**Summary of IEEE Recommended Practice for Software Requirements Specifications by the IEEE Computer Society Software Engineering Standards Committee**

Superior quality software is an integral part of every organization in this generation. This recommendation for software requirements specification (SRS) was provided by the Institute of Electrical and Electronics Engineers (IEEE) Computer Society. SRS works as a bridge between customers and suppliers to produce software that fulfills customers’ requirements. The goal of this SRS is to fill the void of software that is made for the customer instead of software that is produced for the general market and not a specific organization. The SRS helps to decrease software development effort and contributes immensely toward building software that is dedicated to the customer.

This recommended practice is divided into five clauses: scope, references, definitions, background, and essentials. The scope clause helps to narrow down a strong software requirements specification for new software as well as this clause helps to create a product along with the product’s content as well. The references clause records IEEE standards and outline how references are supposed to be in a good SRS. The definitions clause stipulates that this recommendation's definitions of contract, customer, supplier, and user must be consistent with those in IEEE standard 610.12-1990. The contract is a legally binding document between the customer and the supplier. The customer is the one who is paying for the product and the supplier is the one who is producing the product. Users are those who will operate the product in direct or indirect ways, which might not be customers. A good SRS should have a solid background to get output according to the demand of the customer.

The fourth clause discusses eight relevant background points which must be considered when producing the SRS. The first background is the SRS's nature, which explains what the program must accomplish, how it will operate, and its constraints. The second background is the SRS's environment, which describes how the SRS fits into the larger project and considers project components that are relevant to the software requirements. SRS must be precise, understandable, and thorough in addition to being dependable, stable, verifiable, flexible, and traceable which makes up the third background. The fourth background is about a characteristic of a good SRS. An SRS is accurate if—and only if—each requirement within the recommendation sticks to. There is no technology available to SRS that guarantees accuracy. SRS ought to be compared to any relevant higher specification because of this. Both those who develop it and those who use it should have no doubts about the SRS. But because these groups frequently come from divergent backgrounds, they don't always articulate software needs in the same manner. Representations that strengthen the developer's needs specification might work against them by making the user and vice versa less understandable. The SRS is only complete when it contains the following three items: all significant requirements, whether they pertain to functionality, performance, or design constraints; definitions of the software's responses to all realizable classes of input data; and full labels for all the SRS's figures, including definitions of all terms and units of measurement.

Additionally, any SRS that uses to be determined is incomplete. Consistency refers to internal consistency without any conflict in an SRS. An SRS for stability and significance must specify whether demand is stable or significant concerning a software product that is not equally important. Each need in the SRS should be identifiable in some way to make it easier for users and developers to communicate any ambiguous expectations they may have for the product. Only if each criterion included in an SRS can be verified then it is verifiable. If and only if there is a limited, cost-effective procedure that allows a person or machine to verify that the software product satisfies the need, then that requirement is verifiable. Any unclear demand, in general, cannot be verified. An SRS should be modifiable, allowing for easy requirement updates without causing the file's structure to collapse. Additionally, an SRS must be able to be tracked back to all phases of development, which is important when the product is ready for maintenance and operation.

Even after the program is operational, a good SRS should function as a blueprint that is simple to use and put into practice. Joint SRS preparation, the fourth background, calls for supplier and customer collaboration during SRS preparation. Customers typically lack the necessary understanding of the software design and development process to create an effective SRS. Typically, suppliers are not knowledgeable enough about the customer's issue and line of business to provide specifications for a suitable system. The fifth background, the creation of the SRS, emphasizes the significance of precise documentation throughout the development process. The sixth background is the prototyping of the SRS as development assistance. Prototyping is beneficial because it shows unexpected features of the system's behavior and offers immediate feedback. As a result, it generates both new questions and answers, assisting in the resolution of the SRS. The seventh background highlights the fact that specifications offer design guidelines but are not meant to limit design alternatives. There should be a distinct line drawn between projecting a particular design and outlining necessary design restrictions by the SRS writers. The eighth background is made up of project specifications that are integrated into the SRS and that put the software product—rather than the transactions that surround it—as their primary concern. Additionally, a software development plan and a software quality assurance plan, specify project requirements.

The fifth clause addresses four significant components of specifications. The first component is the introduction, which should express the logic behind the software product as well as specify the intended users, including any definitions, acronyms, abbreviations, and references found in the SRS. Also, the first component must include what the remainder of the SRS includes and how it is organized. The second component is the general description, which should contain any interaction between the program and other software systems as well as any limits on software development. Furthermore, the SRS should investigate any restrictions placed on the program by the overall environment in which it will be executed, both existing and foreseeable. These restrictions include any limits imposed by the system, user, hardware, software, and communication interfaces. It also contains any memory, process, or site modifications that impose limits on the software. The software should be logical and accessible, to ensure system compatibility as well as service continuity. Any contingencies or future upgrading requirements should be addressed in the SRS. On top of that, following the second component SRS should define the requirements for any data or initialization sequences that are specific to a given site, mission, or operational mode, as well as specify the site or mission-related features that should be modified to adapt the software to a specific installation. It should also include a backup and recovery feature. The third component is the detailed requirements, which should outline the criteria for creating and assessing the program along with being able to be used to assess the software's performance and level, as well as its reliability and security. A table of contents and an index are frequently included in supporting material, the fourth component, to make it easier to review and maintain the SRS. Additionally, an appendix might be used to offer extra information.

Finally, according to the guidelines specified in IEEE/EIA 12207.1-1997, Annex A and Annex B include templates for arranging software requirements and extra requirements on the SRS.

**Discussion**

A good SRS can guide developers in terms of what to build and how it will simplify and streamline the lives of its users. Until software vendors pay attention to what users are attempting to do and accomplish with the product, they will never receive what they desire. It is significant in the field of computer science since every day, new businesses emerge with their ideas and distinctive characteristics. It enables them to meet customer requests, which is only feasible with the appropriate requirement. This is an excellent summary of what is necessary, what is feasible, and how it may be done in the process of creating quality software. In conclusion, drafting a strong software requirements specification will surely assist and provide a better programming experience than one that does not follow to these guidelines, even though the software design itself could still be theoretical at the planning stage.

**References**

IEEE Computer Society Software Standards Committee. (1998). IEEE Recommended Practice for Software Requirements Specifications (IEEE Std 830-1998) , vol., no., pp.1-40, 20 Oct. 1998, doi: 10.1109/IEEESTD.1998.88286.