Software Requirements Specification

for

<Project>

Version 1.0 approved

Prepared by <author>

<organization>

<date created>

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

<Identify the product whose software requirements are specified in this document, including the revision or release number. Describe the scope of the product that is covered by this SRS, particularly if this SRS describes only part of the system or a single subsystem.>

1~2 paragraphs

1.2 Glossary

<Define the technical terms used in this document. Do not assume the experience or expertise of the reader. Each type of reader will have a technical vocabulary not necessarily shared by other readers.>

Define non-common terms,
DO NOT assume engineering background

1.3 Intended Audience and Reading Suggestions

<Describe the different types of reader that the document is intended for, such as developers, project managers, marketing staff, users, testers, and documentation writers. Describe what the rest of this SRS contains and how it is organized. Suggest a sequence for reading the document, beginning with the overview sections and proceeding through the sections that are most pertinent to each reader type.>

Recall the professor is your customer

1.4 Product Scope

<Provide a short description of the software being specified and its purpose, including relevant benefits, objectives, and goals. Relate the software to corporate goals or business strategies. If a separate vision and scope document is available, refer to it rather than duplicating its contents here.>

What is the purpose of the software?
1~2 paragraphs

1.5 References

<List any other documents or Web addresses to which this SRS refers. These may include user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document. Provide enough information so that the reader could access a copy of each reference, including title, author, version number, date, and source or location.>

These references are for further reading & citation

2. Overall Description

2.1 System Environment

<(Called the system's context in the *Domain* Model by Jacobson, *et al, System Model* by Sommerville and *Product Perspective* by IEEE)
This describes the relationship between the system, its components and the external environment of the system. ***The purpose of this diagram is to clearly show what is part of your system and what is not part of your system***. If it is a stand-alone single-user system, that information is noted here. The UML should be used for this diagram. Describe all the parts of the diagram including the external interfaces, such as the system interfaces, hardware interfaces, software interfaces (with other systems such as a DBMS), and communications interfaces, if needed. Briefly include any memory constraints, modes of operation, non-interactive operations, backup and recovery and any site adaptation requirements needed that may be relevant for the user to understand this diagram.

Use the UML to create labeled actor icons for each user class. Create actor icons for each class of user. Create actor icons for each external system with which your system will interface. An external clock that sends periodic messages to the system is an actor, for example. Create entities for (the parts of) your system to provide a clear explanation of the environment of your system. Use lines to show which units are related and which are not. Label these relationships. In this section, provide descriptions of the external interfaces (represented by actors).>

Draw UML diagram

UML Class diagram / Composite Structure Diagram
&
Topology / Deployment Diagram

2.2 Functional Requirements Definition

<Provide a detailed *overview* of the services provided to the user. Organize this part in a manner that is easy for the user to understand. It must be cross-referenced to the following chapter (Requirements Specification) where these functions are repeated in full detail. The cross-referencing *must* be explicit. There is no guarantee that the next section will have a similar organization. A use case here that is an apparently simple use case as shown to the user may require more than one detailed use case in the next section in order to consider all paths. Do not rely on a numbering scheme such as used for the sections of this document for reference purposes because if one part is renumbered, the other must then be made consistent. A better scheme is to provide an *identification* name or coded number to each use case in each section and to use the same identification (with extensions) in the next section. The cross-references must be correct. Use meaningful names in Verb-Noun format with very specific verbs and nouns. (*Process Items* would be a poor choice for a use case name as it is too vague.)

List all use cases (essential, desired, optional)*

The UML is used here to graphically support the text. The first step is to list all the use cases that might be part of the product. Each of these is classified as essential, desired (conditional) or optional. (Alternatively, the optional use cases may be delayed to the System Evolution section.) Use a top-down approach to describing these use cases. Use the UML to create a graphical listing of all uses cases (functionalities) of the system for each class of user. A use case should provide value added to at least one actor.
A good formalism for each of these use case definitions is to lead with a diagram with all of its the actors and components mentioned (one diagram may be useful for several use cases). Give a brief description followed by an initial step-by-step description as noted here:

First step, draw diagrams of all actors & components mentioned

Use case: <<Use Case Name>>

Diagram:
<< insert diagram or reference here>>

UML use cases per user class (1 diagram may cover multiple use cases)

Brief Description
The use case <<name>> is initiated by the <<actor>> to <<explicit functionality>>. Possible relation to other use cases.

Initial Step-By-Step Description
Before this use case can be initiated, the <<actor>> has already <<preconditions>>.

1. The <actor>> <<initiates>>.
2. <<next action>>.
3. <<next action>>.etc.

Use *active* not *passive* voice. Remember that this is a *black box* approach. If two consecutive steps above are both system operations, you may have started doing *design* instead of specification. Normally, the user and the system alternate steps. Describe the most common path only in the step-by-step.

This is referred to by Sommerville as the *requirement definition* and must be readable by the user with more detail available in the *requirement specification* in the next section. As a practical matter, many of the brief descriptions can be filled in before starting to fill in the initial step-by-step descriptions.>

2.3 User Interface Specifications

<*Describe the characteristics of the intended user in terms of experience and technical expertise. At a minimum, give the characteristics of the interface for each class of users, that is, screen formats, page/window layouts, content of reports or menus. How should the system appear to the user? How detailed should error messages be? If you are using prototyping, sample interfaces may be provided but make clear what principles are required to allow consistent modifications. Sample user interfaces may be placed in an Appendix to this volume.*>

Target users based on experience and technical expertise

Screenshots, layouts, menus

Prototyping (sample interfaces)

What is the goal of the design?
What can be modified?

2.4 Non-Functional Requirements

<We emphasize that these are requirements that are *not* functional in nature. They do not describe what the system will not do. A careful statement of prohibited system behavior would be covered under the *functional* requirements.

This section includes constraints such as minimum memory requirements, regulatory policies, timing considerations, reliability and standards such as process or documentation standards. Remember that *all* requirements must be written in a form that is testable. This section is for the user. A full set of non-functional requirements for the developer is contained in the Requirements Specification below.>

Describe required system behavior

Must be testable

3. Requirements Specifications

3.1 External Interface Requirements

<*Start this section with a formal statement of the* external interface requirements*, that is, list formal requirements for user interfaces, hardware interfaces, software interfaces and communications interfaces. For a stand-alone system, this section should be included with* None *as the contents*>

Include any interface used

3.2 Functional Requirements

<Next provide the *Functional Requirement Specification*. This is a listing of each functionality of the system in full detail. That is, provide a detailed formal specification of the system requirements using, for example, the full descriptions in Use Case Model.

Full system functionality requirements, use cases included

The organization of this chapter may follow the organization of Section 2.2 Functional Requirements Definition above or it may be organized along different lines (discussed above). Each item here is explicitly cross-referenced back to section 2.2 and forward to the Use Case Realizations in the software design description document as that document is developed.

A possible grid for a requirement specification is as follows:

Use Case Name	
Priority	(this field is optional) *essential*, *desired* or *optional*
Trigger	
Precondition	
Basic Path	
Alternative Paths	(If none, so state.)
Postcondition	
Exception Paths	(If none, so state.)
Other	(this field is optional)

Can use table

Trigger is a slot for use cases that are not necessarily initiated by a human actor.
Precondition is the System State necessary before this use case can be initiated.
The *Basic Path* is a step-by-step description of the more common path.
In the Basic Path, repetition can be indicated as follows:
1. <<do something>>
1.1 (optional) <<do again until done>>

Of execution

Alternative Paths are other correct paths that may be pursued.
Postcondition is the System State required after this use case is accomplished.
Exception Paths are what happens when an error condition occurs. They may correct and return to the Basic (or an Alternate) Path or they my abort the use case.
Other can be used for non-functional requirements unique to this use case, to refer to external rules that must be observed here, or any other information that needs to be recorded. It may be used for the explicit cross-referencing or that can be added as an additional box.

Include UML to help explain
(State / Flow / Activity diagram)

Do not forget to consider validity checks on inputs. Also append any diagram (for example, a

state diagram) that will serve to make this description clearer. >

4. Other Nonfunctional Requirements

4.1 Performance Requirements

<If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.>

Performance requirements
(Designers will use)

4.2 Safety Requirements

<Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product's design or use. Define any safety certifications that must be satisfied.>

4.3 Security Requirements

<Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.>

Security
Privacy
Authentication