# Jacob Blazina

# CS 325

# Dr. Salter

# April 15, 2021

# Summary for *IEEE Recommended Practice for Software Requirements Specifications*

# by the Software Engineering Standards Committee.

**Preface**

The SRS best practices documentwas originally approved in 1998 and was a collaboration between members of **the Life Cycle Data Harmonization Working Group**, which is a subset of the **IEEE** **Software Engineering Standards Committee**. This document was then reaffirmed in 2009.

**Introduction**

The *IEEE Recommended Practice for Software Requirements Specifications*document sets a guideline for designing software requirements by standardizing certain steps in the software development lifecycle. This standardization is meant to aide in communication between software providers and software customers. Other standardization goals include reducing development effort, providing a baseline for validating compliance, and facilitating the transfer of software products to new endpoints. The document is organized into five main sections with two appendices. Each main section has up to four subsections for granularity.

The document starts by outlining important definitions and cites references; both are referenced throughout the text to prevent ambiguity. After defining important terms, a large part of the document pertains to overall considerations for good SRS documentation, and wraps up by giving an overview of the specific makeup of a Software Requirement Specification (SRS). Appendixes are then included that present SRS templates and guidelines to remain in compliance with the IEEE/EIA 12207.1-1197 standard.

**References and Definitions**

The authors include a list of references to assist with comprehension of the document. It should be noted that 16 of these references are other IEEE standards, while the first reference is an ASTM standard. These references mostly pertain to the Software Development lifecycle, and have publication dates ranging from 1987-1998. It is also noted that the general definitions provided in this document conform to the definitions used in the other IEEE standards. The document outlines definitions for the terms: *contract*, *customer*, *supplier*, and *user*. These terms are used throughout the document and are meant to be consistent with other IEEE specifications.

**Software Requirement Specification Considerations**

The authors start by defining the objective of an SRS. An SRS is generally a document that outlines the specific details of a software solution. An SRS may be developed in a collaborative process between the supplier and the customer, or by either party individually. The recommended approach is collaboration. An SRS is intended to provide clarity for requirements such as functionality, performance, and design. The authors provide a clear boundary regarding what an SRS should and should not encompass. The SRS should only be used for defining requirements; this should not include out of scope information such as specific implementation details and constraints such as quality assurance guidelines. This is to ensure that an SRS provides clear guidance on what not to do, without being overly restrictive on the final solution. This overall guideline leads into the next section regarding the makeup of a good SRS.

The first characteristic of a good SRS is that it correctly encompasses the software requirements at hand. This is followed by attributes related to clarity, validity, and thoroughness. When following these guidelines an SRS will describe a complete and clear picture of the application requirements. The authors use examples such as avoiding the use of ‘to be determined’ to illustrate that an SRS is not considered complete if there is any grey area involved. Any part of the SRS that cannot be validated may negate the purpose of the entire SRS document. The importance of validation is highlighted in a section on internal consistency. To evaluate the quality of an SRS, every requirement must be verifiable by any party with stake in the application. This process of validation is closely tied into the processes of joint preparation and evolution.

The next characteristic of a good SRS is joint preparation, which includes communication between the customer and the supplier regarding the necessary requirements for a product. The authors note that a good SRS cannot be completed by a single party; customers generally are not subject matter experts on software development and suppliers may not have a comprehensive understanding of the client’s operational field. During product development, either party may request changes to the SRS due to circumstances such as unforeseen technical ramifications. The process of making changes to the SRS should be prescribed in a formal change process between the involved parties. To mitigate the repercussions inherent in making changes to the SRS, changes can be front-loaded earlier in the process via prototyping.

Prototyping is a valuable step in facilitating communication between suppliers and customers. Customers are generally more responsive to prototypes compared to evaluating an SRS document. The prototyping process elicits requirements and limitations that may have been overlooked, and the authors suggest that prototype based SRS documents tend to require fewer changes in the long term. The authors note that an SRS should avoid specific design requirements unless absolutely necessary, so a prototype should focus on functionality and avoid setting a design standard.

The authors conclude the SRS considerations section by highlighting certain considerations that should not be included in the SRS. Important considerations include avoiding things such as cost, Software Development Lifecycle, and validation procedures. Other documents should be used to encapsulate project requirements that lie outside of the scope of the SRS.

**Anatomy of a Software Requirements Specification document**

The authors provide a detailed analysis of the individual parts included in a good SRS. An SRS document should include a table of contents, introduction, general description, specific requirements, supporting appendices, and an index. The authors note that there is no strict format requirement, but these sections should be adhered to where possible.

The introduction should outline important information regarding the whole SRS such as definitions, scope, and product description. It should also include information about the SRS in general, such as what the SRS is accomplishing and who the SRS is written for. Scope is needed to identify what problems the SRS is intended to solve. Important references and stakeholders should be included in this section, and specific product information should be included in the proceeding section regarding product description.

A product description should encompass the entire application landscape. Important points here include product placement in industry, general functionality requirements, and user scope. Key points such as external dependencies, industry constraints, and technical assumptions should be outlined. An SRS should avoid getting into implementation details, but any necessary technical details such as hardware and safety considerations may be included. This section should avoid getting into specific product requirements, which should be outlined separately in an easy to reference manner.

Technical details should be organized in a section for specific requirements. These requirements should follow some sort of referential structure, commonly a numerical pattern of sections and sub-sections. These requirements may include specific internal and external interfaces, performance benchmarks, compliance frameworks, and infrastructure constraints. A comprehensive set of requirements allows an SRS to provide a clear path for both parties to validate the scope, timeline, and progress of a project.

**Appendixes**

An SRS may include appendixes that are considered separate from the document itself. Appendixes may contain information such as sample interfaces, supporting information, and/or special instructions. If an SRS contains appendixes, it must clearly mark whether the appendixes should be considered as hard requirements.

**References**

IEEE Computer Society. (1998, October 20). IEEE Recommended Practice for Software Requirements Specifications.

**Discussion Paragraph**

**How does this article apply to the field of computer programming?**

Computer programming is all about solving problems. A customer generally approaches a technology vendor because of a problem or pain point that can only be efficiently solved via computer interactions. Since most customers know little about how software works behind the scenes and developers know little about the specific industry that a customer is in, an SRS is instrumental in providing communication between parties. Unfortunately, in practice, an SRS is primarily for the developer and a lot of the time is used as a CYA (cover your \*accountability) document. Even well-made SRS documents can be interpreted very differently between a customer and a vendor. The IEEE standard is an attempt to clear up any ambiguity here.