SNS COLLEGE OF ENGINEERING

Kurumbapalayam (po), Coimbatore – 641 107

Accredited by NAAC-UGC with 'A' Grade

Approved by AICTE & Affiliated to Anna University, Chennai

DEPARTMENT OF INFORMATION TECHNOLOGY

23ITT203- OBJECT ORIENTED SOFTWARE ENGINEERING

UNIT 2

Software Requirement Specification (SRS) in Object-Oriented Software Engineering (OOSE)

The **Software Requirement Specification (SRS)** is a formal document that describes the functional and non-functional requirements of a software system. The purpose of the SRS is to provide a detailed description of the system's intended behavior, functionality, and constraints, so that developers, testers, and stakeholders have a shared understanding of what the software will do. The SRS serves as a blueprint for design, implementation, and testing.

In **Object-Oriented Software Engineering**, the SRS is often aligned with object-oriented principles such as **Use Cases**, **Class Diagrams**, and **Interaction Diagrams** to clearly define the system and its components.

1. What is Software Requirement Specification (SRS)?

An SRS is a comprehensive description of the system's requirements, both functional and non-functional. It serves as the foundation for the development process and includes all the information needed to develop, test, and maintain the system. It also acts as a contract between the development team and stakeholders.

2. Key Components of the SRS Document

An SRS document can be structured in different ways, but it typically includes the following sections:

1. **Introduction**:
    - **Purpose**: Explains the purpose of the document and the system.
    - **Scope**: Outlines the boundaries of the system and what it will and will not do.
    - **Definitions and Acronyms**: Lists any terminology and abbreviations used in the document.
    - **References**: Any documents or resources referenced in the SRS.
    - **Overview**: A brief outline of the SRS structure.

Example:

- **Purpose**: The purpose of this document is to specify the requirements for the Library Management System (LMS) that will enable users to search, borrow, and return books.
- **Scope**: The system will support book searches, issue/return operations, overdue fines, and user management.

2. **System Overview**:
   - A high-level description of the system, its components, and how they interact.
   - **Example**:
     - **Overview of LMS**: The Library Management System is an automated system that tracks books, manages user accounts, and handles book borrowing and returning.
     - **System Components**: The system will consist of the **User Interface (UI)**, **Database**, and **Admin Interface**.
3. **Functional Requirements**:
   - Detailed description of the **functional requirements** — what the system must do. Each requirement typically describes a specific function the system must perform and includes associated use cases or workflows.
   - These requirements are often written as **use cases** or **user stories**.

Example:

- **Functional Requirement 1**: *Search for a Book*
  The system shall allow users to search for books by title, author, or ISBN.
- **Functional Requirement 2**: *Borrow a Book*
  The system shall allow users to borrow books, updating the book's status and user's account accordingly.

4. **Non-Functional Requirements**:
   - Defines the **non-functional** aspects of the system, such as performance, scalability, security, and reliability. These requirements define **how** the system should operate.

Example:

- **Performance**: *The system must respond to user queries within 2 seconds.*
- **Security**: *The system must encrypt user passwords before storing them.*
- **Scalability**: *The system must be able to handle up to 1000 concurrent users.*

5. **Use Case Models**:
   - Use Case Diagrams describe interactions between actors (users, systems) and system functions (use cases).
     Use cases should describe how the system will behave in response to user actions.

Example:

- **Use Case Diagram** for a **Library Management System** might show:
  - Actors: **User**, **Admin**, **Librarian**

        - Use cases: **Search Book**, **Borrow Book**, **Return Book**, **Add Book**
6. **System Features**:
    - A more detailed breakdown of individual system features. For each feature, the document describes its function and how it will interact with other components.

    Example:

    - **Search Book Feature**:
      The system will allow users to search books by title, author, or ISBN. Results will display a list of books that match the search criteria with options to view book details or borrow the book.
7. **External Interfaces**:
    - Describes the interfaces the system will interact with (e.g., external systems, APIs, hardware).

    Example:

    - **Payment Gateway Integration**: *The system must integrate with an external payment gateway to handle overdue fines.*
    - **Barcode Scanner**: *The system must be able to read ISBN barcodes from books for easy identification.*
8. **System Architecture**:
    - A high-level architectural view of the system, often illustrated with diagrams like **Component Diagrams** or **Deployment Diagrams**. This section explains how various system components interact with each other.

    Example:

    - **System Architecture for LMS**: The system has three main components:
        - **Client (UI)**: Interfaces with the user.
        - **Backend (Server)**: Handles business logic, such as book searches and borrowing.
        - **Database**: Stores book and user information.
9. **Data Requirements**:
    - Defines what data the system will use and how it will be stored or processed.
    - This may include **Data Flow Diagrams (DFD)**, **Entity-Relationship Diagrams (ERD)**, and **Database Design**.

    Example:

    - **Book Information**: *The system stores each book's title, author, ISBN, genre, and availability status.*
    - **User Information**: *The system stores each user's name, userID, borrowing history, and fines.*
10. **Assumptions and Constraints**:

- Assumptions are things that are assumed to be true during development (e.g., system environment, external tools).
- Constraints include any limitations that affect the design or development of the system (e.g., hardware, software, legal, budgetary).

Example:

- **Assumption**: *The system will be deployed on Linux servers with PostgreSQL as the database.*
- **Constraint**: *The system must be developed using Java.*

3. Example Structure of an SRS for a Library Management System

1. Introduction

- **Purpose**: This document defines the software requirements for the **Library Management System**.
- **Scope**: The system will manage the lending of books in a library, including book search, borrowing, return, and fine calculation.
- **Definitions**:
  - *Book*: A physical object in the library, represented by a title, author, and ISBN.
  - *User*: A person who interacts with the system to borrow or return books.

2. System Overview

- The system includes three main components:
  - **Client Application (User Interface)**: Allows users to search for books, borrow, return, and view account details.
  - **Server**: Handles book data management, user data, and transactions (borrowing/returning).
  - **Database**: Stores information about books, users, and transactions.

3. Functional Requirements

- **FR1**: *Book Search*
  The system shall allow users to search for books by title, author, or ISBN.
- **FR2**: *Book Borrowing*
  The system shall allow a user to borrow a book if the book is available, updating the inventory accordingly.
- **FR3**: *Fine Management*
  The system shall calculate fines based on overdue books and allow users to pay fines.

4. Non-Functional Requirements

- **Performance**: The system must support up to 500 users concurrently.
- **Security**: All sensitive data (user passwords, payment information) must be encrypted.

5. Use Case Model

- **Actors**: User, Librarian, Admin
- **Use Cases**:
  - Search Book
  - Borrow Book
  - Return Book
  - Generate Fine Report
- **Use Case Diagram**:
  - Users interact with the system to borrow and return books.
  - Admins manage the book inventory and user accounts.

6. External Interfaces

- **Payment Gateway**: The system must integrate with a third-party service for handling payments of overdue fines.
- **Barcode Scanner**: Users can scan barcodes to quickly search for books.

7. Data Requirements

- **Database Tables**:
  - *Users*: userID, name, email, borrowedBooks[].
  - *Books*: bookID, title, author, ISBN, status.
- **DFD**: The flow of information between the User Interface, Server, and Database is illustrated.

4. Benefits of an SRS Document

- **Clear Communication**: An SRS ensures that all stakeholders (clients, developers, testers) are on the same page about what the system will do.
- **Foundation for Design**: The SRS acts as a foundation for system design, helping designers understand what needs to be built.
- **Reference for Testing**: The SRS provides a basis for creating test cases to verify that the system meets its requirements.
- **Reduced Risk of Changes**: A well-defined SRS reduces the likelihood of misunderstandings and scope changes during development.

A Software Requirement Specification (SRS) is essential for developing high-quality software. It clearly defines the functional and non-functional requirements of the system and serves as the contract between stakeholders and developers. The document includes **system features**, **use cases**, **non-functional requirements**, and **external interfaces**, and serves as the basis for system design, development, and testing.