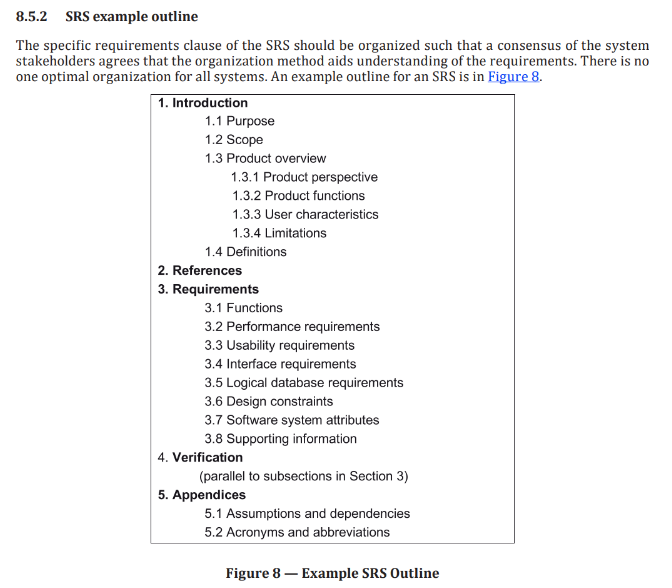
Tutorial folder >> week 4 >> IEEEpg64



Template for SRS:

<https://github.com/jam01/SDD-Template/blob/master/template.md>

diagrams:

context diagram

use case diagram

sequence/activity diagram

EntityRelationship (ER) diagram or class diagram

Software Requirements Specification (SRS) Guidance based on ISO/IEC/IEEE

29148:2018

This document is just an extra guidance/suggestion on how to write the Software Requirements

Specification (SRS) following the outline in Figure 8 and the detailed description in 9.6 of ISO/IEC/IEEE 29148:2018. Each section and subsection includes practical descriptions and examples to help students apply the standard effectively.

**1. Introduction**

**1.1 Purpose**

The main purpose of this document is defining the software requirements for the Student Club Management System with Budget and Venue Integration. This system will be built to streamline and centralize all the operations of student clubs and organizations within a university. This will enable efficient management of club memberships, event planning, budget tracking, and campus venue reservations.

**1.2 Scope**

Here are the student club management system’s functions:

1. Manages administration: The system will handle the admins task for student clubs and organizations.
2. Handles submission and approval: It will manage the process of submitting and getting approval for various club-related items (likely event proposals, funding requests, etc.).
3. Tracks event proposals: The system will keep track of event proposals.
4. Facilitates membership management: It will help manage club memberships
5. Manages club budgets and financial transactions: The system will handle the financial aspects of clubs, including budgets and transactions.
6. Allows reservation for campus venues: Clubs will be able to reserve campus spaces through the system.

**1.3 Product Overview**

The Student Club Management System is a platform that will try its best to support the club operations by integrating with the university financial system and campus space reservation system. It also allows students to manage memberships and events, while admins can track budgets and approve venue bookings.

**1.3.1 Product Perspective (context diagram)**

The Student Club Management System is a web application that integrates with the university’s digital infrastructure. It is built to operate as a broader university ecosystem while connecting with multiple core systems to support the student club activities. The system interfaces with the university authentication system to secure login control for students, club officers, and university staff.

Therefore, it integrates with the financial system to enable budget tracking, funding requests, and financial approvals related to the club operations. Through these integrations, the system also enhances operational efficiency, promotes transparency, and supports coordination among students and administrative departments.

**1.3.2 Product Functions**

View Transaction history

Allow students to check Venue details

Allow students to join or leave student clubs

Enable admins to approve events, budgets, and bookings

Display available campus venues for event planning

Provide a dashboard to track club activities and notifications

**1.3.3 User Characteristics**

The users of the Student Club Management System are students, club officers, faculty advisors, and admins. Students usually have basic computer and technological skills. Club officers probably would have experience in digital tools, as they will be needed to perform tasks such as event creation. Faculty advisors and administrators should have technical proficiency, particularly in reviewing and approving submissions.

**1.3.4 Limitations**

The system is not unlimited to manage student organizations it's recognized by the university. This can only process budget requests through the university’s existing financial system, and venue reservations are usually restricted to spaces listed in the campus reservation system. This system doesn’t support external users or off-campus events. Therefore, performance may be dependent on the availability of the systems, such as authentication, finance, and venue databases.

1.4 Definitions

**Application:** A web-based platform used to manage student club activities, events, budgets, and venue bookings.

**Venue Booking:** The page of reserving an university venue spaces for club events through the reservation system

**Financial Management System : The university’s financial system to manage budgets and transactions, which the application connects to for financial operations.**

**Authentication System :** The university’s login system used to verify user identity and provide access level-based roles such as student, admin, and others.

**2. References**

References list all the sources you’ve cited or consulted while preparing the SRS. These may include standards (like ISO/IEC/IEEE 29148:2018), textbooks, research articles, technical documentation, or software manuals.

Note*:* Use APA 7th edition format for consistency and credibility. This is especially helpful if your SRS will be reviewed in academic settings or by non-technical stakeholders.

Example:

IEEE. (2018). *ISO/IEC/IEEE 29148:2018 Systems and software engineering—Life cycle processes—*

*Requirements engineering*. https://www.iso.org/standard/72089.html

Pohl, K. (2010). *Requirements engineering: Fundamentals, principles, and techniques*. Springer.

**3. Requirements**

**3.1 Functions**

**(Mapped to 9.6.5 Product Functions)**

List the functions of the software and organize them by user (actor). You should provide a use case diagram to represent the system and its functions as use cases. You can place the use case specifications here and relate each use case to the guidelines in 9.6.10. The process for each function must be clearly specified. For each function, you can illustrate by drawing the sequence/activity diagram.

**Example**:

*Student*: Submit, view, and edit research grant applications.

**3.2 Performance Requirements**

The Student Club Management System shall meet the following performance criteria to ensure efficient and reliable operation:

* The system will respond to user interactions (e.g., form submissions, data retrieval, navigation) within **2 seconds** under normal load conditions.
* The system shall support at least **500 concurrent users** without significant degradation in performance.
* The system shall process and display data for dashboard views and reports within **3 seconds** for 95% of requests.
* The authentication and login process shall complete within **2 seconds** for at least 90% of login attempts.
* The system shall be capable of handling a throughput of **100 budget or event requests per hour** during peak usage.
* The application shall maintain performance levels with up to **10,000 registered users** and scalable database storage for long-term growth.
* Background processes (e.g., report generation, data backup) shall not exceed **5 minutes** per operation and shall not impact real-time user functions.

**3.3 Usability Requirements**

* Users should be able to complete key tasks, such as submitting event proposals or budget requests, in no more than **three clicks** from the main dashboard.
* First-time users should be able to learn how to use the system and complete basic functions within **15 minutes**, without formal training.
* The interface will follow a consistent design in terms of layout, icons, and labels, so that users can easily understand and predict how to navigate the system.
* The system will meet common accessibility standards (such as **WCAG 2.1 AA**) so that users with visual, motor, or cognitive impairments can also interact with it effectively.
* A search feature will help users quickly find events, clubs, or documents, with at least **90% accuracy** in search results.
* During user testing, at least **80% of users** should rate the system **4 out of 5 or higher** in overall usability.
* Help tips or brief instructions will be built into key parts of the system to guide users as they go, minimizing confusion or the need for external support.

**3.4 Interface Requirements**

**(Mapped to 9.6.11 External Interfaces and 9.6.4 System Interfaces, User Interfaces, Hardware Interfaces, Software Interfaces, Communications Interfaces)**

Specify all system interfaces, including external systems, user interfaces, hardware, and communications.

* **3.4.1 System Interfaces**: Interfaces with external systems or hardware.

Example: The system will integrate with the university’s authentication system (LDAP).

* **3.4.2 User Interfaces**: Describe the layout and interaction elements, e.g., navigation, buttons, data entry fields.

Example: The web interface will use a responsive layout with a fixed top navigation bar for easy access to key features.

* **3.4.3 Hardware Interfaces**: Specify hardware connections, devices, and communication protocols.

Example: The system shall support USB-connected fingerprint readers for user authentication.

* **3.4.4 Software Interfaces**: Describe interactions with other software or APIs. Example: The system will interact with a third-party cloud service for file storage (e.g., Amazon S3).
* **3.4.5 Communications Interfaces**: Specify protocols, message formats, and network requirements.

Example: The system will use HTTPS for secure communication between client and server.

**3.5 Logical Database Requirements**

**(Mapped to 9.6.15 Logical Database Requirements)**

Describe key data entities, relationships, and constraints. This could include an EntityRelationship (ER) diagram or class diagram.

Example:

The “Application” entity has attributes such as applicationID, title, and submissionDate, and it is related to the “Reviewer” entity.

**3.6 Design Constraints**

**(Mapped to 9.6.16 Design Constraints)**

List any restrictions or limitations imposed on the design of the software, whether they are from external standards, regulations, or technical limitations.

Examples:

The user interface must comply with the university’s branding guidelines.

**3.7 Software System Attributes**

**(Mapped to 9.6.18 Software System Attributes)**

Specify the required attributes of the software product, which affect its quality and performance:

* **Reliability**: The system should be able to recover from a crash within 1 minute.
* **Availability**: The system should be available 99.9% of the time during working hours (Monday through Friday, 8 AM to 6 PM).
* **Security**: The system should use role-based access control (RBAC) and encryption for all sensitive user data.
* **Maintainability**: The system should follow best coding practices and be modular to facilitate updates.
* **Portability**: The software should be able to run on both Linux and Windows servers without additional configuration.

**(Mapped to 9.6.20 Supporting Information)** Any additional supporting information, including:

1. sample input/output formats, descriptions of cost analysis studies or results of questionnaires or any other elicitation techniques;
2. supporting or background information that can help the readers of the SRS;
3. a description of the problems to be solved by the software; and
4. special packaging instructions for the code and the media to meet security, export, initial loading or other requirements.

The SRS should explicitly state whether or not these information items are to be considered part of the requirements.

Example:

Sample input/output formats for key system functions (e.g., CSV format for data export).

x

**4. Verification**

**4.1 Verification Approach**

**(Mapped to 9.6.19 Verification)**

Specify how the system will be verified, including methods, responsible parties, timing, and locations. Example:

* **How**: Functional testing, unit testing, and system integration testing will be used to verify system performance.
* **Who**: Verification will be conducted by the product team and quality assurance (QA) department.
* **When**: Verification will occur at key milestones in the development cycle (e.g., after each sprint).
* **Where**: Verification activities will take place in the QA testing environment.

**4.2 Verification Criteria**

Define the criteria against which the software will be verified. These should align with the functional and quality requirements.

Example:

The response time for a search query should be less than 3 seconds under normal load.

**5. Appendices**

**5.1 Assumptions and Dependencies**

**(Mapped to 9.6.8 Assumptions and Dependencies)**

List any assumptions and dependencies that impact the software development process or its requirements.

Example:

The system depends on the availability of the university's student database for user authentication.

**5.2 Acronyms and Abbreviations**

**(Mapped to 9.6.4 Definitions)**

Include a list of acronyms and abbreviations used in the document. Example:

**SaaS**: Software as a Service

**5.3 Glossary *(Optional Section)***

**Explain the purpose**:

Include a glossary if your project involves many domain-specific or technical terms. This section is especially useful when your system is used by non-technical users, stakeholders, or clients. It complements Section 1.4 (Definitions), but allows for a broader, more explanatory list of terms.