**Practical No.1:** To Study Software Development Life Cycle and Identifying the Requirements from Problem Statements.

### **Aim of the Experiment:**

To study the process of requirements identification by analyzing a problem statement related to a Music Recommendation System. The experiment focuses on distinguishing and documenting functional and non-functional requirements to prepare a foundation for software development.

### **Problem Definition (Identifying the Requirements from Problem Statements):**

In this experiment, the key task is to carefully analyze a given problem statement about a Music Recommendation System and extract all necessary requirements. These requirements should be clear, complete, and free from contradictions to avoid ambiguities during system design. The Music Recommendation System is designed to suggest songs and playlists personalized to users’ music preferences, preferences, and mood or emotional state.

### **Introduction:**

Requirements identification is the **foundation** of the software development process. It ensures that the system to be developed meets user expectations and performs as intended. This step involves gathering information from stakeholders, understanding user goals, and translating these needs into precise requirements.

The process includes:

* Gathering user needs through interviews, surveys, and observation.
* Analyzing the problem domain to identify core functionalities.
* Classifying requirements into functional and non-functional.
* Documenting the requirements clearly to guide the development team.

### **Objectives:**

After completing this experiment, you will be able to:  
(a) Recognize and resolve ambiguities and inconsistencies in requirements.  
(b) Identify core functional requirements of a Music Recommendation System.  
(c) Determine critical non-functional requirements affecting system quality.  
(d) Categorize requirements to improve clarity and communication among stakeholders.  
(e) Understand the importance of well-documented requirements for successful project delivery.

**Theory:**

Requirements identification is the first and most crucial step in software development. It involves understanding what the system should do and documenting these needs clearly to guide design, development, and testing.

#### Characteristics of Requirements

* **Unambiguity:** Requirements must be clear and precise. For example, "The system shall recommend meals within 3 seconds" is better than "recommend meals quickly."
* Consistency: Requirements should not conflict. For instance, recommending only vegan meals but also including meat songs is inconsistent.
* Completeness: All necessary functionalities must be covered, such as allowing users to save favorite songs or generate song queues or downloads.
* **Verifiability:** Requirements should be testable to confirm they are met.

#### Categorization of Requirements

* User Requirements: Simple language understandable by users, e.g., "Users can search songs by ingredients."
* **System Requirements:** Technical details for developers, e.g., "User data must be encrypted."

They are further divided into:

* **Functional Requirements:** Define system actions like registration, recipe search, personalized recommendations.
* **Non-Functional Requirements:** Define system qualities such as performance, security, usability, and availability.

#### Importance of Documentation

Formal documentation of requirements in a Software Requirements Specification (SRS) ensures all stakeholders agree on what the system should do, helps manage scope, and serves as a reference during development and testing.



#### **Requirements:**

##### (1) Characteristics of Requirements

* Unambiguous: Requirements must be expressed in a clear, precise language with no multiple interpretations. For example, “The system should provide personalized playlists” is ambiguous without defining “personalized.”
* **Consistent:** There should be no contradictions among requirements. For example, if one requirement states the system must work offline and another states it must always connect to the server, these conflict.
* Complete: All relevant features and constraints must be included. For instance, specifying maximum daily calorie intake limits is necessary if playlists are personalized for health goals.

##### (2) Categorization of Requirements

* User Requirements: Written in simple, natural language understandable by non-technical stakeholders. Example: “Users can save their favorite songs.”
* **System Requirements:** More technical and detailed, used by developers and testers. Example: “The system shall store user data using AES-256 encryption.”

Requirements can also be divided into:

* **Functional Requirements:** What the system must do.
* **Non-Functional Requirements:** How the system performs under various conditions.

##### (3) Functional Requirements

Some major functional requirements for the Music Recommendation System include:

* **User Registration and Authentication:** New users can register by providing personal data, dietary preferences, and health goals; users log in securely.
* **Profile Management:** Users can update their dietary restrictions and preferences.
* Personalized Food Recommendations: The system uses profile data to suggest meals and songs tailored to user needs.
* Recipe Search and Filtering: Users can search songs by ingredients, genre, song duration, etc.
* Favorites and Ratings: Users can save songs they like and provide ratings for quality feedback.
* Shopping List Generation: The system creates song queues or downloads based on chosen songs or playlists.
* Admin Functions: Admins can manage songs, users, and system data.

### **Preparing Software Requirements Specifications:**

After gathering all requirements, the next step is to formalize them into an SRS document. This document should:

* Clearly list all functional and non-functional requirements.
* Provide identifiers and priorities to each requirement.
* Serve as a contractual agreement between client and developers.
* Help prevent scope creep by detailing agreed-upon features.
* Facilitate testing by providing clear acceptance criteria.

### **Case Study:**

Music Recommendation System

The Music Recommendation System aims to help users find nutritious and delicious songs tailored to their unique health profiles and preferences. The system will be accessible via both web and mobile platforms, featuring user-friendly interfaces and secure data handling.

Key system features include user registration, personalized meal planning, recipe search, favorites management, and shopping list creation. The system also must meet performance standards to support multiple users concurrently, ensure data privacy, and maintain high availability.

### **References:**

1. Ian Sommerville, Software Engineering, 10th Edition, Pearson Education, 2015.
2. IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications.
3. S. K. Singh, Requirements Engineering: Fundamentals, Principles, and Techniques, Wiley India, 2012.
4. Research papers and articles on Music Recommendation Systems from IEEE Xplore and ACM Digital Library.
5. Online tutorials and documentation on requirements engineering best practices.

### **Conclusion:**

This experiment demonstrated the importance of systematically identifying and documenting requirements from a given problem statement. Proper requirements gathering for the Music Recommendation System ensures that all user needs are met and the system behaves as expected. Accurate requirements reduce risks, improve communication among stakeholders, and set a clear path for successful software development.