A RayComm, Inc. Resource
TECHWR-L

Writing Software Requirements Specifications

by Donn Le Vie, Jr.

Here's the scenario: You're finishing up your latest HTML Help project...no more late nights or weekends...back to a "normal" 50-hour work week. That's when the development team lead strolls into your office and says she just got your manager's okay for you to help the development team "put together the functional requirements specification template for the next major project."

"A what?" you ask with a look of semi-shock. Panic sets in. "What did I do to deserve this? I don't even know where to start! Maybe someone on the TECHWR-L list can help...."

For technical writers who haven't had the experience of designing *software requirements specifications* (SRSs, also known as software functional specifications or system specifications) templates or even writing SRSs, they might assume that being given the opportunity to do so is either a reward or punishment for something they did (or failed to do) on a previous project. Actually, SRSs are ideal projects for technical writers to be involved with because they lay out the foundation for the development of a new product and for the types of user documentation and media that will be required later in the project development life cycle. It also doesn't hurt that you'd be playing a visible role in contributing to the success of the project.

This article will describe what an SRS is and why it's important, discuss how and why technical writers should be involved with them, and discuss the critical elements for writing an SRS. Although this article does not attempt to address all aspects of developing SRSs, it aims to help you determine the scope for such a project, to provide some guidelines for writing SRSs, and to provide additional resources. Hopefully with this information, you'll not be asking, "Why me?" but proclaiming "Why not me?"

What is a Software Requirements Specification?

An SRS is basically an organization's understanding (in writing) of a customer or potential client's system requirements and dependencies *at a particular point in time* (usually) prior to any actual design or development work. It's a two-way insurance policy that assures that both the client and the organization understand the other's requirements from that perspective at a given point in time.

The SRS document itself states in precise and explicit language those functions and capabilities a software system (i.e., a software application, an eCommerce Web site, and so on) must provide, as well as states any required constraints by which the system must abide. The SRS also functions as a blueprint for completing a project with as little cost growth as possible. The SRS is often referred to as the "parent" document because all subsequent project management documents, such as design specifications, statements of work, software architecture specifications, testing and validation plans, and documentation plans, are related to it.

It's important to note that an SRS contains functional and nonfunctional requirements only; it doesn't offer design suggestions, possible solutions to technology or business issues, or any other information other than what the development team understands the customer's system requirements to be.

A well-designed, well-written SRS accomplishes four major goals:

- It provides feedback to the customer. An SRS is the customer's assurance that the development organization understands the issues or problems to be solved and the software behavior necessary to address those problems. Therefore, the SRS should be written in natural language (versus a formal language, explained later in this article), in an unambiguous manner that may also include charts, tables, data flow diagrams, decision tables, and so on.
- It decomposes the problem into component parts. The simple act of writing down software requirements in a well-designed format organizes information, places borders around the problem, solidifies ideas, and helps break down the problem into its component parts in an orderly fashion.
- It serves as an input to the design specification. As mentioned previously, the SRS serves as the parent document to subsequent documents, such as the software design specification and statement of work. Therefore, the SRS must contain sufficient detail in the functional system requirements so that a design solution can be devised.
- It serves as a product validation check. The SRS also serves as the parent document for testing and validation strategies that will be applied to the requirements for verification.

SRSs are typically developed during the first stages of "Requirements Development," which is the initial product development phase in which information is gathered about what requirements are needed--and not. This information-gathering stage can include onsite visits, questionnaires, surveys, interviews, and perhaps a return-on-investment (ROI) analysis or needs analysis of the customer or client's current business environment. The actual specification, then, is written after the requirements have been gathered and analyzed.

Why Should Technical Writers be Involved with Software Requirements Specifications?

Unfortunately, much of the time, systems architects and programmers write SRSs with little (if any) help from the technical communications organization. And when that assistance is provided, it's often limited to an edit of the final draft just prior to going out the door. Having technical writers involved throughout the entire SRS development process can offer several benefits:

- Technical writers are skilled information gatherers, ideal for eliciting and articulating customer requirements. The presence of a technical writer on the requirements-gathering team helps balance the type and amount of information extracted from customers, which can help improve the SRS.
- Technical writers can better assess and plan documentation projects and better meet customer document needs. Working on SRSs provides technical writers with an opportunity for learning about customer needs firsthand--early in the product development process.
- Technical writers know how to determine the questions that are of concern to the user or customer regarding ease of use and usability. Technical writers can then take that knowledge and apply it not only to the specification and documentation development, but also to user interface development, to help ensure the UI (User Interface) models the customer requirements.
- Technical writers, involved early and often in the process, can become an information resource throughout the process, rather than an information gatherer at the end of the process.

In short, a requirements-gathering team consisting solely of programmers, product marketers, systems analysts/architects, and a project manager runs the risk of creating a specification that may be too heavily loaded with technology-focused or marketing-focused issues. The presence of a technical writer on the team helps place at the core of the project those user or customer requirements that provide more of an overall balance to the design of the SRS, product, and documentation.

What Kind of Information Should an SRS Include?

You probably will be a member of the SRS team (if not, ask to be), which means SRS development will be a collaborative effort for a particular project. In these cases, your company will have developed SRSs before, so you should have examples (and, likely, the company's SRS template) to use. But, let's assume you'll be starting from scratch. Several standards organizations (including the IEEE) have identified nine topics that must be addressed when designing and writing an SRS:

1. Interfaces
2. Functional Capabilities
3. Performance Levels
4. Data Structures/Elements
5. Safety
6. Reliability
7. Security/Privacy
8. Quality
9. Constraints and Limitations

But, how do these general topics translate into an SRS document? What, specifically, does an SRS document include? How is it structured? And how do you get started? An SRS document typically includes four ingredients, as discussed in the following sections:

1. A template
2. A method for identifying requirements and linking sources
3. Business operation rules
4. A traceability matrix

Begin with an SRS Template

The first and biggest step to writing an SRS is to select an existing template that you can fine tune for your organizational needs (if you don't have one already). There's not a "standard specification template" for all projects in all industries because the individual requirements that populate an SRS are unique not only from company to company, but also from project to project within any one company. The key is to select an existing template or specification to begin with, and then adapt it to meet your needs.

In recommending using existing templates, I'm not advocating simply copying a template from available resources and using them as your own; instead, I'm suggesting that you use available templates as guides for developing your own. It would be almost impossible to find a specification or specification template that meets your particular project requirements exactly. But using other templates as guides is how it's recommended in the literature on specification development. Look at what someone else has done, and modify it to fit your project requirements. (See the sidebar called "Resources for Model Templates" at the end of this article for resources that provide sample templates and related information.)

Table 1 shows what a basic SRS outline might look like. This example is an adaptation and extension of the IEEE Standard 830-1998:

Table 1 A sample of a basic SRS outline

1. Introduction 1.1 Purpose 1.2 Document conventions

1.3 Intended audience
1.4 Additional information
1.5 Contact information/SRS team members
1.6 References

2. Overall Description
2.1 Product perspective
2.2 Product functions
2.3 User classes and characteristics
2.4 Operating environment
2.5 User environment
2.6 Design/implementation constraints
2.7 Assumptions and dependencies

3. External Interface Requirements
3.1 User interfaces
3.2 Hardware interfaces
3.3 Software interfaces
3.4 Communication protocols and interfaces

4. System Features
4.1 System feature A
4.1.1 Description and priority
4.1.2 Action/result
4.1.3 Functional requirements
4.2 System feature B

5. Other Nonfunctional Requirements
5.1 Performance requirements
5.2 Safety requirements
5.3 Security requirements
5.4 Software quality attributes
5.5 Project documentation
5.6 User documentation

6. Other Requirements
Appendix A: Terminology/Glossary/Definitions list
Appendix B: To be determined

Table 2 shows a more detailed SRS outline, showing the structure of an SRS template as found on Ken Rigby's informative Web site at http://neon.airtime.co.uk/users/wysywig/srs_mt.htm. Reprinted with permission.

Table 2 A sample of a more detailed SRS outline

1. Scope	1.1 Identification. *Identify the system and the software to which this document applies, including, as applicable, identification number(s), title(s), abbreviation(s), version number(s), and*

	release number(s). 1.2 System overview. *State the purpose of the system or subsystem to which this document applies.* 1.3 Document overview. *Summarize the purpose and contents of this document.* This document comprises six sections: • Scope • Referenced documents • Requirements • Qualification provisions • Requirements traceability • Notes Describe any security or privacy considerations associated with its use.
2. Referenced Documents	2.1 Project documents. *Identify the project management system documents here.* 2.2 Other documents. 2.3 Precedence. 2.4 Source of documents.
3. Requirements	This section shall be divided into paragraphs to specify the Computer Software Configuration Item (CSCI) requirements, that is, those characteristics of the CSCI that are conditions for its acceptance. CSCI requirements are software requirements generated to satisfy the system requirements allocated to this CSCI. Each requirement shall be assigned a project-unique identifier to support testing and traceability and shall be stated in such a way that an objective test can be defined for it. 3.1 Required states and modes. 3.2 CSCI capability requirements.

	3.3 CSCI external interface requirements. 3.4 CSCI internal interface requirements. 3.5 CSCI internal data requirements. 3.6 Adaptation requirements. 3.7 Safety requirements. 3.8 Security and privacy requirements. 3.9 CSCI environment requirements. 3.10 Computer resource requirements. 3.11 Software quality factors. 3.12 Design and implementation constraints. 3.13 Personnel requirements. 3.14 Training-related requirements. 3.15 Logistics-related requirements. 3.16 Other requirements. 3.17 Packaging requirements. 3.18 Precedence and criticality requirements.
4. Qualification Provisions	To be determined.
5. Requirements Traceability	To be determined.
6. Notes	This section contains information of a general or explanatory nature that may be helpful, but is not mandatory. 6.1 Intended use. This Software Requirements specification shall ... 6.2 Definitions used in this document. *Insert here an alphabetic list of definitions and their source if different from the declared sources specified in the "Documentation*

	standard." 6.3 Abbreviations used in this document. *Insert here an alphabetic list of the abbreviations and acronyms if not identified in the declared sources specified in the "Documentation Standard."* 6.4 Changes from previous issue. *Will be "not applicable" for the initial issue.* Revisions shall identify the method used to identify changes from the previous issue.

Identify and Link Requirements with Sources

As noted earlier, the SRS serves to define the functional and nonfunctional requirements of the product. Functional requirements each have an origin from which they came, be it a *use case* (which is used in system analysis to identify, clarify, and organize system requirements, and consists of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal), government regulation, industry standard, or a business requirement. In developing an SRS, you need to identify these origins and link them to their corresponding requirements. Such a practice not only justifies the requirement, but it also helps assure project stakeholders that frivolous or spurious requirements are kept out of the specification.

To link requirements with their sources, each requirement included in the SRS should be labeled with a unique identifier that can remain valid over time as requirements are added, deleted, or changed. Such a labeling system helps maintain change-record integrity while also serving as an identification system for gathering metrics. You can begin a separate requirements identification list that ties a requirement identification (ID) number with a description of the requirement. Eventually, that requirement ID and description become part of the SRS itself and then part of the Requirements Traceability Matrix, discussed in subsequent paragraphs. Table 3 illustrates how these SRS ingredients work together.

Table 3 This sample table identifies requirements and links them to their sources

ID No.	Paragraph No.	Requirement	Business Rule Source
17	5.1.4.1	Understand/communicate using SMTP protocol	IEEE STD xx-xxxx
18	5.1.4.1	Understand/communicate using POP protocol	IEEE STD xx-xxxx
19	5.1.4.1	Understand/communicate	IEEE STD

		using IMAP protocol	xx-xxxx
20	5.1.4.2	Open at same rate as OE	Use Case Doc 4.5.4

Establish Business Rules for Contingencies and Responsibilities

"The best-laid plans of mice and men..." begins the famous saying. It has direct application to writing SRSs because even the most thought-out requirements are not immune to changes in industry, market, or government regulations. A top-quality SRS should include plans for planned and unplanned contingencies, as well as an explicit definition of the responsibilities of each party, should a contingency be implemented. Some business rules are easier to work around than others, when Plan B has to be invoked. For example, if a customer wants to change a requirement that is tied to a government regulation, it may not be ethical and/or legal to be following "the spirit of the law." Many government regulations, as business rules, simply don't allow any compromise or "wiggle room." A project manager may be responsible for ensuring that a government regulation is followed as it relates to a project requirement; however, if a contingency is required, then the responsibility for that requirement may shift from the project manager to a regulatory attorney. The SRS should anticipate such actions to the furthest extent possible.

Establish a Requirements Traceability Matrix

The business rules for contingencies and responsibilities can be defined explicitly within a Requirements Traceability Matrix (RTM), or contained in a separate document and referenced in the matrix, as the example in Table 3 illustrates. Such a practice leaves no doubt as to responsibilities and actions under certain conditions as they occur during the product-development phase.

The RTM functions as a sort of "chain of custody" document for requirements and can include pointers to links from requirements to sources, as well as pointers to business rules. For example, any given requirement must be traced back to a specified need, be it a use case, business essential, industry-recognized standard, or government regulation. As mentioned previously, linking requirements with sources minimizes or even eliminates the presence of spurious or frivolous requirements that lack any justification. The RTM is another record of mutual understanding, but also helps during the development phase.

As software design and development proceed, the design elements and the actual code must be tied back to the requirement(s) that define them. The RTM is completed as development progresses; it can't be completed beforehand (see Table 3).

What Should I Know about Writing an SRS?

Unlike formal language that allows developers and designers some latitude, the natural language of SRSs must be exact, without ambiguity, and precise because the design specification, statement of work, and other project documents are what drive the development of the final product. That final product must be tested and validated against the design and original requirements. Specification language that allows for interpretation of key requirements will not yield a satisfactory final product and will likely lead to cost overruns, extended schedules, and missed deliverable deadlines.

Table 4 shows the fundamental characteristics of a quality SRS, which were originally presented at the April 1998 Software Technology Conference presentation "Doing Requirements Right the First

Time." Reprinted with permission from the Software Assurance Technology Center at NASA (http://www.gsfc.nasa.gov/). These quality characteristics are closely tied to what are referred to as "indicators of strength and weakness," which will be defined next.

Table 4 The 10 language quality characteristics of an SRS

SRS Quality Characteristic	**What It Means**
Complete	SRS defines precisely all the go-live situations that will be encountered and the system's capability to successfully address them.
Consistent	SRS capability functions and performance levels are compatible, and the required quality features (security, reliability, etc.) do not negate those capability functions. For example, the only electric hedge trimmer that is safe is one that is stored in a box and not connected to any electrical cords or outlets.
Accurate	SRS precisely defines the system's capability in a real-world environment, as well as how it interfaces and interacts with it. This aspect of requirements is a significant problem area for many SRSs.
Modifiable	The logical, hierarchical structure of the SRS should facilitate any necessary modifications (grouping related issues together and separating them from unrelated issues makes the SRS easier to modify).
Ranked	Individual requirements of an SRS are hierarchically arranged according to stability, security, perceived ease/difficulty of implementation, or other parameter that helps in the design of that and subsequent documents.
Testable	An SRS must be stated in such a manner that unambiguous assessment criteria (pass/fail or some quantitative measure) can be derived from the SRS itself.
Traceable	Each requirement in an SRS must be uniquely identified to a source (use case, government requirement, industry standard, etc.)
Unambiguous	SRS must contain requirements statements that can be interpreted in one way only. This is another area that creates significant problems for SRS development because of the use of natural language.

Valid	A valid SRS is one in which all parties and project participants can understand, analyze, accept, or approve it. This is one of the main reasons SRSs are written using natural language.
Verifiable	A verifiable SRS is consistent from one level of abstraction to another. Most attributes of a specification are subjective and a conclusive assessment of quality requires a technical review by domain experts. Using indicators of strength and weakness provide some evidence that preferred attributes are or are not present.

What makes an SRS "good?" How do we know when we've written a "quality" specification? The most obvious answer is that a quality specification is one that fully addresses all the customer requirements for a particular product or system. That's part of the answer. While many quality attributes of an SRS are subjective, we do need indicators or measures that provide a sense of how strong or weak the language is in an SRS. A "strong" SRS is one in which the requirements are tightly, unambiguously, and precisely defined in such a way that leaves no other interpretation or meaning to any individual requirement.

The Goddard Space Flight Center (GSFC) studied dozens of NASA requirements specifications that revealed nine categories of SRS quality indicators. The individual components in each category are words, phrases, and sentence structures that are related to quality attributes. The nine categories fall into two classes: those related to individual specification statements, and those related to the total SRS document. Table 5 summarizes the classes, categories, and components of these quality indicators. This table was also originally presented at the April 1998 Software Technology Conference presentation "Doing Requirements Right the First Time." Reprinted with permission from the Software Assurance Technology Center at NASA (http://www.gsfc.nasa.gov/).

Table 5 Quality measures related to individual SRS statements

Imperatives: Words and phrases that command the presence of some feature, function, or deliverable. They are listed below in decreasing order of strength.	
Shall	Used to dictate the provision of a functional capability.
Must or must not	Most often used to establish performance requirement or constraints.
Is required to	Used as an imperative in SRS statements when written in passive voice.
Are applicable	Used to include, by reference, standards, or other documentation as an addition to the requirement being specified.
Responsible	Used as an imperative in SRSs that are written

for	for systems with pre-defined architectures.
Will	Used to cite things that the operational or development environment is to provide to the capability being specified. For example, The vehicle's exhaust system will power the ABC widget.
Should	Not used often as an imperative in SRS statements; however, when used, the SRS statement always reads weak. Avoid using Should in your SRSs.

Continuances: Phrases that follow an imperative and introduce the specification of requirements at a lower level. There is a correlation with the frequency of use of *continuances* and SRS organization and structure, up to a point. Excessive use of *continuances* often indicates a very complex, detailed SRS. The *continuances* below are listed in decreasing order of use within NASA SRSs. Use *continuances* in your SRSs, but balance the frequency with the appropriate level of detail called for in the SRS.

1. Below:

2. As follows:

3. Following:

4. Listed:

5. In particular:

6. Support:

Directives: Categories of words and phrases that indicate illustrative information within the SRS. A high ratio of total number of *directives* to total text line count appears to correlate with how precisely requirements are specified within the SRS. The *directives* below are listed in decreasing order of occurrence within NASA SRSs. Incorporate the use of *directives* in your SRSs.

1. Figure

2. Table

3. For example

4. Note

Options: A category of words that provide latitude in satisfying the SRS statements that contain them. This category of words loosens the SRS, reduces the client's control over the final product, and allows for possible cost and schedule risks. You should avoid using them in your SRS. The *options* below are listed in the order they are found most often in NASA SRSs.

1. Can

2. May

3. Optionally

Weak phrases: A category of clauses that can create uncertainty and multiple/subjective interpretation. The total number of *weak phrases* found in an SRS indicates the relative ambiguity and incompleteness of an SRS. The *weak phrases* below are listed alphabetically.

adequate	**be able to**	**easy**	**provide for**
as a minimum	**be capable of**	**effective**	**timely**
as applicable	**but not limited to**	**if possible**	**tbd**
as appropriate	**capability of**	**if practical**	
at a minimum	**capability to**	**normal**	

Size: Used to indicate the *size* of the SRS document, and is the total number of the following:

1. Lines of text

2. Number of imperatives

3. Subjects of SRS statements

4. Paragraphs

Text Structure: Related to the number of statement identifiers found at each hierarchical level of the SRS and indicate the document's organization, consistency, and level of detail. The most detailed NASA SRSs were nine levels deep. High-level SRSs were rarely more than four levels deep. SRSs deemed well organized and a consistent level of detail had *text structures* resembling pyramids (few level 1 headings but each lower level having more numbered statements than the level above it). Hour-glass-shaped *text structures* (many level 1 headings, few a mid-levels, and many at lower levels) usually contain a greater amount of introductory and administrative information. Diamond-shaped *text structures* (pyramid shape followed by decreasing statement counts at levels below the pyramid) indicated that subjects introduced at higher levels were addressed at various levels of detail.

Specification Depth: The number of imperatives found at each of the SRS levels of text structure. These numbers include the count of lower level list items that are introduced at a higher level by an imperative and followed by a continuance. The numbers provide some insight into how much of the Requirements document was included in the SRS, and can indicate how concise the SRS is in specifying the requirements.

Readability Statistics: Measurements of how easily an adult can read and understand the requirements document. Four readability statistics are used (calculated by Microsoft Word). While readability statistics provide a relative quantitative measure, don't sacrifice sufficient technical depth in your SRS for a number.

1. Flesch Reading Ease index

2. Flesch-Kincaid Grade Level index

3. Coleman-Liau Grade Level index

4. Bormuth Grade Level index

Conclusion

There's so much more we could say about requirements and specifications. Hopefully, this information will help you get started when you are called upon--or step up--to help the development team. Writing top-quality requirements specifications begins with a complete definition of customer requirements. Coupled with a natural language that incorporates strength and weakness quality indicators--not to mention the adoption of a good SRS template--technical communications professionals well-trained in requirements gathering, template design, and natural language use are in the best position to create and add value to such critical project documentation.

Additional Resources

Brooks, Frederick P. Jr., No Silver Bullet: Essence and accidents of software engineering, IEEE Computer, vol. 15, no. 1, April 1987, pp. 10-18.

Gause, Donald C., and Weinberg, Gerald M., Exploring Requirements Quality Before Design, Dorset House Publishing, NY, NY, 1989.

IEEE Std 830-1993, Recommended Practice for Software Requirements Specifications, December 2, 1993.

Wiegers, Karl E. Software Requirements, Microsoft Press, Redmond, WA, 1999.

Resources for Model Templates

As previously noted, you should first look for SRS documents developed by your company. Not only are these documents readily available to you, but also they're likely for products that are similar to the product you're developing an SRS for. Additional resources include:

- IEEE Standard 830-1998 describes an SRS template (Table 1 is an adaptation and extension of this template).
- Ken Rigby's informative Web site at http://neon.airtime.co.uk/users/wysywig/srs_mt.htm includes sample specification templates and articles about writing various types of specifications.
- STC's Management Special Interest Group (SIG) offers a Reference Documents Page at http://www.stcsig.org/mgt/reference.htm that includes a sample template and other related resources.
- Other sites are also available on the Web that offer examples of actual SRSs and templates. A quick search for "software requirements specifications" at your favorite search engine should yield some useful results.

Donn Le Vie, Jr. has helped develop requirements specifications for the hardware and software fields, the earth and space sciences, and the eCommerce industry. Donn works for Intel Corporation's Communications Group, and Wireless Communication and Computing Group, where he provides process development and information engineering support for wireless/mobile products and storage area network projects that use the StrongARM® and XScale™ microarchitectures. He has authored more than 60 technical/industry-focused articles, 700 general-interest articles, and two nonfiction books. His latest book, Designing Successful eBusiness Proposals, *will be available later this year. He can be reached at donald.s.levie@intel.com..*

Pssst!

- See also *The Inspection Method: An Approach to Planning and Managing a Successful Team Document Review* by Donn Le Vie, Jr.
- Send this article to someone you know!

http://www.techwr-l.com/techwhirl/magazine/writing/softwarerequirementspecs.html

This page printed from: http://www.techwr-l.com/techwhirl/magazine/writing/softwarerequirementspecs.html