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

Requirement analysis and specification

Requirement Analysis is the process of gathering and analyzing the functional and non-functional needs of stakeholders, users, and systems, to ensure that the software will meet the needs of the end users and other stakeholders.

Key Steps in Requirement Analysis:

- **Identify Stakeholders**:
  Stakeholders include users, customers, developers, and anyone who has an interest in the system's functionality. **Example**: In a **Library Management System**, stakeholders might include the librarian, library users (students or staff), system administrators, and developers.
- **Gather Requirements**:
  Collect information through methods such as:
  Interviews with stakeholders
  Surveys and Questionnaires
  Observation of current systems
  Document analysis (existing software or process documents)

Example: For the Library System, you might conduct interviews with librarians to understand their workflow in managing books, issuing, and tracking loans.

1. **Classify Requirements**:
   - **Functional Requirements**: Describe what the system should do (actions or services).
     - **Example**: "The system should allow a user to search for books by title, author, or ISBN."
   - **Non-Functional Requirements**: Define constraints such as performance, security, reliability, or usability.
     - **Example**: "The system should handle up to 200 concurrent users."
2. **Prioritize Requirements**:
   - Identify essential (must-have) vs. nice-to-have features.
   - Consider constraints like budget, time, and technology limitations.
   - **Example**: **High Priority**: "Allow users to borrow and return books."
   - **Low Priority**: "Allow users to rate books after borrowing."

3. **Define Use Cases**:
    - Use cases describe the system's behavior from the perspective of a user interacting with it. Each use case represents a sequence of actions the system will perform in response to user input.

    Example:

    - **Use Case**: "Borrow Book"
      Actor: User
      Scenario:
        - User logs in.
        - Searches for a book.
        - Selects a book.
        - System checks if it is available.
        - If available, the book is issued to the user.
        - If not, the system informs the user.

Software Requirement Specification (SRS)

The **Software Requirement Specification (SRS)** is a formal document that provides detailed descriptions of the software's expected behavior. The SRS outlines both functional and non-functional requirements, and serves as a reference point throughout the development and testing phases.

Key Components of SRS:

1. **Introduction**:
    - **Purpose**: Explains the purpose of the document and the software system.
    - **Scope**: What the software will do and the boundaries of the system.
    - **Definitions and Acronyms**: Key terms used in the system.

    Example:

    - **Purpose**: "This document specifies the requirements for the Library Management System (LMS)."
    - **Scope**: "The LMS will allow users to search for books, borrow and return books, and manage fines for overdue books."
2. **System Overview**:
    - High-level description of the system’s architecture, components, and their interactions.

    Example:

    - The LMS consists of **User Interface** (for searching and borrowing books), **Database** (for storing user and book data), and **Admin Interface** (for managing library operations).

3. **Functional Requirements**:
    - Describes specific functions or features the system should support, often represented by use cases.

    Example:

    - **Requirement 1**: The system should allow a user to search for books by title, author, or ISBN.
    - **Requirement 2**: The system must allow a user to borrow a book by verifying availability and updating the inventory.
4. **Non-Functional Requirements**:
    - Defines constraints such as performance, security, and usability.

    Example:

    - **Performance**: The system should respond to user queries within 2 seconds.
    - **Security**: The system should store user data securely, encrypting passwords.
    - **Usability**: The system should be easy to navigate, with a simple, user-friendly interface.
5. **System Models**:
    - **Use Case Diagram**: Shows interactions between actors and system use cases.
        - **Example**: A **Use Case Diagram** for the **Library System** shows actors like **User**, **Admin**, and **Librarian**, interacting with use cases like **Borrow Book**, **Return Book**, **Add Book**, and **Search Book**.
    - **Class Diagram**: Shows the structure of classes within the system and their relationships.
        - **Example**:
            - **Class Book** with attributes like title, author, and methods like borrow(), return().
            - **Class User** with attributes like userID, name, and methods like borrowBook(), returnBook().
    - **Data Flow Diagram (DFD)**: Illustrates the flow of data through the system, helping identify processes, data stores, and external entities.
        - **Example**:
            - **Level 0**: The entire system is represented as a single process (Library System), interacting with external entities (User, Library Database).
            - **Level 1**: Decomposes the system into sub-processes such as SearchBook(), BorrowBook(), ReturnBook().
6. **Assumptions and Constraints**:
    - Any assumptions made about the system, such as software or hardware limitations.

    Example:

    - **Assumption**: "Users will have internet access to use the library's online system."

- **Constraint**: "The system must run on both Windows and Linux operating systems."

Object-Oriented Approach to Requirement Specification

In object-oriented software engineering, requirements are often modeled using objects, their relationships, and interactions, which align well with the overall object-oriented design.

Key Concepts:

1. **Use Case Model**:
   - Defines system functionality in terms of interactions between actors (users, systems) and use cases.
   - **Example**:
     - **Use Case**: "Search for a Book"
     - **Actor**: User
     - **Flow**:
       1. User enters a search query (book title, author, etc.).
       2. System searches the database and displays relevant books.
2. **Object Class Model**:
   - Focuses on defining objects and their relationships.
   - **Example**:
     - **Classes in Library System**:
       - Book (attributes: title, author, isbn, status; methods: borrow(), return()).
       - User (attributes: userID, name, borrowedBooks; methods: borrowBook(), returnBook()).

Object Interaction:
Describes how objects interact to achieve system functionality.
Example:
Borrow Book Use Case:
- **Objects involved**: User, Book, LibrarySystem.
- **Interaction**:
  1. User requests to borrow a Book.
  2. LibrarySystem checks availability.
  3. If available, Book is borrowed by the User.

Non-Functional Requirements (NFRs) in O-O systems:
O-O design supports modularization, making it easier to meet **non-functional** requirements like **performance**, **scalability**, and **security**.
Example:
To improve **performance**, implement caching strategies for book search results.
For **security**, ensure that sensitive data, such as user information, is encrypted using secure algorithms.