PRACTICAL NO. 1

AIM: To study Software Development Life Cycle and identifying the requirements from problem statement

PROBLEM STATEMENT: In many rural and semi-urban areas, Public Health Centres (PHCs) still rely on manual record-keeping for managing patient information, prescriptions, lab tests, and staff activity. This leads to inefficiencies, errors, and difficulties in accessing past medical records, tracking prescriptions, and coordinating lab tests.

To address these issues, there is a need to design and implement a centralized **PHC database system** with an intuitive **user interface (UI)** that can:

- Store and manage patient details,
- Track prescriptions and lab tests,
- Record staff roles and responsibilities (e.g., medical officer, nurses, pharmacist),
- Monitor medicine dispensation and test results,
- Ensure data integrity and easy access for medical staff.

PROBLEM DEFINITION: In this experiment we will learn how to identify functional and non-functional requirements from a given problem statement. Functional and non-functional requirements are the primary components of a Software Requirements Specification.

INTRODUCTION: Requirements identification is the first step of any software development project. Until the requirements of a client have been clearly identified, and verified, no other task (design, coding, testing) could begin. Usually business analysts having domain knowledge on the subject matter discuss with clients and decide what features are to be implemented.

OBJECTIVES:

- Identify ambiguities, inconsistencies and incompleteness from a requirements specification
- Identify and state functional requirements
- Identify and state non-functional requirements

THEORY: Requirements specify how the target system should behave. It specifies what to do, but not how to do. Requirements engineering refers to the process of understanding what a customer expects from the system to be developed, and to document them in a standard and easily readable and understandable format. This documentation will serve as reference for the subsequent design, implementation and verification of the system.

Categorization of Requirements

Based on the target audience or subject matter, requirements can be classified into different types, as stated below:

- **User requirements:** They are written in natural language so that both customers can verify their requirements have been correctly identified
- **System requirements:** They are written involving technical terms and/or specifications, and are meant for the development or testing teams

Requirements can be classified into two groups based on what they describe:

- **Functional requirements (FRs):** These describe the functionality of a system -- how a system should react to a particular set of inputs and what should be the corresponding output.
- **Non-functional requirements (NFRs):** They are not directly related what functionalities are expected from the system. However, NFRs could typically define how the system should behave under certain situations. For example, a NFR could say that the system should work with 128MB RAM. Under such condition, a NFR could be more critical than a FR.

Non-functional requirements could be further classified into different types like:

- **Product requirements:** For example, a specification that the web application should use only plain HTML, and no frames
- **Performance requirements:** For example, the system should remain available 24x7
- **Organizational requirements:** The development process should comply to SEI CMM level 4

Functional Requirements

- Register, update, delete, and search patient records.
- Store prescriptions (medicine name, dosage, duration).
- Track lab tests (test name, date, results).
- Map which staff member performed which task.
- Record medicine dispensed by pharmacist with quantity and date.
- Generate simple reports (e.g., list of patients for a day, pending tests).

Non-Functional Requirements

- The system should be **easy to use** by medical staff.
- It should be **reliable** and **store data securely**.
- It should be **fast enough** to handle daily operations without delays.

SDLC

1. Planning

What are we going to build and why?

- Understand the need of the PHC (store patient records, prescriptions, lab tests, staff details).
- Decide the scope, resources, timeline, and tools (e.g., MySQL for database, simple UI).

2. Defining

Write down clear requirements.

- Create an **SRS (Software Requirements Specification)** document.
- Specify functional requirements (e.g., add patient, store lab test) and non-functional ones (security, performance).

3. Designing

How will the system work and look?

- Create the **ER diagram** for database.
- Design database schema (tables, relationships).
- Plan the **UI layout** (patient page, staff page, etc.).

4. Building (Implementation)

Start coding and creating the system.

- Create the database in MySQL (tables for Patient, Staff, Prescription, Lab_Test, etc.).
- Develop the UI (web app or desktop app) and connect it to the database.

5. Testing

Does everything work correctly?

- Run queries to check if data is inserted, updated, and fetched correctly.
- Test the UI to ensure adding patients, prescriptions, lab tests works as expected.
- Fix any bugs.

6. Deployment

Make the system live for use.

Install the database and UI on PHC computers. Train staff to use the system.