SOFTWARE ENGINEERING

UNIT – 2

TOPIC – 2

SOFTWARE REQUIREMENTS DOCUMENT

Note: This document is the same one previously uploaded for the IEEE SRS document under Lab Experiment 1.b in Unit 1, but it now includes the additional section on the Characteristics of a Good SRS Document.

1. Introduction to SRS (Software Requirements Specification)

- **What is an SRS Document?**
  - **SRS** stands for **Software Requirements Specification**. It's a detailed document that outlines exactly what a software product should do.
  - **Purpose:**
    Think of it as a blueprint for software development, similar to how an architect's plan guides the construction of a building.
  - **Example:**
    If you were building a house, the blueprint would show the size and layout of each room, where the doors and windows go, and what materials to use. Similarly, an SRS document lays out every detail of how the software should function, what features it should have, and how it should look.

2. Why is an SRS Document Needed?

- **Importance of an SRS Document:**

1. **Clarity and Understanding:** It ensures that everyone involved in the project—developers, designers, and stakeholders—understands what the software is supposed to do.
2. **Guides Development:** The SRS document serves as a guide for the development team, helping them build the software correctly.
3. **Communication Tool:** It acts as a communication bridge between different teams and stakeholders, making sure everyone is on the same page.
4. **Manages Changes:** If there are changes needed during development, the SRS document helps manage those changes without confusion.

5. **Helps in Testing:** The SRS document also helps testers by providing clear criteria on what the software should do, so they can check if it meets all the requirements.

3. Good SRS Document

A Software Requirements Specification (SRS) document is a crucial part of software development. It outlines what the software should do, helping both developers and stakeholders understand the project requirements clearly. A well-written SRS document serves as a foundation for designing, coding, testing, and maintaining software.

Characteristics of a Good SRS Document

i. **Correctness:** The document should accurately describe all the system's requirements. Every feature that the software is supposed to have must be included, and any incorrect or missing information should be avoided.

ii. **Unambiguous:** The language used in the SRS should be clear and straightforward. There should be no confusion or multiple interpretations of any requirement. This helps ensure that everyone, from developers to stakeholders, has the same understanding of what needs to be built.

iii. **Complete:** A good SRS covers all aspects of the software. It should include functional requirements (what the system does) and non-functional requirements (like performance, security, usability, etc.), leaving nothing out.

iv. **Consistent:** The SRS must be free of contradictions. For instance, one section should not specify a requirement that another section contradicts. Consistency across all parts of the document ensures smooth development.

v. **Ranked for importance and stability:** Each requirement should be prioritized based on how critical it is to the project. Some requirements may also change over time, so it's essential to identify which parts are more likely to remain stable.

vi. **Verifiable:** Every requirement in the SRS should be measurable and testable. For example, if the document states that the system should respond in under two seconds, this can be tested and verified.

vii. **Modifiable:** A good SRS document is flexible and can be updated when needed. It should be well-organized so that changes can be made without affecting other parts of the document.

viii. **Traceable:** Each requirement should be traceable back to its source, whether that's a customer need or a regulatory requirement. This helps in ensuring that every feature has a valid reason for being in the project.

A good SRS is one that is accurate, clear, complete, consistent, well-organized, and easy to update. This ensures a smooth development process and helps avoid misunderstandings later on.

4. What is IEEE - SRS?

- **Definition:**
  - **IEEE SRS** refers to the **Software Requirements Specification** as defined by the **Institute of Electrical and Electronics Engineers (IEEE)**.
  - **Purpose:**
    The IEEE SRS is a standardized format for writing an SRS document, ensuring that all important aspects of the software are covered consistently.
  - **Example:**
    Just like a recipe book has a standard format for listing ingredients and steps, the IEEE SRS template ensures that all software projects follow a standard way of documenting requirements. This makes it easier for everyone to understand and follow the document.

Software Requirements Specification (SRS) Template

1. **Introduction**
   - **Purpose**: Describes the product that the SRS covers, including the specific version or part of the system.

     Example: "This document outlines the requirements for Version 2.0 of our customer management software."

   - **Document Conventions**: Lists any standards or formatting used in the document, such as fonts, colours, or symbols that indicate priorities.

     Example: "Critical requirements are highlighted in bold and marked with an asterisk (*)."

   - **Intended Audience**: Identifies who should read the SRS, such as developers, managers, or testers, and suggests reading sequences.

Example: "This document is intended for developers, testers, and project managers to ensure a shared understanding of system requirements."

- **Product Scope**: Provides a brief description of the software, its purpose, and how it aligns with business goals.

  Example: "The software aims to automate inventory management, enhancing operational efficiency."

- **References**: Lists other documents or resources related to the SRS.

  Example: "Refer to the User Interface Style Guide, Version 1.1, for design standards."

2. **Overall Description**

- **Product Perspective**: Explains the software's background and its relationship to other systems or components.

  Example: "This tool is an upgrade to the existing analytics module, designed to integrate with our sales platform."

- **Product Functions**: Summarizes the main features the product will perform, organized in a simple list.

  Example: "Key functions include data import, user authentication, and report generation."

- **User Classes and Characteristics**: Defines different user types and their key characteristics, such as skill level or security access.

  Example: "User classes include Admins, who have full access, and Regular Users, who have limited access."

- **Operating Environment**: Details the hardware and software environments where the software will run.

  Example: "The system will operate on Linux servers with Java 11 and MySQL databases."

- **Design and Implementation Constraints**: Outlines any limitations or specific requirements that affect design choices.

Example: "The system must comply with GDPR regulations and use Python for backend development."

- **User Documentation**: Lists the documentation that will be provided to users, such as manuals or online help guides.

  Example: "A user manual and an interactive help guide will be included."

- **Assumptions and Dependencies**: Identifies assumptions that could impact the project, such as reliance on third-party components.

  Example: "The system assumes the availability of an existing LDAP server for user authentication."

3. **External Interface Requirements**

- **User Interfaces**: Describes how the software will interact with users, including screen layouts and navigation standards.

  Example: "Each screen will have a consistent layout with navigation buttons at the top."

- **Hardware Interfaces**: Details how the software will connect with hardware, such as printers or sensors.

  Example: "The software will communicate with RFID scanners via Bluetooth."

- **Software Interfaces**: Defines connections between the product and other software components, including databases and APIs.

  Example: "The software will interact with REST APIs to fetch real-time data."

- **Communications Interfaces**: Specifies requirements for any communication-related functions, like protocols or security needs.

  Example: "The system will use SSL/TLS for secure data transmission over the network."

4. **System Features**

- **Feature Description and Priority**: Provides brief descriptions of each feature and its importance level (high, medium, low).

Example: "User authentication is a high-priority feature to ensure secure access."

- **Stimulus/Response Sequences**: Lists the expected user actions and corresponding system responses.

  Example: "When the user submits a form, the system will validate the data and display a success message."

- **Functional Requirements**: Details the specific tasks the software must perform for each feature.

  Example: "REQ-1: The system must allow users to reset their passwords via email verification."

5. **Other Nonfunctional Requirements**

- **Performance Requirements**: Specifies how the software should perform under different conditions, such as speed and scalability.

  Example: "The application must process up to 500 transactions per second."

- **Safety Requirements**: Defines measures to prevent harm or data loss.

  Example: "The software must include auto-save functionality to protect user data during unexpected shutdowns."

- **Security Requirements**: Lists the security protocols that must be followed, such as data encryption or user authentication.

  Example: "User data must be encrypted using AES-256 encryption standards."

- **Software Quality Attributes**: Describes attributes like usability, reliability, and maintainability.

  Example: "The system should have a 99.9% uptime to ensure high availability."

- **Business Rules**: Outlines operational rules that influence software behaviour.

  Example: "Only users with the admin role can approve transactions over $5,000."

6. **Annexures**

- **Appendix A: Glossary**: Defines terms, abbreviations, and acronyms used throughout the SRS.

Example: "API: Application Programming Interface."

- **Appendix B: Analysis Models**: Includes relevant analysis diagrams, such as data flow diagrams or class diagrams.

Example: "See Figure B.1 for the system's state-transition diagram."

- **Appendix C: To Be Determined List**: Tracks items that are still pending decisions or additional information.

Example: "TBD-1: Decide on the final database technology."