate all risk (which is impossible) but to minimize the *impact* of "known" risks and to have a plan ready for "unknown" risks. A risk is any uncertain event that, if it occurs, will have a negative effect on the project's scope, schedule, budget, or quality.

Steps in Preparing an RMMM Plan

An **RMMM (Risk Mitigation, Monitoring, and Management)** plan is the document that outlines the complete risk strategy. The steps to create one are:

1. **Risk Identification:**
    - **What it is:** A brainstorming-style process to identify all potential risks.
    - **How:** Conduct team meetings, review past projects (post-mortems), and create checklists. Risks are categorized (e.g., **Technical** (new tech), **Project** (scope creep), **Business** (market changes), **People** (key developer leaves)).
2. **Risk Analysis (or Assessment):**
    - **What it is:** Evaluating each identified risk.
    - **How:** For each risk, assess its **Probability** (0-100% likelihood of occurring) and its **Impact** (1-10 scale of damage if it occurs). Risks are then prioritized, typically by `Risk Exposure = Probability * Impact`.
3. **Risk Mitigation (Planning):**
    - **What it is:** Defining proactive strategies to *reduce* the risk *before* it happens.
    - **How:** This is the first "M." For each high-priority risk, you plan a **Mitigation Strategy**.
        - *Example Risk:* "New database technology is untested by the team."
        - *Mitigation Plan:* "Send two developers for professional training before the project starts."
4. **Risk Monitoring:**
    - **What it is:** Defining how you will *watch* for the risks.
    - **How:** This is the second "M." For each risk, you identify **triggers** or "early warning signs."
        - *Example Risk:* "Scope creep from client."
        - *Monitoring Plan:* "Track the number of change requests per week. If > 3, trigger a meeting."
5. **Risk Management (Contingency):**
    - **What it is:** Defining reactive strategies to *deal* with the risk *after* it has happened.
    - **How:** This is the third "M." This is your "Plan B."
        - *Example Risk:* "Key developer leaves."
        - *Management/Contingency Plan:* "Have a documentation standard to ensure easy handover. Have an external consultant on-call."

The RMMM plan is a living document that records all this information in a table, which is reviewed regularly (e.g., in every weekly sprint meeting).

---

17. Version Control Systems (VCS)

Concept of Version Control Systems (VCS)

A **Version Control System (VCS)**, also known as Source Code Management (SCM), is a software tool that tracks and manages changes to a set of files over time. It is a "database" of changes.

At its core, a VCS allows you to:

- **Maintain History:** Keep a complete record of every change made to a file, including who made it, when, and why.
- **Revert Changes:** "Rewind" a file or the entire project to a previous, known-good state.
- **Enable Collaboration:** Allow multiple developers to work on the same project simultaneously without overwriting each other's work, by helping to **merge** changes together.
- **Branching:** Create isolated "branches" (e.g., a "new-feature" branch or "bug-fix" branch) to work on changes without affecting the main, stable codebase (often called "main" or "master").

Centralized vs. Distributed VCS

The main difference lies in where the "master" copy of the history is stored and how developers interact with it.

Feature	Centralized VCS (CVCS)	Distributed VCS (DVCS)
Core Idea	A single, central server holds the entire project history. Developers "check out" files.	Every developer has a *full copy (clone)* of the entire project history on their local machine.
Workflow	`Check Out -> Edit Files -> Check In / Commit` (to the central server)	`Clone` (once) -> `Edit Files -> Commit` (locally) -> `Push` (to a remote server)
Offline Work	Very limited. You cannot commit, view history, or compare versions without a network connection.	Fully functional. You can commit, create branches, view history, and revert changes all while offline.
Single Point ofFailure	**Yes.** If the central server goes down, all collaboration stops, and new commits are blocked.	**No.** If the main remote server (e.g., GitHub) goes down, developers can continue to work locally and even share changes with each other peer-to-peer.
Branching	Possible, but often slow and "heavy," as it's managed on the server.	Extremely fast and lightweight, as branches are just pointers in your local copy. This encourages their use.
Examples	**Subversion (SVN)**, CVS, Perforce	**Git**, Mercurial (Hg), Bazaar

```
D1: Attendance_Records
D2: Student_Data
D3: Course_Data
Data Flows (Sample):
Faculty -> Attendance Data -> 1.0 Record Attendance
1.0 Record Attendance -> New Record -> D1: Attendance_Records
Admin -> Student/Course Info -> 2.0 Manage System Data
2.0 Manage System Data -> Student Updates -> D2: Student_Data
```

2.0 Manage System Data -> Course Updates -> D3: Course_Data
Faculty -> Report Request -> 3.0 Generate Reports
3.0 Generate Reports (reads from D1, D2, D3) -> Attendance Report -> Faculty
Student -> View Attendance -> 3.0 Generate Reports
3.0 Generate Reports -> Personal Report -> Student
make a dfd digram fir this an dgive me the image

Here is the Level 1 Data Flow Diagram (DFD) for the Student Attendance System based on your description.

give the diagram

Here is the diagram based on that description:

give the block digram

Here is the Level 1 DFD, which functions as a block diagram in this context: