Software Requirements Specification

Abstract:			
Keywords:			
Approved:			
Author(s)	**Project Manager(s)**	**Product Owner**	
Andy Torres	**Sebastian Lopez**		
	Dan		

General information/recommendations

A SRS provides a description of the software requirements in order to start the design and development process.
The SRS describes the contract that needs to be satisfied by the software product.
This document is intended to be used to collect, translate and classify the requirements. These requirements can be explicitly expressed by the clients (product owner, users or other stakeholders) or can be implicitly deduced as a result of meetings, presentations, interviews, etc.
Requirements need to be "validated" by the clients. In order to guarantee an adequate validation, requirements needs to be represented not only in textual form, but also using standard modeling notations such us UML/SysML (e.g. use cases, sequence, class, activity, statechart diagrams, etc.).

A matrix of well identified/classified/approved requirements needs to be produced during the requirement analysis process. This matrix will be use to trace how the software product satisfies the requirements during the whole process (i.e. the various iterations or releases).
More information about requirements traceability matrix can be found at http://en.wikipedia.org/wiki/Traceability_matrix)

This template document is based on the IEEE 830 standard.
This template can be used directly or it can be adapted in order to better fit the followed software design process.

This document is organized as following:
Section 1 gives a global introduction to the document and the software product.
Section 2 informally introduces the software product and the context where it will be used.
Section 3 specifies the requirements for the software product.
The Appendix section gives general guidelines to write and organize the requirements and proposes a very important SRS element: ***the traceability matrix.***

Revision History

Version	Date	Author	Change Description
1.0			

TABLE OF CONTENTS

Section 1. INTRODUCTION

Purpose

a) Provide an overview of the entire document.

b) Delineate the purpose of the SRS;

c) Specify the intended audience for the SRS.

Quick answers for now.

a) A roadmap

Scope

a) Identify the software product;

b) Explain what the software product(s) will, and, if necessary, will not do;

c) Describe the application of the software being specified (benefits, objectives, and goals);

d) Be consistent with similar statements in higher-level specifications (e.g., the system requirements specification), if they exist.

Overview

a) Describe what the rest of the SRS contains;

b) Explain how the SRS is organized.

Definitions, Acronyms, and Abbreviations

Definitions of all terms, acronyms, and abbreviations required to properly interpret the SRS

All non-standard terms, acronyms and abbreviations that are unique to this document should be included in this section. References to other appendices or reference documents may be included. Index information should be in bold.

- ***IEEE** – Institute of Electrical and Electronics Engineers*
- ***SRS** – System Requirements Specification*
- ***PRD** – Product Requirements Document*
- ***MRD** – Marketing Requirements Document*

Also see IEEE Std 1002-1987, IEEE Standard Taxonomy for Software Engineering Standards.

Term/Acronym	Definition

References

a) Provide a complete list of all documents referenced elsewhere in the SRS;

b) Identify each document by title, report number (if applicable), date, and publishing organization;

c) Specify the sources from which the references can be obtained.

Section 2. General Description

This section of the SRS should describe the general factors that affect the product and its requirements. This section does not state specific requirements. Instead, it provides a background for those requirements, which are defined in detail in Section 3 of the SRS, and makes them easier to understand.

Product Perspective

This subsection of the SRS should put the product into perspective with other related products. If the product is independent and totally self-contained, it should be so stated here. If the SRS defines a product that is a component of a larger system, as frequently occurs, then this subsection should relate the requirements of that larger system to functionality of the software and should identify interfaces between that system and the software.

A block diagram showing the major components of the larger system, interconnections, and external interfaces can be helpful. Likewise, any informal diagram describing the context and the software product could be used for illustration purposes.

This subsection should also describe how the software operates inside various constraints. For example, these constraints could include

a) System interfaces;

b) User interfaces;

c) Hardware interfaces;

d) Software interfaces;

e) Communications interfaces;

f) Memory;

g) Operations;

h) Site adaptation requirements.

Product Functions

This subsection of the SRS should provide a summary of the major functions that the software will perform. Sometimes the function summary that is necessary for this part can be taken directly from the section of the higher-level specification (if one exists) that allocates particular functions to the software product. Note that for the sake of clarity

a) The functions should be organized in a way that makes the list of functions understandable to the customer or to anyone else reading the document for the first time.

b) Textual or graphical methods can be used to show the different functions and their relationships (e.g. use case diagrams or any informal diagram)

User Characteristics

This subsection of the SRS should describe those general characteristics of the intended users of the product including educational level, experience, and technical expertise. It should not be used to state specific requirements, but rather should provide the reasons why certain specific requirements are later specified in Section 3 of the SRS

General Constraints

This subsection of the SRS should provide a general description of any other items that will limit the developer's options. These include

a) Regulatory policies;

b) Hardware limitations (e.g., signal timing requirements);

c) Interfaces to other applications;

d) Parallel operation;

e) Audit functions;

f) Control functions;

g) Higher-order language requirements;

h) Signal handshake protocols (e.g., XON-XOFF, ACK-NACK);

i) Reliability requirements;

j) Criticality of the application;

k) Safety and security considerations.

Assumptions and Dependencies

This subsection of the SRS should list each of the factors that affect the requirements stated in the SRS. These factors are not design constraints on the software but are, rather, any changes to them that can affect the requirements in the SRS.

Section 3. Requirements

This section of the SRS should contain all of the software requirements to a level of detail suficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements. Throughout this section, every stated requirement should be externally perceivable by users, operators, or other external systems. These requirements should include at a minimum a description of every input (stimulus) into the system, every output (response) from the system, and all functions performed by the system in response to an input or in support of an output. As this is often the largest and most important part of the SRS, the following principles apply:

a) Specific requirements should be stated in conformance with all the characteristics described in Annex 1

b) Specific requirements should be cross-referenced to earlier documents that relate.

c) All requirements should be uniquely identifiable.

d) Careful attention should be given to organizing the requirements to maximize readability.

Functional Requirements

Functional requirements should define the fundamental actions that must take place in the software in accepting and processing the inputs and in processing and generating the outputs. These are generally listed as "shall" statements starting with "The system shall". These include:

a) Validity checks on the inputs

b) Exact sequence of operations

c) Responses to abnormal situations, including

1) Overflow

2) Communication facilities

3) Error handling and recovery

d) Effect of parameters

e) Relationship of outputs to inputs, including

1) Input/output sequences

2) Formulas for input to output conversion

It may be appropriate to partition the functional requirements into subfunctions or subprocesses. This does not imply that the software design will also be partitioned that way.

External Interface Requirements

This should be a detailed description of all inputs into and outputs from the software system. It should complement the interface descriptions in 2 and should not repeat information there.

It should include both content and format as follows:

a) Name of item;

b) Description of purpose;

c) Source of input or destination of output;

d) Valid range, accuracy, and/or tolerance;

e) Units of measure;

f) Timing;

g) Relationships to other inputs/outputs;

h) Screen formats/organization;

i) Window formats/organization;

j) Data formats;

k) Command formats;

l) End messages.

User Interfaces

This should specify the following:

a) The requirements of each interface between the software product and its users.

This includes those configuration characteristics (e.g., required screen formats, page or window layouts, content of any reports or menus, or availability of programmable function keys) necessary to accomplish the software requirements.

b) All the aspects of optimizing the interface with the person who must use the system.

This may simply comprise a list of do's and don'ts on how the system will appear to the user. One example may be a requirement for the option of long or short error messages. Like all others, these requirements should be verifiable

Hardware Interfaces

This should specify the requirements of each interface between the software product and the hardware components of the system. This includes configuration characteristics (number of ports, instruction sets, etc.). It also covers such matters as what devices are to be supported, how they are to be supported, and protocols. For example, terminal support may specify full-screen support as opposed to line-by-line support.

Software Interfaces

This should specify the use of other required software products (e.g., a data management system, an operating system, or a mathematical package), and interfaces with other application systems. For each required software product, the following should be provided:

- Name;

- Mnemonic;

- Specification number;

- Version number;

- Source.

For each interface, the following should be provided:

- Discussion of the purpose of the interfacing software as related to this software product.

- Definition of the interface in terms of message content and format. It is not necessary to detail any well-documented interface, but a reference to the document defining the interface is required.

Communications Interfaces

This should specify the various interfaces to communications such as local network protocols, etc.

Performance Requirements

This subsection should specify both the static and the dynamic numerical requirements placed on the software or on human interaction with the software as a whole. Static numerical requirements may include the following:

a) The number of terminals to be supported;

b) The number of simultaneous users to be supported;

c) Amount and type of information to be handled.

Dynamic numerical requirements may include, for example, the numbers of transactions and tasks and the amount of data to be processed within certain time periods for both normal and peak workload conditions.

All of these requirements should be stated in measurable terms. For example,

95% of the transactions shall be processed in less than 1 s.

rather than,

An operator shall not have to wait for the transaction to complete.

Design Constraints

This should specify design constraints that can be imposed by other standards, hardware limitations, etc.

Standards Compliance

This subsection should specify the requirements derived from existing standards or regulations. They may include the following:

a) Report format;

b) Data naming;

c) Accounting procedures;

d) Audit tracing.

For example, this could specify the requirement for software to trace processing activity. Such traces are needed for some applications to meet minimum regulatory or financial standards. An audit trace requirement may, for example, state that all changes to a payroll database must be recorded in a trace file with before and after values.

Hardware Limitations

Attributes

There are a number of attributes of software that can serve as requirements. It is important that required attributes be specified so that their achievement can be objectively verified.

Availability

This should specify the factors required to guarantee a defined availability level for the entire system such as checkpoint, recovery, and restart.

Security

This should specify the factors that protect the software from accidental or malicious access, use, modification, destruction, or disclosure. Specific requirements in this area could include the need to:

a) Utilize certain cryptographical techniques;

b) Keep specific log or history data sets;

c) Assign certain functions to different modules;

d) Restrict communications between some areas of the program;

e) Check data integrity for critical variables.

Maintainability

This should specify attributes of software that relate to the ease of maintenance of the software itself. There may be some requirement for certain modularity, interfaces, complexity, etc. Requirements should not be placed here just because they are thought to be good design practices.

Transferability/Conversion

This should specify attributes of software that relate to the ease of porting the software to other host machines and/or operating systems. This may include the following:

a) Percentage of components with host-dependent code;

b) Percentage of code that is host dependent;

c) Use of a proven portable language;

d) Use of a particular compiler or language subset;

e) Use of a particular operating system.

Other Requirements

Operations

This should specify the normal and special operations required by the user such as

a) The various modes of operations in the user organization (e.g., user-initiated operations);

b) Periods of interactive operations and periods of unattended operations;

c) Data processing support functions;

d) Backup and recovery operations.

Site Adaptation

This should

a) Define the requirements for any data or initialization sequences that are specific to a given site, mission, or operational mode (e.g., grid values, safety limits, etc.);

b) Specify the site or mission-related features that should be modified to adapt the software to a particular installation.

Appendixes

The appendixes are not always considered part of the actual SRS and are not always necessary. They may include:

a) Sample input/output formats, descriptions of cost analysis studies, or results of user surveys;

b) Supporting or background information that can help the readers of the SRS;

c) A description of the problems to be solved by the software;

d) Special packaging instructions for the code and the media to meet security, export, initial *loading, or*

other requirements.

When appendixes are included, the SRS should explicitly state whether or not the appendixes are to be considered part of the requirements.

1. *Annex 1: Characteristics of a good SRS*

An SRS should be

a) Correct: An SRS is correct if, and only if, every requirement stated therein is one that the software shall meet.

b) Unambiguous: An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation. As a minimum, this requires that each characteristic of the final product be described using a single unique term.

c) Complete: An SRS is complete if, and only if, it includes the following elements:

c.1) All significant requirements, whether relating to functionality, performance, design constraints, attributes, or external interfaces. In particular any external requirements imposed by a system specification should be acknowledged and treated.

c.2) Definition of the responses of the software to all realizable classes of input data in all realizable classes of situations. Note that it is important to specify the responses to both valid and invalid input values.

c.3) Full labels and references to all figures, tables, and diagrams in the SRS and definition of all terms and units of measure.

d) Consistent: Consistency refers to internal consistency. (Note: If an SRS does not agree with some higher-level document, such as a system requirements specification, then it is not correct). An SRS is internally consistent if, and only if, no subset of individual requirements described in it conflict.

e) Ranked for importance and/or stability: all of the requirements that relate to a software product are not equally important. Some requirements may be essential, especially for life-critical applications, while others may be desirable. Each requirement in the SRS should be identified to make these differences clear and explicit.

e.1) Degree of necessity:

- Essential: Implies that the software will not be acceptable unless these requirements are provided in an agreed manner.

- Conditional: Implies that these are requirements that would enhance the software product, but would not make it unacceptable if they are absent.

- Optional: Implies a class of functions that may or may not be worthwhile. This gives the supplier the opportunity to propose something that exceeds the SRS.

e.2) Degree of stability: One method of identifying requirements uses the dimension of stability. Stability can be expressed in terms of the number of expected changes to any requirement based on experience or knowledge of forthcoming events that affect the organization, functions, and people supported by the software system.

f) Verifiable: An SRS is verifiable if, and only if, every requirement stated therein is verifiable. A requirement is verifiable if, and only if, there exists some finite cost-effective process with *which a person or machine can check that the software product meets the requirement. In general any ambiguous requirement is not verifiable. Nonverifiable requirements include statements such as "works well", "good human interface", or "shall usually happen".*

If a method cannot be devised to determine whether the software meets a particular requirement, then that requirement should be removed or revised.

g) Modifiable: An SRS is modifiable if, and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style. Modifiability generally requires an SRS to:

g.1) Have a coherent and easy-to-use organization with a table of contents, an index, and explicit crossreferencing;

g.2) Not be redundant (i.e., the same requirement should not appear in more than one place in the SRS);

g.3) Express each requirement separately, rather than intermixed with other requirements.

h) Traceable: An SRS is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation. The following two types of traceability

are recommended:

h.1) Backward traceability (i.e., to previous stages of development): This depends upon each requirement

explicitly referencing its source in earlier documents.

h.2) Forward traceability (i.e., to all documents spawned by the SRS): This depends upon each requirement

in the SRS having a unique name or reference number. The forward traceability of the SRS is especially important when the software product enters the operation and maintenance phase. As code and design documents are modified, it is essential to be able to ascertain the complete set of requirements that may be affected by those modifications.

2. *Annex 2: Organizing the specific requirements*

For anything but trivial systems the detailed requirements tend to be extensive. For this reason, it is recommended that careful consideration be given to organizing these in a manner optimal for understanding. There is no one optimal organization for all systems. Different classes of systems lend themselves to different organizations of requirements in Section 3 of the SRS. Some of these organizations are:

a) System mode

Some systems behave quite differently depending on the mode of operation. For example, a control system may have different sets of functions depending on its mode: training, normal, or emergency. When organizing this section by mode, the outline in Annex 5.3.1. The choice depends on whether interfaces and performance are dependent on mode.

b) User class

Some systems provide different sets of functions to different classes of users. For example, an elevator control system presents different capabilities to passengers, maintenance workers, and fire fighters. When organizing this section by user class, the outline in Annex 5.3.2 should be used.

c) Objects

Objects are real-world entities that have a counterpart within the system. For example, in a patient monitoring system, objects include patients, sensors, nurses, rooms, physicians, medicines, etc. Associated with each object is a set of attributes (of that object) and functions (performed by that object). These functions are also called services, methods, or processes. When organizing this section by object, the outline in Annex 5.3.3 should be used. Note that sets of objects may share attributes and services. These are grouped together as classes.

d) Feature

A feature is an externally desired service by the system that may require a sequence of inputs to effect the desired result. For example, in a telephone system, features include local call, call forwarding, and conference call. Each feature is generally described in a sequence of stimulus-response pairs. When organizing this section by feature, the outline in Annex 5.3.4 should be used.

e) Stimulus

Some systems can be best organized by describing their functions in terms of stimuli. For example, the functions of an automatic aircraft landing system may be organized into sections for loss of power, wind shear, sudden change in roll, vertical velocity excessive, etc. When organizing this section by stimulus, the outline in Annex 5.3.5 should be used.

f) Response

Some systems can be best organized by describing all the functions in support of the generation of a response. For example, the functions of a personnel system may be organized into sections corresponding to all functions associated with generating paychecks, all functions associated with generating a current list of employees, etc. The outline in Annex 5.3.5 (with all occurrences of stimulus replaced with response) should be used.

g) Functional hierarchy

When none of the above organizational schemes prove helpful, the overall functionality can be organized into a hierarchy of functions organized by either common inputs, common outputs, or common internal data access. Data flow diagrams and data dictionaries can be used to show the relationships between and among the functions and data. When organizing this section by functional hierarchy, the outline in Annex 5.3.6 should be used.

Additional comments:

Whenever a new SRS is contemplated, more than one of the organizational techniques given here may be appropriate. In such cases, organize the specific requirements for multiple hierarchies tailored to the specific needs of the system under specification. For example, see Annex 5.3.7 for an organization combining user class and feature. Any additional requirements may be put in a separate section at the end of the SRS.

There are many notations, methods, and automated support tools available to aid in the documentation of requirements. For the most part, their usefulness is a function of organization. For example, when organizing by mode, finite state machines or state charts may prove helpful; when organizing by object, object-oriented analysis may prove helpful; when organizing by feature, stimulus-response sequences may prove helpful; and when organizing by functional hierarchy, data flow diagrams and data dictionaries may prove helpful.

In any of the outlines given in Annex 5.3.1 through Annex 5.3.7, those sections called

"Functional Requirement i" may be described in native language (e.g., English), in pseudocode, in a system definition language, or in four subsections titled: Introduction, Inputs, Processing, and Outputs.

3. ***Annex 3: SRS templates***

1. **Template of SRS Section 3 organized by feature**

3. Specific requirements
3.1 External interface requirements
3.1.1 User interfaces
3.1.2 Hardware interfaces
3.1.3 Software interfaces
3.1.4 Communications interfaces
3.2 System features
3.2.1 System Feature 1
3.2.1.1 Introduction/Purpose of feature
3.2.1.2 Stimulus/Response sequence
3.2.1.3 Associated functional requirements
3.2.1.3.1 Functional requirement 1
.
.
.
3.2.1.3.n Functional requirement n
3.2.2 System feature 2
.
.
.
3.2.m System feature m
.
.
.
3.3 Performance requirements
3.4 Design constraints
3.5 Software system attributes
3.6 Other requirements

2. **Template of SRS Section 3 organized by user class**

3. *Specific requirements*
3.1 External interface requirements
3.1.1 User interfaces
3.1.2 Hardware interfaces
3.1.3 Software interfaces
3.1.4 Communications interfaces
3.2 Functional requirements
3.2.1 User class 1
3.2.1.1 Functional requirement 1.1
.
.
3.2.1.n Functional requirement 1.n
3.2.2 User class 2

.
.
3.2.m User class m
3.2.m.1 Functional requirement m.1
.
.
3.2.m.n Functional requirement m.n
3.3 Performance requirements
3.4 Design constraints
3.5 Software system attributes
3.6 Other requirements

3. **Template of SRS Section 3 showing multiple organizations**

3. *Specific requirements*
3.1 External interface requirements
3.1.1 User interfaces
3.1.2 Hardware interfaces
3.1.3 Software interfaces
3.1.4 Communications interfaces
3.2 Functional requirements
3.2.1 User class 1
3.2.1.1 Feature 1.1
3.2.1.1.1 Introduction/Purpose of feature
3.2.1.1.2 Stimulus/Response sequence
3.2.1.1.3 Associated functional requirements
3.2.1.2 Feature 1.2
3.2.1.2.1 Introduction/Purpose of feature
3.2.1.2.2 Stimulus/Response sequence
3.2.1.2.3 Associated functional requirements
.
.
3.2.1.m Feature 1.m
3.2.1.m.1 Introduction/Purpose of feature
3.2.1.m.2 Stimulus/Response sequence
3.2.1.m.3 Associated functional requirements
3.2.2 User class 2
.
.
3.2.n User class n
.
.
3.3 Performance requirements
3.4 Design constraints
3.5 Software system attributes
3.6 Other requirements

Annex 4: SRS Traceability Matrix

User (or	Refined list of functional or	Link to a refined list	Priority

feature)	non-functional requirements	of technical requirements	(High, Medium, Low)
e.g.	e.g.	e.g.	e.g.
Client	UC1: Buy a product:	Cookbooks C1, C2	H
			H
	1.1. Browse products	C1.Recipe2.1: create	H
		list of items	M
	1.1.1. by category	C1.Recipe2.1.1: sort	H
		list by a column	H
	1.1.2. by keyword(s)	C1.Recipe2.1.2: filter	H
		list by a column	H
		keyword	H
	1.2 Select product	C1.Recipe2.2: select	L
		row	
	1.3. Indicate amount	C1.Recipe2.2: enter	
		valid number field	
	1.4. Check inventory	C1.Recipe4.1: execute	
		query and return	
		evaluation of a	
		condition	
	1.5. Select pay method	C2.Recipe1.1: select	
		from radio button or list	
		C2.Recipe1.1.1:	
	1.5.1. Process credit card payment	execute payment	
		webservice	
		C2.Recipe1.1.2:	
	1.5.2. Process check payment	execute check web	
		service	
		…..	
	1.5.3. ….		
….			