Lab 1: Introduction to Software Requirement Specification (SRS) document

Lab Objective:

Understand the components of a Software Requirements Specification (SRS) document and create a sample SRS for a simple application, such as a to-do list app.

Material Needed:

- Computer with a word processor (e.g., Microsoft Word, Google Docs)
- Software Tools (ReqView, IBM Rational DOORS)
- Access to internet resources for SRS examples (optional)
- Whiteboard or flip chart (for group discussion)

THEORY

Introduction to SRS:

The **Software Requirements Specification (SRS)** document serves as a formal contract between stakeholders and the development team, outlining the project's functional and non-functional requirements. It defines the **what** of the software, specifying what the system should do without prescribing how it will be done. This ensures that all parties involved have a common understanding of the system's purpose, scope, and limitations.

The primary purposes of an SRS document are:

1. **Clarity of Requirements**: It provides a clear and detailed description of the system's functionality, helping prevent misunderstandings and miscommunications.
2. **Basis for Design and Development**: It acts as a reference point for software engineers, guiding them in designing and developing the system to meet the specified needs.
3. **Contract Between Stakeholders**: It formalizes the agreement between stakeholders (e.g., clients, users) and developers, ensuring that both parties are aligned on the expected outcome.
4. **Validation and Verification**: The document helps in ensuring that the final product meets the defined requirements, aiding both in testing and validation phases.
5. **Change Management**: In case of changes in requirements, the SRS serves as a baseline for comparison, making it easier to track and manage modifications.

Importance of an SRS Document:

1. **Improves Communication**:
    - It acts as a communication bridge between stakeholders (clients, users, developers, testers), ensuring that everyone understands the project scope and objectives. A well-written SRS reduces ambiguities and provides a clear, common language for all parties involved.
2. **Prevents Scope Creep**:

   - By documenting all requirements up front, the SRS minimizes the risk of unplanned feature additions or changes (scope creep), which can lead to project delays and increased costs. It establishes a clear boundary for the project's scope.
3. **Facilitates Project Planning**:
   - It aids in estimating the project's cost, time, and resources by providing detailed functional and non-functional requirements. This helps project managers in developing accurate schedules and budget plans.
4. **Ensures Quality and Consistency**:
   - A well-defined SRS ensures that the final product aligns with the initial requirements, leading to better quality and user satisfaction. It also promotes consistency throughout the development process by providing a single source of truth for all requirements.
5. **Provides a Basis for Testing**:
   - Testers use the SRS as a baseline to create test cases and ensure that the developed system meets the specified requirements. It is vital for validation and verification, ensuring the system behaves as expected.
6. **Supports Maintenance and Future Enhancements**:
   - As software evolves, future developers or maintainers can refer to the SRS to understand the system's original requirements and constraints. This makes it easier to implement changes or add new features while maintaining system integrity.
7. **Reduces Development Costs and Time**:
   - By having clear and detailed requirements, the likelihood of rework and costly modifications during development is reduced. This leads to more efficient use of resources and shorter development cycles.

Components of an SRS (Software Requirements Specification)

An SRS document typically consists of the following key components:

1. Introduction

This section provides a high-level overview of the project, its objectives, and its context. It serves as a foundation for understanding the rest of the document.

Key sub-sections include:

- **Purpose**: Describes the objective of the SRS and its intended audience (e.g., developers, testers, stakeholders).
- **Scope**: Outlines the boundaries of the system, including what the software will and won't cover.
- **Definitions, Acronyms, and Abbreviations**: Lists important terms used in the document for better clarity and consistency.
- **References**: Provides external documents or resources that were referenced in preparing the SRS.
- **Overview**: A brief summary of the structure of the document to help navigate through its sections.

2. Overall Description

This section offers a general description of the system, providing context and insight into the product's environment and constraints.

Key sub-sections include:

- **Product Perspective**: Describes the relationship between the software being developed and other existing systems or interfaces. It highlights system dependencies, assumptions, and interactions.
- **Product Functions**: A summary of the major functionalities of the software. It outlines the system's key features without going into detailed requirements.
- **User Characteristics**: Defines the target audience or users of the system, their expertise levels, and any special requirements they might have.
- **Operating Environment**: Details the environment where the software will operate, such as hardware, software, network, or platform specifications.
- **Assumptions and Dependencies**: Lists assumptions made during the project, such as reliance on specific hardware or software. It also outlines any external factors that could impact development.

3. Specific Requirements

This section contains detailed functional and non-functional requirements that describe the system's behavior, constraints, and performance.

Key sub-sections include:

- **Functional Requirements**: A detailed description of each function the system must perform, typically organized by use cases or features. This outlines inputs, outputs, and expected behaviors for each function.
- **Non-functional Requirements**: Defines constraints on the system, such as:
  - **Performance Requirements**: Speed, response time, and resource utilization.
  - **Security Requirements**: Data protection, authentication, and authorization requirements.
  - **Usability Requirements**: User interface and ease-of-use considerations.
  - **Reliability and Availability**: System uptime, fault tolerance, and failure recovery.
  - **Scalability**: The ability of the system to handle growth.
  - **Portability**: Requirements for adapting the system to different environments.
- **Interface Requirements**: Describes how the system will interact with other systems, hardware, or software, including APIs, file formats, and user interfaces.
- **Data Requirements**: Specifies the data structures, databases, and storage needed for the system to function.

4. Appendices

This section provides additional supporting information that may not be directly part of the main requirements but can be useful for understanding the document or system.

Key sub-sections include:

- **Glossary**: Definitions of terms or jargon used within the SRS.
- **Diagrams and Figures**: Visual representations like data flow diagrams, use case diagrams, or system architecture to aid understanding.
- **References**: Detailed links or citations to external sources, documentation, or standards that are referenced within the SRS.

How an SRS Helps in Communication Between Stakeholders

An SRS is essential in facilitating clear communication between various stakeholders (e.g., clients, developers, testers, and project managers). Here's how it enhances communication:

1. **Establishes a Common Understanding**:
   - The SRS ensures that everyone—technical and non-technical stakeholders—has a shared understanding of the project requirements. By documenting detailed requirements, it eliminates ambiguities, preventing misinterpretations or assumptions about what the software is supposed to do.
2. **Serves as a Point of Reference**:
   - As a central document, the SRS serves as the single source of truth that stakeholders can refer to throughout the project lifecycle. This ensures consistency, especially when there are multiple parties involved, such as developers, testers, and clients, who may need to reference it at different stages.
3. **Bridges the Gap Between Clients and Developers**:
   - Non-technical stakeholders like clients or business managers may not always fully understand the technical details of software development. An SRS document breaks down complex requirements into understandable terms, providing a clear outline of what the software will achieve. This helps align client expectations with the actual capabilities of the development team.
4. **Facilitates Requirements Validation**:
   - Stakeholders, especially the clients, can review the SRS to verify if the documented requirements align with their business needs and goals. This validation process ensures that both the stakeholders and the development team agree on the features and functionalities before the project begins.
5. **Helps Manage Changes**:
   - The SRS provides a baseline for tracking changes in requirements. If a stakeholder requests a new feature or change, the impact of that change can be easily analyzed by referring to the SRS. This structured approach ensures that changes are managed in an organized way without creating confusion.
6. **Supports Project Planning and Estimation**:
   - The detailed requirements in the SRS allow project managers to more accurately estimate time, cost, and resources. Clear requirements also enable developers to better plan their tasks and avoid rework, which can happen due to unclear or evolving requirements.
7. **Provides a Basis for Testing and Validation**:
   - Testers use the SRS to create test cases and validate whether the software meets the specified requirements. It ensures that developers are building the correct features and that the final product aligns with what the stakeholders requested.

Group Activity:

Analyzing an existing SRS

For group activity of analyzing an existing Software Requirements Specification (SRS) document, you can use these sample SRS documents:

1. **ReqView SRS Example**: This is a detailed example of an SRS document that covers key sections like the introduction, overall description, specific requirements, and more. It can be used as a great reference for your group activity. You can access it here

   https://www.reqview.com/papers/ReqView-Example_Software_Requirements_Specification_SRS_Document.pdf

2. **Chalmers University SRS Example**: This sample SRS provides another clear structure that includes sections on system features, external interfaces, and functional and non-functional requirements. It's available for analysis here;

   https://www.cse.chalmers.se/~feldt/courses/reqeng/examples/srs_example_2010_group2.pdf

Individual Activity: Creating a Sample SRS

simple to-do list application

A **simple to-do list application** is a basic software tool designed to help users manage and organize tasks. It is focused on improving productivity and ensuring tasks are completed in a structured manner. Typically, it includes an intuitive interface that allows users to create, manage, and track their tasks with minimal effort.

Key Features:

1. **Task Creation**: The core feature of a to-do list app is the ability to create tasks. Users input task names or descriptions into a text field, which are then added to the list. Some apps might allow for additional information such as due dates, priorities, or task categories.
2. **Task Management**:
    - **Marking as Complete**: Once a task is done, the user can mark it as complete, typically by clicking a checkbox or swiping the task. This provides a sense of progress and accomplishment.
    - **Editing or Deleting Tasks**: Users can edit existing tasks to update descriptions or change deadlines. Completed or irrelevant tasks can also be deleted from the list to keep the interface clean and focused.
3. **Task Categorization**: Some applications provide categories or labels (e.g., work, personal, shopping) to help users organize tasks into different groups, allowing for better prioritization and focus.
4. **Sorting and Filtering**: To make task navigation easier, many to-do list apps allow sorting by different criteria, such as due date, priority, or category. This feature helps users focus on immediate or high-priority tasks.
5. **Deadlines and Reminders**: Advanced to-do list apps often incorporate deadline settings, allowing users to set due dates for tasks. Reminders or notifications can also alert the user when a task is approaching its deadline.
6. **User-friendly Interface**: Simple to-do list apps are designed to be intuitive and easy to use. They feature minimalistic, distraction-free interfaces where the focus remains on managing tasks efficiently.

Benefits of a Simple To-Do List Application:

1. **Improved Productivity**: By organizing tasks in a clear, manageable way, users can better prioritize and complete their work. This helps in time management and ensures important tasks are completed on time.
2. **Reduces Cognitive Load**: Keeping tasks written down in an app means users don’t have to mentally track everything they need to do. This reduces stress and frees up mental space for more important cognitive tasks.
3. **Flexibility**: A simple to-do list can be used for various purposes—work-related tasks, household chores, shopping lists, or even long-term goals. It provides flexibility in adapting to the user's needs.
4. **Accessibility**: Most modern to-do list apps are available across multiple platforms—desktop, mobile, and even web-based—so users can manage their tasks from anywhere. Some apps also sync across devices, ensuring updates are reflected on all platforms.

5. **Progress Tracking**: As users mark tasks as complete, they gain a sense of achievement. This motivates them to continue working through their list, fostering a habit of task management.

Common Use Cases:

- **Personal Task Management**: Users can create daily to-do lists to manage work, errands, or personal projects.
- **Team Collaboration**: Some to-do list apps enable teams to collaborate by sharing lists, assigning tasks, and tracking progress, although this feature is usually found in more advanced task management tools.
- **Study or Homework Planning**: Students can use to-do lists to track assignments, exams, and study schedules.

Popular Simple To-Do List Apps:

1. **Microsoft To Do**: A simple app that allows users to create tasks, set deadlines, and organize lists. It integrates well with Microsoft products like Outlook.
2. **Todoist**: A widely-used task manager that features task prioritization, categorization, and reminders. It also supports collaboration on shared projects.
3. **Google Keep**: A very simple to-do list app that focuses on note-taking and task management, integrating well with other Google services.

Example Scenario:

Imagine a user who wants to manage their daily routine. They could create a simple list with tasks such as "Finish report by 5 PM," "Grocery shopping," and "Call mom." As they complete each task, they check it off, helping them visually track their progress for the day.

Individual Activity Tasks:

1. Each student will create a sample SRS document for a simple to-do list application.
2. Follow the structure outlined in the introduction. Include:
    - **Introduction:**
        - Purpose
        - Scope
        - Definitions, acronyms, and abbreviations
        - References
        - Overview
    - **Overall Description:**
        - Product perspective
        - Product functions
        - User classes and characteristics
        - Constraints
        - Assumptions and dependencies
    - **Specific Requirements:**
        - Functional requirements
        - Non-functional requirements (e.g., performance, security)
    - **Appendices:**
        - Any additional information or diagrams (if needed)
3. **Submission and Feedback**
    - Students will submit their SRS documents for review.
    - Conduct a brief feedback session where selected students can share their SRS and receive constructive feedback from peers and the instructor.

Lab No 1

Sample SRS Template for To-Do List Application

1. Introduction

1.1 Purpose
This SRS document describes the requirements for the To-Do List application, aimed at helping users manage tasks efficiently.

1.2 Scope
The application will allow users to create, edit, delete, and categorize tasks.

1.3 Definitions, Acronyms, and Abbreviations

- SRS: Software Requirements Specification
- UI: User Interface

1.4 References

- [Link to relevant documentation or standards]

1.5 Overview
This document will outline the functionality and constraints of the To-Do List application.

2. Overall Description

2.1 Product Perspective
The application will be a standalone mobile app compatible with iOS and Android.

2.2 Product Functions

- User authentication
- Task creation and management
- Notifications and reminders

2.3 User Classes and Characteristics

- General users: Individuals looking to manage personal tasks.
- Administrators: Manage user accounts and settings.

2.4 Constraints

- Must operate on devices with iOS 12.0 and Android 8.0 or higher.

2.5 Assumptions and Dependencies

- Users have internet access for cloud synchronization.

3. Specific Requirements

3.1 Functional Requirements

- The system shall allow users to create a new task.
- The system shall allow users to edit existing tasks.
- The system shall send notifications for upcoming tasks.

3.2 **Non-Functional Requirements**

- The application shall respond to user actions within 2 seconds.
- The application shall maintain data security through encryption.

4. Appendices

- Additional diagrams (e.g., wireframes) or details can be included here.

Conclusion:

This lab introduces students to the essential components of a Software Requirements Specification. By creating an SRS for a simple application, students will gain practical experience in documenting requirements that can guide software development.

Assessment:

Students will be evaluated based on:

- Completeness and clarity of their SRS document.
- Participation in group discussions.
- Ability to provide constructive feedback to peers.

References for Further Reading:

- IEEE Std 830-1998 - IEEE Recommended Practice for Software Requirements Specifications
- "Software Requirements" by Karl Wiegers and Joy Beatty

Lab No 1

Topics for SRS document:

1. **AI-Powered Healthcare Diagnostics Platform**
   Develop a system using AI/ML algorithms to assist doctors in diagnosing diseases such as cancer, heart diseases, or diabetes using data from medical scans or health records.
2. **Autonomous Electric Vehicle Fleet Management System**
   A system to manage fleets of autonomous electric vehicles, integrating real-time data from the vehicles, traffic conditions, and energy consumption to optimize performance and efficiency.
3. **Blockchain-Based Voting System**
   A secure, transparent, and decentralized voting system leveraging blockchain to enable remote and tamper-proof elections for government or organizational voting.
4. **Virtual Reality Training Platform for Remote Workforce**
   Create a virtual reality-based training system to simulate real-world scenarios for training employees who work remotely, across industries like manufacturing, healthcare, and education.
5. **AI-Powered Content Moderation System for social media**
   An AI-based platform designed to detect and moderate harmful or inappropriate content on social media platforms, ensuring compliance with legal and community guidelines.
6. **Smart City Infrastructure Management System**
   Develop a platform for managing smart city infrastructure like traffic lights, energy grids, public transport, and waste management using IoT (Internet of Things) devices and real-time data analysis.
7. **Quantum Computing Simulation Platform**
   A software simulation platform that allows developers and researchers to simulate quantum computing algorithms before deploying them on actual quantum computers.
8. **Personalized Learning Management System using AI**
   An AI-driven LMS (Learning Management System) that adapts and customizes learning paths for students based on their learning styles, progress, and performance.
9. **Cybersecurity Threat Detection and Response Platform**
   A platform that uses AI and machine learning to identify, predict, and respond to cybersecurity threats in real-time, focusing on protecting critical infrastructure and sensitive data.
10. **Climate Change Data Analytics Platform**
    A platform that integrates real-time environmental data from various sources (e.g., satellite data, IoT sensors) and uses machine learning to predict and analyze the impact of climate change on specific regions.