Rules of Writing Software Requirement Specifications

Short Note. Version 1a

Janusz Zalewski, FGCU

April 10, 2018

A properly written Software Requirements Specification should adhere to a number of rules that can be expressed as matching the following properties:

1) Clarity

The property of the specification that ensures its understanding and nonambiguous interpretation by the reader proficient in the specification language.

In other words, the document has to be readable and understandable.

Clarity is ensured when:

- each requirement is uniquely identified, for example, by a number (listing them just under bullet points is not acceptable)
- a requirement is expressed as one simple sentence containing one of the verbs “shall”, “must”, “should”, etc., accordingly with the strength of a requirement
- any additional explanations are included after the requirement, as a separate “Note”.

2) Completeness

The property of the specification that ensures inclusion of all the information necessary to develop a specified system.

In other words, the specification must ensure that nothing important is missing.

A simple example illustrates whether the specification is complete or not. Assume that the requirement reads:

“The software shall enable the User log into the system with username and password.”

If there is no other requirement specifying how the software should respond in case the User enters a wrong username or password, then the specification is incomplete. Also, if there is no other requirement specifying the maximum/minimum lengths and composition of the username and password, the specification is incomplete either.
3) Consistency

*The property of the specification, which ensures that there is no contradictory information in the specification document.*

In fact, consistency should also mean lack of redundancy, because a single concept explained more than once is more likely to be misinterpreted.

An example of the lack of consistency would be to specify the physical distance in one requirement, in SI units (meters) and in the U.S. units (feet), in a different requirement. One, not so obvious, example that traces back to inconsistency in requirements specification is the way multiple vehicle manufacturers in the U.S., construct the red lights on the back of the vehicles they produce. Very often the same back lights are assigned 2 functions: the Stop function (warning about stopping of the vehicle), as well as the Signal function (signaling the turn of the vehicle). This may lead to serious safety problems, and can be viewed as a specification defect. The bottom line is that a single variable (the Light) shall not be assigned two different functions (Stop and Signal).

4) Correctness

*The property of the specification that ensures this document's compliance with external requirements, such as related documents, standards, scientific knowledge, engineering practice, common sense, etc.*

One example of an incorrect requirement is the one which imposes an action on a user or an external device, for instance:

- A User shall be able to login to the system with a username and password.

This should be rewritten as follows:

- The Application shall be able to handle user logins with username and password.

Another common example of incorrect software requirement is the one, which requests the use of a specific operating system or a specific programming language, for example:

- The application shall be written in Java and run on Windows 10 computer.

This is not a software requirement, even though it involves software products: Java and Windows. First, requirements should be written only on software to be developed, not on system software. Second, if the customer requires the use of specific technology in the development of applications, this is a design constraint, never a software requirement.
5) Testability

*The extent to which an objective and feasible test can be designed to determine whether a requirement is met.* [IEEE]

From the practical perspective, one may want to look at the following factors for each requirement:
- can a specific input (or inputs) be determined, applied manually or automatically, or captured in the data stream influencing the behavior of your software, such that:
- a response or output of this software can be obtained to see, if the specific requirement considered is met.

6) Traceability

*The degree to which each element in a software development product establishes its reason for existing.* [IEEE]

Traceability ensures that all stakeholders know the path back, which led to the implementation of a specific feature, from the implementation back to a specific design component, from the design back to a specific requirement, and from the requirement back to its source. That is, every requirement should include a reference, who wrote it or who submitted it (whether an individual or an organization).

Before the requirements are actually written, there are two important diagrams, which have to be developed and drawn, as indispensable parts of the specification: a Physical Diagram of the system and a Context Diagram of software.

The Physical Diagram is defined as a representation of all physical entities that are involved during software operation, that is, the computer(s) the software will be running on, users, human operators and other clients, servers, the Internet, databases, external devices, etc.

The Context Diagram should mark the Software To-Be-Developed in the center, without any details, as a black box or circle, and all its interfaces with entities external to this software, showing just interfaces with descriptions what kind of information they carry in each direction, from the software towards the environment and back, from the environment to the software.
A Physical Diagram and Context Diagram for a simple case study of developing a Microwave Oven Controller Application are shown in Figures 1 and 2, respectively. Please note that a Physical Diagram is itself an abstraction, so showing it as illustrated in Fig. 0 is not appropriate for the purpose of software requirements specification. A better, abstract representation of the system is shown on a Physical Diagram in Fig. 1.

Fig. 0. A Photo of a Microwave Oven is not a Physical Diagram.

Fig. 1. Sample Physical Diagram of the System for Microwave Oven Controller.

A corresponding Context Diagram for the Microwave Oven Controller Software, featuring many more details than the Physical Diagram, is shown in Fig. 2. Normally, a Context Diagram has to include the interfaces featuring all interactions of the Software To-Be-Developed with the external environment. This includes the description of all signals, which are exchanged between the software and the environment, as illustrated in Fig. 2.
Very often, the software application being developed has to provide connectivity with the Internet. While it is not appropriate or even practical or safe for a Microwave Oven, it is certainly necessary for an ATM machine, which is illustrated on the Context Diagram in Fig. 3.

Another kind of diagram, which is commonly included in the Software Requirements Specification is a Use Case Diagram. Its content is defined in the UML notation. Please note that it is not a design diagram, though. The use case is showing “what” the software is supposed to do, not “how” this software will do it, therefore it cannot be categorized as a design diagram.
An example of such diagram for the Oven Controller Software is shown in Fig. 4. Use Case Diagrams are normally used to help develop the software requirements specification document.

Fig. 4. Sample Use Case Diagram for the Oven Microcontroller Software.

To summarize, two points are important to derive from this Note:

1) software requirements have to meet 6 important criteria of quality as stated at the beginning: clarity, completeness, consistency, correctness, testability and traceability;

2) before beginning to write the requirements, it is appropriate to illustrate the problem with three kinds of diagrams: the Physical Diagram of the entire system, the Context Diagram for software, and optionally the Use Case Diagram showing in general terms how the software is supposed to operate.

Finally, there are other important aspects of developing the Software Requirements Specification document. These topics are out of scope of this Short Note, however:

- a crucial requirements specification activity known as the elicitation of requirements
- the requirements review, done by a team or an engineer independent of the requirements development team
- requirements often change, so there should be a point agreed upon when they are frozen.

To keep things straight, one should also add that the examples shown above are very simple to illustrate the main concepts. Real life may be significantly more complicated and require a lot of other considerations.

*** END OF NOTE ***