**PRACTICAL-1**

**AIM: Describe various software development models with appropriate diagram.**

Software development models, also known as Software Development Life Cycle (SDLC) models, provide a structured approach to developing software. Each model follows a series of phases to ensure the successful completion of a project.

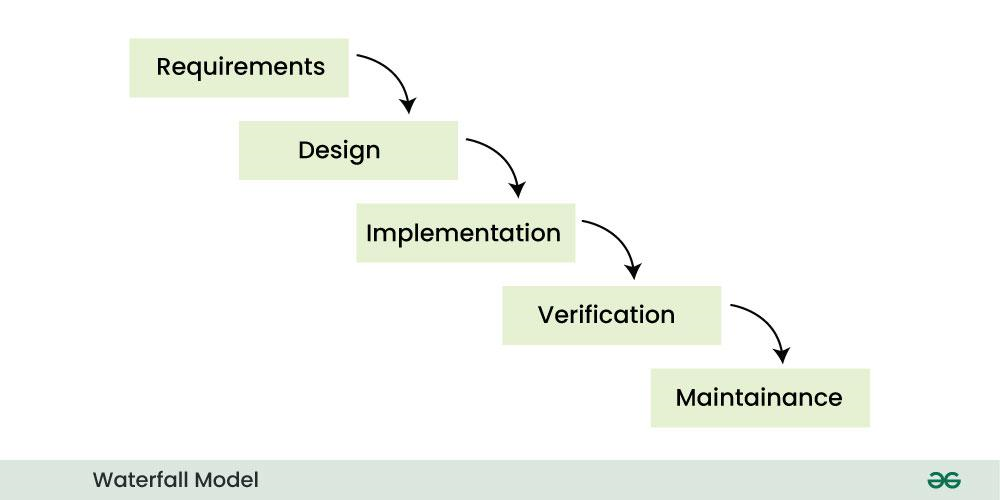
**1. Waterfall Model:**

**The Waterfall Model** is a linear and sequential approach where each phase must be completed before the next one begins. It is one of the most straightforward models.

**Phases:**

1. Requirements
2. Design
3. Implementation
4. Verification (Testing)
5. Maintenance

**WORKING:** Requirements → Design → Implementation → Testing → Maintenance



**Advantages:**

* Simple and easy to understand.
* Works well for small, well-defined projects.

**Disadvantages:**

* Inflexible; changes are difficult to implement once the process has started.
* Not suitable for large or complex projects.

**2. Agile Model:**

The Agile Model is an iterative and incremental approach that focuses on delivering small, functional pieces of software in short iterations (called sprints). It emphasizes customer collaboration and adaptability.

**Phases:**

* Planning
* Design
* Development
* Testing
* Review
* Deployment

**Diagram:** Sprint 1: Plan → Design → Develop → Test → Review → Deploy

Sprint 2: Plan → Design → Develop → Test → Review → Deploy



**Advantages:**

* Highly flexible and adaptable to changes.
* Continuous delivery of working software.

**Disadvantages:**

* Requires active customer involvement.
* May lack documentation compared to other models.

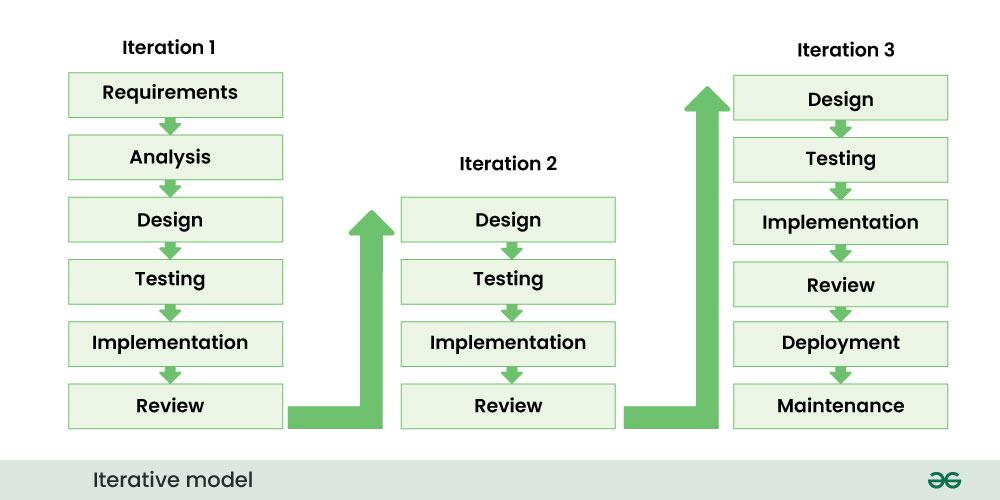
**3. Iterative Model:**

The Iterative Model develops software through repeated cycles (iterations) and incremental improvements. Each iteration produces a working version of the software, which is refined in subsequent iterations.

**Phases:**

* Initial Planning
* Design & Development
* Testing
* Evaluation
* Repeat

**Diagram:** Initial Planning → [Design & Develop → Test → Evaluate] → Repeat



**Advantages:**

* Allows for early delivery of partial solutions.
* Easier to incorporate changes.

**Disadvantages:**

* Requires careful planning and management.
* Can be resource-intensive.

**4. Spiral Model:**

The Spiral Model combines elements of both the Waterfall and Iterative models. It emphasizes risk analysis and is divided into four quadrants: Planning, Risk Analysis, Engineering, and Evaluation.

**Phases:**

1. Planning
2. Risk Analysis
3. Engineering (Design, Development, Testing)
4. Evaluation

**Diagram:** Start → Planning → Risk Analysis → Engineering → Evaluation → Repeat



**Advantages:**

* Focuses on risk management.
* Suitable for large, complex projects.

**Disadvantages:**

* Can be expensive and time-consuming.
* Requires expertise in risk analysis.

**5. V-Model (Verification and Validation Model):**

The V-Model is an extension of the Waterfall Model that emphasizes testing at each stage of development. Each development phase has a corresponding testing phase.

**Phases:**

* Requirements → Acceptance Testing
* Design → System Testing
* Implementation → Integration Testing
* Unit Testing

**Diagram:** Requirements → Design → Implementation

↓ ↓ ↓

Acceptance Testing → System Testing → Integration Testing → Unit Testing



**Advantages:**

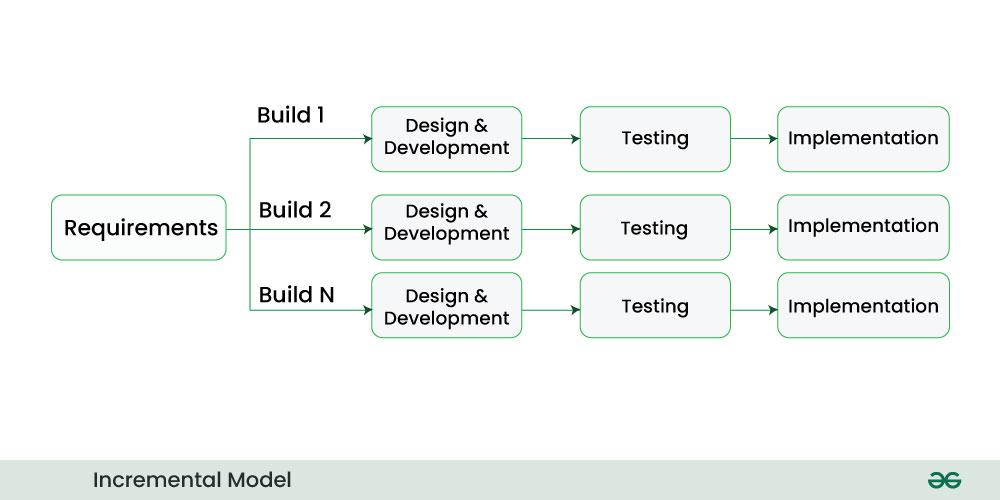
* Ensures high-quality software through rigorous testing.
* Clear and structured approach.

**Disadvantages:**

* Inflexible to changes.
* Not suitable for projects with evolving requirements.

**6. Incremental Model:**

In Incremental Model, the [software development process](https://www.geeksforgeeks.org/software-development-process/) is divided into several increments and the same phases are followed in each increment. In simple language, under this model a complex project is developed in many modules or builds.



**Phases of Incremental Model:**

* **Communication**: In the first phase, we talk face to face with the customer and collect his mandatory requirements. Like what functionalities does the customer want in his software, etc.
* **Planning**: In this phase the requirements are divided into multiple modules and planning is done on their basis.
* **Modeling**: In this phase the design of each module is prepared. After the design is ready, we take a particular module among many modules and save it in DDS (Design Document Specification). Diagrams like ERDs and DFDs are included in this document.
* **Construction**: Here we start construction based on the design of that particular module. That is, the design of the module is implemented in coding. **Deployment**: After the testing of the code is completed, if the module is working properly then it is given to the customer for use. After this, the next module is developed through the same phases and is combined with the previous module. This makes new functionality available to the customer.

**Advantages of Incremental Model:**

* This model is flexible and less expensive to change requirements and scope.
* Project progress can be measured.
* It is easier to test and debug during a short iteration.
* Errors are easy to identify.

**Disadvantages of Incremental Model:**

* Management is a continuous activity that must be handled.
* Before the project can be dismantled and built incrementally,
* The complete requirements of the software should be clear.
* This requires good planning and designing.

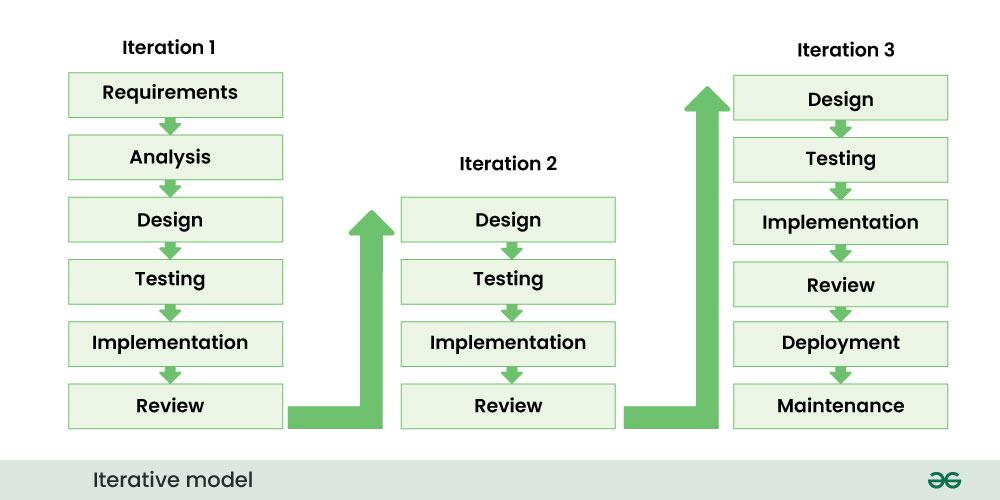
**7. Prototype Model:**

The Prototype Model involves creating an initial version of the software (prototype) to gather feedback from users. The prototype is refined iteratively until the final product is achieved.

**Phases:**

* Requirements Gathering
* Quick Design
* Prototype Development
* User Evaluation
* Refinement

**Diagram:** Requirements → Quick Design → Prototype → User Evaluation → Refinement



**Advantages:**

* Early user feedback improves the final product.
* Reduces the risk of misunderstandings.

**Disadvantages:**

* Can lead to scope creep.
* May require additional time and resources.

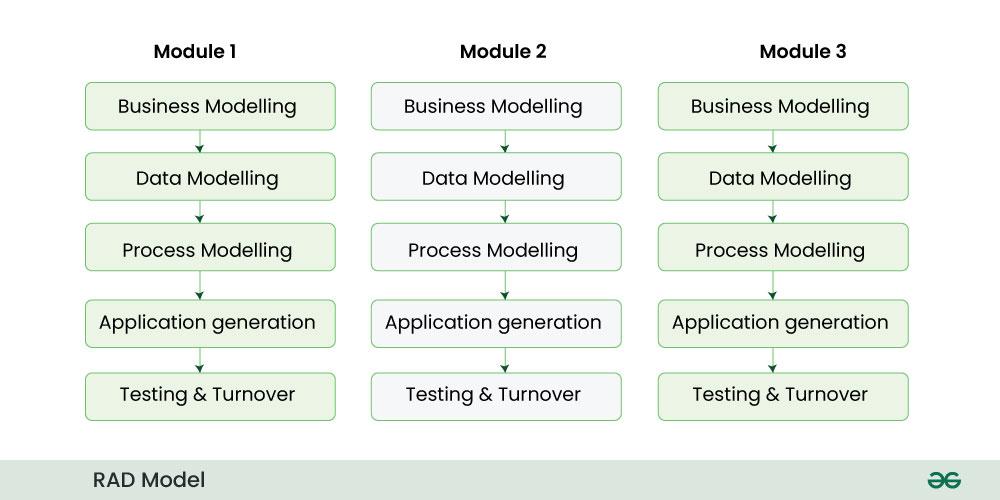
**8. RAD Model (Rapid Application Development):**

The RAD Model focuses on rapid prototyping and iterative development. It emphasizes user feedback and quick delivery of functional components.

**Phases:**

* Requirements Planning
* User Design
* Construction
* Cutover (Deployment)

**Diagram:** Requirements Planning → User Design → Construction → Cutover



**Advantages:**

* Faster delivery of software.
* High user involvement ensures better alignment with requirements.

**Disadvantages:**

* Requires highly skilled developers.
* Not suitable for small projects.

**PRACTICAL-2**

**AIM: Write problem statement to define the project title with bounded scope of the project.**

**Project Title:**

**"Student Attendance Management System"**

**Problem Statement:**

* Manual attendance tracking methods are time-consuming, error-prone, and inefficient, especially in institutions with a large number of students.

The lack of a centralized, automated system often leads to inaccuracies, delays in reporting, and difficulty in generating summary reports.

* To address this, there is a need for a digital attendance management system that automates the process of recording, tracking, and managing student attendance in an efficient, reliable, and user-friendly manner.

The system should be designed following software design fundamentals such as modularity, abstraction, usability, and maintainability.

**Bounded Scope of the Project:**

**✅ In Scope (Features to be Included):**

* Admin panel to manage students, teachers, and class schedules.
* Attendance marking module (manual or with ID input).
* Daily and monthly attendance reports.
* Notifications for low attendance.
* Simple login system for admin and teachers.
* Dashboard with statistics (total students, attendance rate).

**❌ Out of Scope (Excluded Features):**

* Integration with biometric or RFID devices.
* Mobile application support.
* Real-time GPS or location tracking.
* Multi-language support.
* Cloud backup and synchronization.

**PRACTICAL-3**

**AIM: Select relevant process model to define activities and related tasks set for assigned project.**

**Selected Process Model:** Iterative Model

* The Iterative Model is suitable for the Student Attendance Management System **(SAMS)** because:
* It allows incremental development, where core features are built first and refined over iterations.
* Feedback can be incorporated at each stage, improving usability and functionality.
* Well-suited for academic projects with limited time, as it delivers a working prototype early.

**Activities & Task Sets for the Project:**

**1. Requirement Gathering & Analysis**

* **Tasks:**
  + Conduct stakeholder interviews (teachers, admin, students).
  + Define functional requirements (attendance marking, reports, user roles).
  + Identify non-functional requirements (security, usability).
  + Document use cases and system constraints.

**2. System Design**

* **Tasks:**
  + Create UML diagrams (Use Case, Class, Sequence, Activity).
  + Design database schema (tables: Students, Teachers, Attendance, Courses).
  + Plan user interface (wireframes for login, dashboard, attendance entry).
  + Select technology stack

**3. Implementation (Iterative Phases)**

* **Iteration 1 (Core Features):**
  + Develop user authentication (login/logout for admin, teachers).
  + Implement basic attendance entry (manual input by teachers).
  + Store records in a database.
* **Iteration 2 (Enhancements):**
  + Add report generation (daily/monthly attendance summaries).
  + Introduce notifications (email alerts for absent students).
  + Improve UI/UX (responsive design).
* **Iteration 3 (Testing & Refinement):**
  + Add admin controls (add/remove users, course management).
  + Optimize performance and fix bugs.

**4. Testing**

* **Tasks:**
  + Unit testing (individual modules like login, attendance entry).
  + Integration testing (data flow between modules).
  + User Acceptance Testing (UAT) with stakeholders.
  + Bug fixing and retesting.

**5.** **Deployment & Maintenance**

* **Tasks:**
  + Deploy on a local server/web platform (e.g., XAMPP, Heroku).
  + Train end-users (teachers/admin).
  + Monitor system performance and gather feedback.
  + Plan future enhancements (e.g., biometric integration).

**Why Not Other Models?**

* Waterfall: Too rigid; changes are hard to implement mid-project.
* Agile: Requires frequent client feedback (less practical for a class project).
* Spiral: Overly complex for a small-scale system like SAMS.
* The Iterative Model balances flexibility and structure, making it ideal for SAMS in a Fundamentals of Software Design course.

**PRACTICAL-4**

**AIM: Gather application specific requirements- Requirement gathering**

* **Requirement Gathering for Student Attendance Management System (SAMS):**
* To define **application-specific requirements**, we will categorize them into **functional** (what the system should do) and **non-functional** (quality attributes) requirements.

**1. Functional Requirements (Features)**

**User Roles & Authentication**

* **FR1:** The system shall support three user roles: **Admin, Teacher, and Student**.
* **FR2:** Users shall log in using a **username and password**.
* **FR3:** Admin can **add, edit, or deactivate** user accounts.

**Attendance Management**

* **FR4:** Teachers shall mark attendance **daily/weekly** for their assigned classes.
* **FR5:** The system shall allow **manual entry** (Present/Absent/Late) with date and time.
* **FR6:** Students shall view their own attendance records.

**Reporting & Analytics**

* **FR7:** The system shall generate **class-wise, student-wise, and date-wise** attendance reports.
* **FR8:** Reports shall be exportable in **PDF/Excel** format.
* **FR9:** The system shall calculate **attendance percentage** for each student.

**Notifications & Alerts**

* **FR10:** The system shall send **email notifications** to parents if a student is absent.
* **FR11:** Teachers shall receive **reminders** for unattended classes.

**Data Management**

* **FR12:** The system shall store attendance records for at least **one academic year**.
* **FR13:** Admin can **backup and restore** the database.

**2. Non-Functional Requirements (Quality Attributes)**

**Performance**

* **NFR1:** The system shall handle **up to 500 concurrent users** (teachers/admin).
* **NFR2:** Attendance marking shall respond within **2 seconds**.

**Security**

* **NFR3:** All user data shall be **encrypted** (passwords hashed).
* **NFR4:** Only authorized users shall access sensitive functions (e.g., admin controls).

**Usability**

* **NFR5:** The interface shall be **intuitive** for non-technical users (teachers).
* **NFR6:** The system shall be **accessible via web browsers** (Chrome, Firefox).

**Reliability & Scalability**

* **NFR7:** The system shall have **99% uptime** during school hours.
* **NFR8:** The database shall support **future expansion** (e.g., more students/courses).

**3. Constraints**

* **C1:** Must run on **Windows/Linux** servers.
* **C2:** Should use **open-source technologies** (e.g., MySQL, Django).
* **C3:** Must comply with **GDPR/FERPA** (student data privacy).

**PRACTICAL-5**

**AIM: Prepare broad SRS (software requirement software) for the above selected project**

**Software Requirements Specification (SRS)**

* **Document for Student Attendance Management System (SAMS):**

**Version:** 1.4  
**Prepared by:** [ M.J.J.B. INTERNATIONAL ]  
**Date:** [31-04-2025]

**1. Introduction**

**1.1 Purpose**

This document outlines the functional and non-functional requirements for the **Student Attendance Management System (SAMS)**, a web-based application designed to automate attendance tracking in educational institutions.

**1.2 Scope**

* Replaces manual attendance registers with a digital system.
* Provides role-based access (Admin, Teacher, Student).
* Generates real-time attendance reports.
* Sends automated notifications for absenteeism.

**1.3 Definitions**

* **Admin:** Manages users, courses, and system settings.
* **Teacher:** Marks and views attendance.
* **Student:** Views personal attendance records.

**1.4 References**

* IEEE SRS Template
* GDPR/FERPA compliance guidelines

**2. Overall Description**

**2.1 User Needs**

* Teachers need a quick way to mark attendance.
* Admin requires centralized attendance tracking.
* Parents/students need transparency in attendance records.

**2.2 Assumptions**

* Users have basic computer literacy.
* The system will be hosted on a school server/cloud.

**2.3 Constraints**

* Must use open-source technologies (e.g., Django, MySQL).
* Must comply with data protection laws.

**3. Functional Requirements**

**3.1 User Authentication & Roles**

|  |  |
| --- | --- |
| ID | Requirement |
| FR1 | System shall allow login via username & password. |
| FR2 | Admin can create/edit/delete user accounts. |
| FR3 | Teachers can only mark/view attendance for their classes. |

**3.2 Attendance Management**

|  |  |
| --- | --- |
| ID | Requirement |
| FR4 | Teachers can mark students as Present/Absent/Late. |
| FR5 | System shall auto-save attendance with timestamp. |
| FR6 | Students can view their attendance history. |

**3.3 Reporting & Notifications**

|  |  |
| --- | --- |
| ID | Requirement |
| FR7 | Generate PDF/Excel reports (daily/monthly). |
| FR8 | Send email alerts to parents for absent students. |
| FR9 | Admin can export attendance data for audits. |

**4. Non-Functional Requirements**

**4.1 Performance**

|  |  |
| --- | --- |
| ID | Requirement |
| NFR1 | System must handle 500+ concurrent users. |
| NFR2 | Attendance submission response time < 2 sec. |

**4.2 Security**

|  |  |
| --- | --- |
| ID | Requirement |
| NFR3 | Password encryption (SHA-256). |
| NFR4 | Session timeout after 30 mins of inactivity. |

**4.3 Usability**

|  |  |
| --- | --- |
| ID | Requirement |
| NFR5 | Mobile-friendly interface (responsive design). |
| NFR6 | Support for Chrome, Firefox, and Edge. |

**5. System Architecture (High-Level)**

**5.1 Technology Stack**

* **Frontend:** HTML5, CSS3, JavaScript (Bootstrap)
* **Backend:** Python (Django Framework)
* **Database:** MySQL
* **Hosting:** Local server (XAMPP) / Cloud (Heroku)

**6. Future Enhancements**

* Biometric/RFID attendance integration.
* Mobile app for parents/students.
* AI-based absenteeism pattern detection.

**PRACTICAL-6**

**AIM: Develop data designs using DFDs (data flow diagram) and E-R (entity relationship) diagram.**

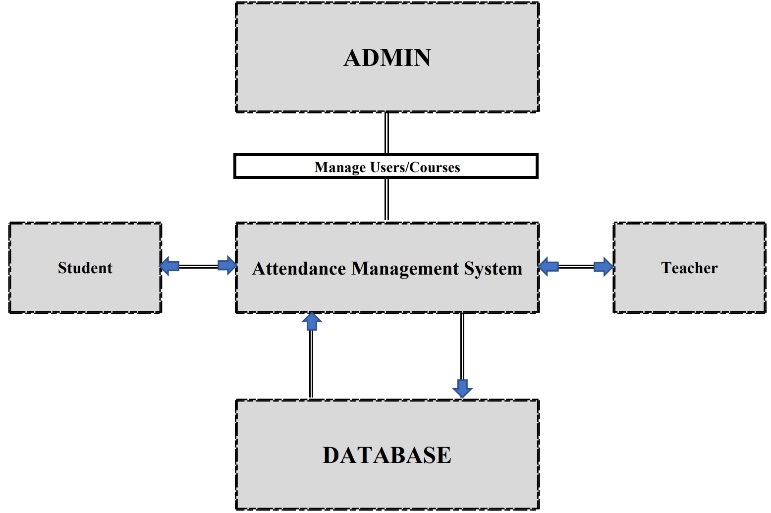
**Data Design for Student Attendance Management System (SAMS):**

**1. Data Flow Diagrams (DFDs):**

* 1. **Level 0 (Context Diagram)**

**Description:**

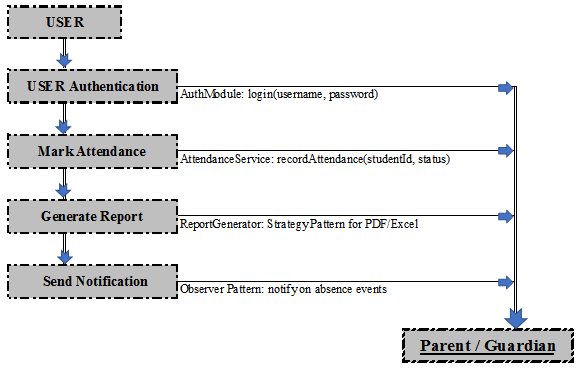
Shows the system’s interactions with external entities (Admin, Teacher, Student, Database).

****

|  |  |
| --- | --- |
| Component | Description |
| Admin | Manages users, courses, and system settings. |
| Teacher | Marks attendance and generates reports. |
| Student | Views attendance records. |
| Database | Stores attendance, user, and course data. |

**1.2 Level 1 DFD – Core Processes**

**Focus:** Modular decomposition of system **behavior** (not tables!).



|  |  |
| --- | --- |
| Process | Software Design Insight |
| User Authentication | Encapsulated in an **AuthModule** (e.g., login(username, password)). |
| Mark Attendance | Part of **AttendanceService** (e.g., recordAttendance(studentId, status)). |
| Generate Report | Separated into **ReportGenerator** (Strategy Pattern for PDF/Excel). |
| Send Notification | **Observer Pattern** (e.g., notify parents on absence events). |

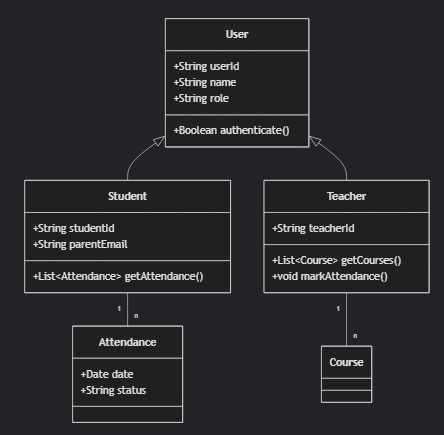
**2. Entity-Relationship (E-R) Diagram – Domain Modeling**

**2.1 Domain Entities (Class-Level View)**

**Focus:** Key domain objects and their **behavioral relationships** (not database tables).

**3. Software Design Principles Applied**

**3.1 Modularity**



* **AuthModule**: Handles login/logout (Single Principle).
* **AttendanceService**: Separates business logic from storage.
* **ReportModule**: Open/Closed Principle (extensible for new report formats).

**3.2 Interfaces (Contract-First Design)**

**` JAVA PROGRAM:-**

interface AttendanceRecorder {

void recordAttendance(String studentId, String status);

}

interface ReportGenerator {

void generateReport(LocalDate startDate, LocalDate endDate);

}

**3.3 Data Flow vs. Control Flow**

* **DFDs** show data movement (e.g., Attendance data → Report).
* **Class Diagram** shows control (e.g., Teacher invokes recordAttendance()).

**PRACTICAL-7**

**AIM: Prepare use-cases and draw use case diagram.**

**1. Use Case Analysis**

**1.1 Actors:**

1. **Admin**
2. **Teacher**
3. **Student**
4. **Parent** (Secondary Actor)

**1.2 Core Use Cases:**

**Admin Use Cases**

|  |  |  |
| --- | --- | --- |
| UC-ID | Use Case Name | Description |
| UC-1 | Manage Users | Admin can add, edit, or deactivate user accounts (students/teachers). |
| UC-2 | Manage Courses | Admin creates/modifies course assignments for teachers. |
| UC-3 | Generate System Reports | Admin exports attendance analytics (e.g., school-wide trends). |

**Teacher Use Cases:**

|  |  |  |
| --- | --- | --- |
| UC-ID | Use Case Name | Description |
| UC-4 | Mark Attendance | Teacher records daily attendance for their class. |
| UC-5 | View Class Attendance | Teacher checks attendance history for their courses. |
| UC-6 | Generate Class Report | Teacher exports attendance reports for a specific class. |

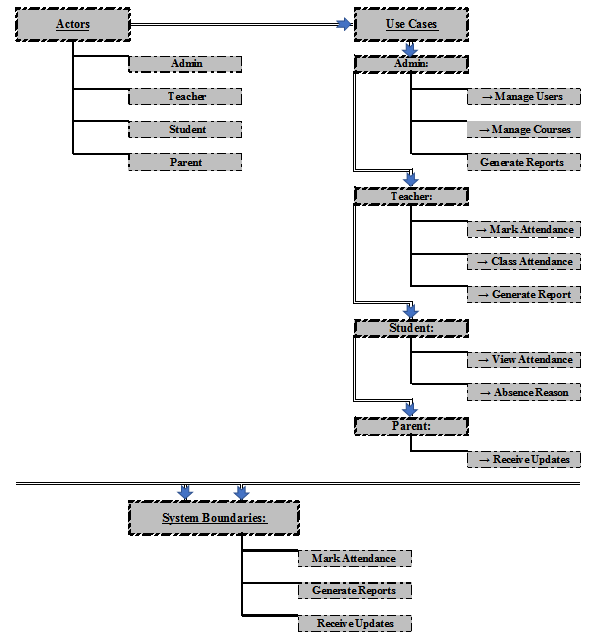
**Student Use Cases:**

|  |  |  |
| --- | --- | --- |
| UC-ID | Use Case Name | Description |
| UC-7 | View Attendance | Student checks their own attendance records. |
| UC-8 | Request Absence Justification | Student submits excuses for absences (optional). |

**Parent Use Cases:**

|  |  |  |
| --- | --- | --- |
| UC-ID | Use Case Name | Description |
| UC-9 | Receive Notifications | Parent gets email/SMS alerts for student absences. |

**2. Use Case Diagram :**

****

**3. Detailed Use Case Specifications:**

**Use Case UC-4: Mark Attendance**

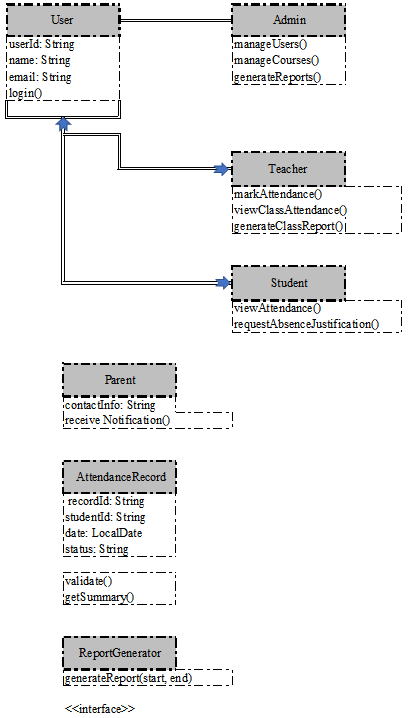
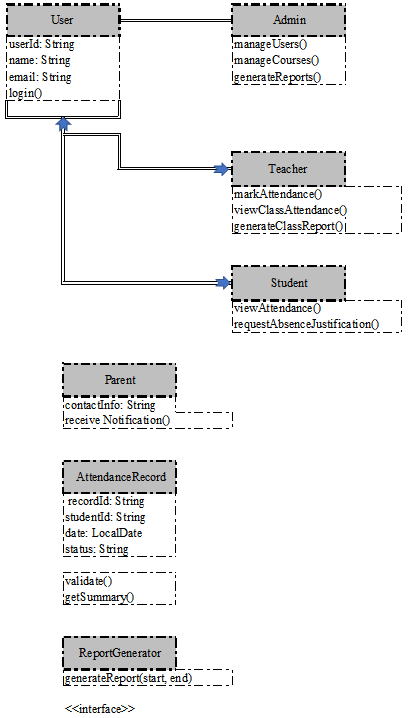
|  |  |
| --- | --- |
| Field | Description |
| Actor | Teacher |
| Preconditions | Teacher is logged in and assigned to a course. |
| Main Flow | 1. Teacher selects a course and date. 2. System displays student list. 3. Teacher marks status (Present/Absent/Late). 4. System saves records. |
| Alternate Flow | If no students are enrolled, system shows error. |
| Postconditions | Attendance is recorded in the database. |

**Use Case UC-9: Receive Notifications**

|  |  |
| --- | --- |
| Field | Description |
| Actor | Parent |
| Trigger | Student is marked absent. |
| Main Flow | 1. System detects absence. 2. Email/SMS is sent to parent. |
| Postconditions | Parent is alerted in real-time. |

**PRACTICAL-8**

**AIM: Develop a class diagram for selected project**

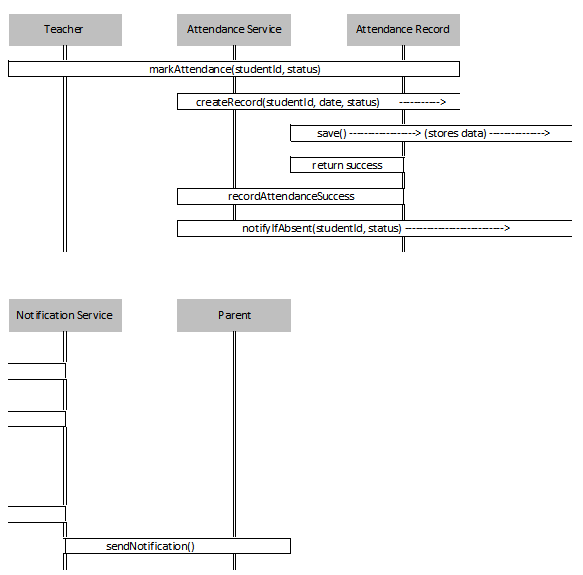
 

* **Key Design Elements:**

1. **Abstraction/Inheritance**:
   * User abstract class extended by Student, Teacher, and Admin.
2. **Encapsulation**:
   * All fields are private (- visibility) with public methods (+).
3. **Interfaces**:
   * ReportGenerator interface with PDFReportGenerator implementation
4. **Associations**:
   * A Teacher teaches multiple Courses.
   * A Course has many Attendance records.
5. **Dependency**:
   * EmailService is triggered by Attendance updates (Observer Pattern).
6. **Separation of Concerns**:
   * Report generation logic decoupled from Teacher class.

**PRACTICAL-9**

**AIM: Develop Sequence diagram for selected project.**

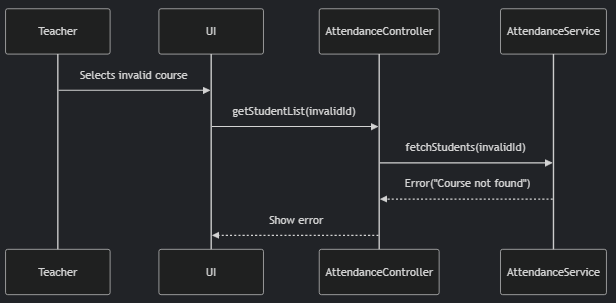


**🧠 Design Concepts Shown:**

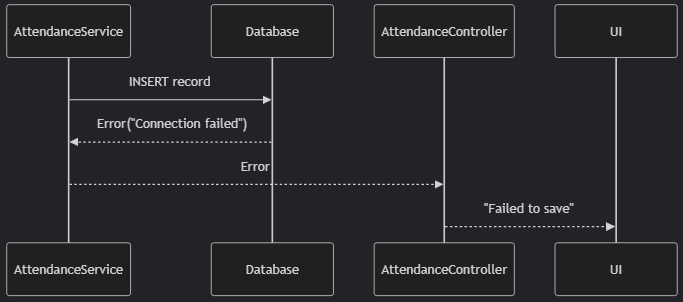
|  |  |
| --- | --- |
| Element | Purpose |
| AttendanceService | Acts as controller — coordinates request. |
| AttendanceRecord | Entity representing attendance data. |
| NotificationService | Applies **Observer Pattern** for notifying guardians. |
| Parent | Receives notification if student is marked absent. |
| Separation of Concerns | Each component handles a **distinct role**. |

**=> Alternate Scenarios:**

**1. Invalid Course/Date**

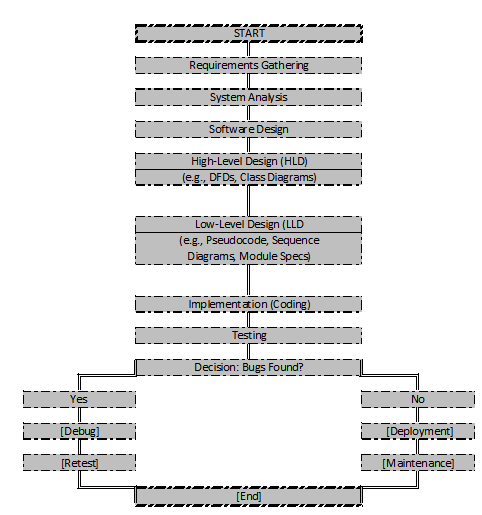


**2. Database Failure**



**PRACTICAL-10**

**AIM: Develop the activity diagram to represent flow from one activity to another for software development.**



**🧠 Key Design Concepts Represented**

|  |  |
| --- | --- |
| Activity | Purpose |
| Requirements Gathering | Understand what the client/user needs. |
| System Analysis | Feasibility, technical assessment, scope clarification. |
| High-Level Design | Architecture, modules, and data flow diagrams (e.g., DFDs). |
| Low-Level Design | Detailed class/method behavior, sequence diagrams, etc. |
| Implementation | Actual code development based on design. |
| Testing & Debugging | Identify and fix functional or logical errors. |
| Deployment & Maintenance | Deliver and maintain the software in production. |

**PRACTICAL-11**

**AIM: Use Gantt charts to track progress of the assigned project. (Use Sprint burn down chart if agile model is selected).**

**📈 1. Gantt Chart for Project Tracking:**

A **Gantt Chart** is a useful tool for visualizing project tasks, timelines, and dependencies. For a typical **software development project** like **SAMS**.

### Example: 📊 Gantt Chart for SAMS Development

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task | Start Date | End Date | Duration | Status |
| 1. Requirements Gathering | 2025-05-01 | 2025-05-07 | 7 days | Not Started |
| 2. System Analysis | 2025-05-08 | 2025-05-10 | 3 days | Not Started |
| 3. High-Level Design | 2025-05-11 | 2025-05-15 | 5 days | Not Started |
| 4. Low-Level Design | 2025-05-16 | 2025-05-20 | 5 days | Not Started |
| 5. Implementation (Coding) | 2025-05-21 | 2025-06-10 | 21 days | Not Started |
| 6. Testing & Debugging | 2025-06-11 | 2025-06-15 | 5 days | Not Started |
| 7. Deployment & Maintenance | 2025-06-16 | 2025-06-20 | 5 days | Not Started |

**📉 2. Sprint Burndown Chart (Agile Method):**

In **Agile**, especially with **Scrum**, progress is tracked using a **Sprint Burndown Chart**. This chart shows how many tasks are left over time in the sprint, giving insights into whether the team is on track to complete the work within the sprint.

**Example:** 📉 **Sprint Burndown Chart**

Let’s assume you're in Sprint 1, which lasts for 2 weeks (10 working days). Below is a representation of how the **burndown** would look, assuming the total story points are 100.

|  |  |
| --- | --- |
| Day | Remaining Tasks (Story Points) |
| Day 1 | 100 |
| Day 2 | 90 |
| Day 3 | 85 |
| Day 4 | 80 |
| Day 5 | 70 |
| Day 6 | 65 |
| Day 7 | 60 |
| Day 8 | 40 |
| Day 9 | 30 |
| Day 10 | 10 |

**PRACTICAL-12**

**AIM: Prepare various test case for selected project.**

## **Test Cases for SAMS:**

### ****1. User Authentication Test Cases****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ****Test Case ID**** | ****Test Case Description**** | ****Test Steps**** | ****Expected Result**** | ****Status**** |
| TC\_001 | Verify login with valid credentials | 1. Enter valid username and password. | User should successfully log in and be redirected to the dashboard. | Pass/Fail |
| TC\_002 | Verify login with invalid credentials | 1. Enter invalid username or password. | An error message "Invalid credentials" should be shown. | Pass/Fail |
| TC\_003 | Verify login with empty credentials | 1. Leave username and password fields empty. | An error message "Please enter username and password" should be shown. | Pass/Fail |
| TC\_004 | Verify logout functionality | 1. Login as valid user.  2. Click on logout button. | User should be logged out and redirected to the login screen. | Pass/Fail |

### ****2. Attendance Management Test Cases****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ****Test Case ID**** | ****Test Case Description**** | ****Test Steps**** | ****Expected Result**** | ****Status**** |
| TC\_005 | Verify marking attendance for a student | 1. Login as teacher.  2. Select a student.  3. Mark attendance (present/absent). | The attendance should be recorded in the system. | Pass/Fail |
| TC\_006 | Verify attendance status change (absent to present) | 1. Mark attendance as absent for a student.  2. Change status to present. | The attendance status should update to "Present". | Pass/Fail |
| TC\_007 | Verify invalid student ID for marking attendance | 1. Enter an invalid student ID while marking attendance. | An error message "Invalid student ID" should appear. | Pass/Fail |
| TC\_008 | Verify attendance recording for multiple students | 1. Select multiple students.  2. Mark attendance for all students. | All selected students' attendance should be recorded. | Pass/Fail |

### ****3. Report Generation Test Cases****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ****Test Case ID**** | ****Test Case Description**** | ****Test Steps**** | ****Expected Result**** | ****Status**** |
| TC\_009 | Verify report generation for class | 1. Login as teacher.  2. Select class.  3. Generate class attendance report (PDF/Excel). | A report should be generated in the selected format (PDF/Excel). | Pass/Fail |
| TC\_010 | Verify report generation for individual student | 1. Login as teacher.  2. Select a student.  3. Generate individual student report. | A report with the selected student's attendance should be generated. | Pass/Fail |

### ****4. Notification System Test Cases****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ****Test Case ID**** | ****Test Case Description**** | ****Test Steps**** | ****Expected Result**** | ****Status**** |
| TC\_012 | Verify parent notification for absent student | 1. Mark a student's attendance as absent.  2. System should send a notification to the parent. | The parent should receive a notification (email/SMS) about the student's absence. | Pass/Fail |
| TC\_013 | Verify notification for attendance update | 1. Change a student's attendance status from absent to present.  2. Notify the parent about the change. | The parent should receive a notification about the updated attendance. | Pass/Fail |
| TC\_014 | Verify no notification for present students | 1. Mark a student's attendance as present.  2. No notification should be sent to the parent. | No notification should be sent if the student is present. | Pass/Fail |

### ****5. General System Test Cases****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ****Test Case ID**** | ****Test Case Description**** | ****Test Steps**** | ****Expected Result**** | ****Status**** |
| TC\_015 | Verify system response to heavy load | 1. Simulate simultaneous logins, attendance markings, and report generations. | The system should handle multiple requests and respond without crashing or slowing down significantly. | Pass/Fail |
| TC\_016 | Verify database updates after attendance mark | 1. Mark attendance for a student.  2. Check the database to ensure the attendance record is updated. | The database should have the correct attendance record for the student. | Pass/Fail |
| TC\_017 | Verify system’s error handling for failed actions | 1. Simulate a failed action (e.g., failed database connection). | The system should show an error message without crashing. | Pass/Fail |

### ****6. Security Test Cases****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ****Test Case ID**** | ****Test Case Description**** | ****Test Steps**** | ****Expected Result**** | ****Status**** |
| TC\_018 | Verify password encryption | 1. Enter a password during login.  2. Check the database to ensure the password is stored encrypted. | The password should be stored in an encrypted format (e.g., bcrypt, AES). | Pass/Fail |
| TC\_019 | Verify unauthorized access | 1. Attempt to access a restricted page without logging in. | The system should redirect to the login page with an error message "Unauthorized access." | Pass/Fail |
| TC\_020 | Verify session expiration | 1. Login and remain inactive for a specified period (e.g., 30 minutes).  2. Try to perform an action. | The session should expire, and the user should be logged out. | Pass/Fail |