

Chapter Four: Requirement Specification

A software requirements specification (SRS) is a detailed description of a software system to be developed with its functional and non-functional requirements. The SRS is developed based the agreement between customer and contractors. It may include the use cases of how user is going to interact with software system. The software requirement specification document consistent of all necessary requirements required for project development. To develop the software system, we should have clear understanding of Software system. To achieve this, we need to **continuous communication** with customers to gather all requirements.

A good SRS defines the how Software System will interact with all internal modules, hardware, communication with other programs and human user interactions with wide range of real-life scenarios. Using the Software requirements specification (SRS) document on QA lead, managers create test plan. It is very important that testers must be cleared with every detail specified in this document in order to avoid faults in test cases and its expected results. It is highly recommended to review or test SRS documents before start writing test cases and making any plan for testing. Let's see how to test SRS and the important point to keep in mind while testing it.

Following are the features of a good SRS document:

1. Correctness: User review is used to provide the **accuracy of requirements** stated in the SRS. SRS is said to be perfect if it covers all the needs that are **truly expected** from the system.
2. Completeness: The SRS is complete if, and only if, it includes the following elements:
 - (1). **All essential requirements**, whether relating to functionality, performance, design, constraints, attributes, or external interfaces.
 - (2). Definition of **their responses of the software** to all realizable **classes of input data** in all available categories of situations. Note: It is essential to specify the responses to both valid and invalid values.
 - (3). Full **labels** and references to all figures, tables, and diagrams in the SRS and definitions of all terms and units of measure.
3. Consistency: The SRS is consistent if, and only if, **no subset of individual requirements described in its conflict**. There are three types of possible conflict in the SRS:
 - (1). The specified characteristics of real-world objects may conflict. For example,
 - (a) The format of an output report may be described in one requirement as tabular but in another as textual.
 - (b) One condition may state that all lights shall be green while another states that all lights shall be blue.
 - (2). There may be a reasonable or **temporal conflict** between the two specified actions. For example,

(a) One requirement may determine that the program will add two inputs, and another may determine that the program will multiply them.

(b) One condition may state that "A" must always follow "B," while other requires that "A and B" co-occurs.

(3). Two or more requirements may define the **same real-world object but use different terms for that object**. For example, a program's request for user input may be called a "prompt" in one requirement's and a "cue" in another. The use of standard terminology and descriptions promotes consistency.

4. Unambiguousness: SRS is unambiguous when every fixed requirement **has only one interpretation**. This suggests that each element is uniquely interpreted. In case there is a method used with multiple definitions, the requirements report should determine the implications in the SRS so that it is clear and simple to understand.

5. Ranking for importance and stability: The SRS is ranked for importance and stability if each requirement in it has an **identifier** to indicate either the significance or stability of that particular requirement. Typically, all requirements are not equally important. Some prerequisites may be essential, especially for life-critical applications, while others may be desirable. Each element should be identified to make these differences clear and explicit. Another way to rank requirements is to distinguish classes of items as essential, conditional, and optional. **6. Modifiability:** SRS should be made as modifiable as likely and should be capable of **quickly obtain changes** to the system to some extent. Modifications should be perfectly indexed and cross-referenced.

7. Verifiability: SRS is correct when the specified requirements can be verified with a cost-effective system to check **whether the final software meets those requirements**. The requirements are verified with the help of reviews. **8. Traceability:** The SRS is **traceable if the origin of each of the requirements** is clear and if it facilitates the referencing of each condition in future development or enhancement documentation

There are two types of Traceability:

1. Backward Traceability: This depends upon each requirement explicitly referencing its source in earlier documents.

2. Forward Traceability: This depends upon each element in the SRS having a unique name or reference number. The forward traceability of the SRS is especially crucial when the software product enters the operation and maintenance phase. As code and design document is **modified**, it is necessary to be able to ascertain the complete set of requirements that may be concerned by those modifications.

9. Design Independence: There should be an option to select from multiple design alternatives for the final system. More specifically, the SRS should **not contain any implementation details**. **10. Testability:** An SRS should be written in such a method that it is simple to **generate test cases** and test plans from the report.

11. Understandable by the customer: An end user may be an expert in his/her explicit domain but might not be trained in computer science. Hence, the purpose of formal

notations and symbols should be avoided too as much extent as possible. The language should be kept simple and clear.

12. The right level of abstraction: If the SRS is written for the requirements stage, the details **should be explained explicitly**. Whereas, for a feasibility study, fewer analysis can be used. Hence, the level of abstraction modifies according to the objective of the SRS.

Properties of a good SRS document

The essential properties of a good SRS document are the following:

Concise: The SRS report should be concise and at the same time, unambiguous, consistent, and complete. **Verbose and irrelevant descriptions decrease readability** and also increase error possibilities.

Structured: It should be well-structured. A well-structured document is simple to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the user requirements. Often, user requirements evolve over a period of time. Therefore, **to make the modifications** to the SRS document easy, it is vital to make the report well-structured.

Black-box view: It should only define what the system **should do** and refrain from stating **how to** do these. This means that the SRS document should define the **external behavior** of the system and not discuss the **implementation issues**. The SRS report should view the system to be developed as a black box and should define the externally visible behavior of the system. For this reason, the SRS report is also known as the black-box specification of a system.

Conceptual integrity: It should show conceptual integrity so that the reader can merely understand it. **Response to undesired events:** It should characterize acceptable responses to unwanted events. These are called system response to exceptional conditions.

Verifiable: All requirements of the system, as documented in the SRS document, should be correct. This means that it should be possible to decide **whether or not requirements have been met in an implementation**.

In most organizations, system requirements are written as paragraphs of **natural language** (English, French, Japanese, etc.), supplemented by **diagrams and equations**. Natural language is the only notation that we have which is generally understandable by all potential readers of the requirements for a system. Natural language can be ambiguous, surprisingly opaque and is often misunderstood. Nevertheless, everyone can read natural language, so requirements will continue to be written in this way for the foreseeable future.

Natural language can be used to describe requirements clearly. All too often, however, natural language requirements are difficult to understand. **Common problems** are that:

- the requirements are written using **complex conditional clauses** (if A then if B then if C...) which are confusing
- terminology is used in a **sloppy and inconsistent way**

- the writers of the requirement assume that the reader has specific knowledge of the domain or the system and they leave essential information out of the requirements document.

These problems make it difficult to check the set of requirements for errors and omissions. Different interpretations of the requirements may lead to contractual disagreements between the customer and the system engineering contractor.

We do not think that these problems can be solved by introducing new languages for writing requirements. Good writing is all that is needed to write good requirements. Well-written requirements include all necessary information. They are succinct and are easy to understand. They use language that is as simple as possible. Technical terms are used in a consistent way and only when necessary.

We are convinced that, irrespective of the level of detail in your requirements description, there are three essential things that you must bear in mind when considering how to improve your requirements descriptions.

1. Requirements are read more often than they are written. Investing effort in writing requirements which are easy to read and understand is almost always cost-effective.
2. Readers of requirements come from diverse backgrounds. If you are a requirements writer, you should not assume that readers have the same background and knowledge as you.
3. Writing clearly and concisely is not easy. If you do not allow sufficient time for requirements descriptions to be drafted, reviewed and improved, you will inevitably end up with poorly written specifications.

Different organizations write requirements at different levels of abstraction from deliberately vague product specifications to detailed and precise descriptions of all aspects of a system. You must decide for yourself on the level of detail that you need. This depends on the type of requirements (stakeholder, system or process requirements), customer expectations, your organizational procedures, and external standards or regulations which you may have to follow. All we can do here is give you general advice which can be applied irrespective of the detail of the requirements.

A. Define Standard Templates for Describing Requirements

When you are writing detailed system requirements, you should define a set of standard templates which you use for organizing the description of these requirements. The template should include fields to collect the detailed information necessary to completely specify the requirement.

Benefits:

- Standards make requirements easier to read. Once readers understand a standard, they know what to expect when reading requirements.
- Standards make requirements easier to collect. The standard form can be used as a checklist by analysts when collecting requirements from end-users, customers, managers, etc.

- Standards make requirements **easier to write**. They reduce the chances that important information will be forgotten and make it easier for reviewers to check if information has been left out.

Implementation:

To ensure that natural language requirements are readable, you should define some structure for the description of each requirement. This structure should include fields to note the information that you wish to Define Standard Templates for associate with the requirement. For example, if the requirement is a **functional requirement**, your standard may specify that the **inputs, the processing and the outputs** must be specified in the requirements definition.

A standard form or template should be defined to record requirements information. These forms should have named fields **which remind the analyst which information is to be collected and recorded**. These replace unstructured sentences which describe the requirement. This template should be based on the type of system which you are specifying and the specific information needs of your organization. You will need to define several templates to cater for different types of requirements.

Within your requirements template you may wish to include fields to support other guidelines such as the requirements reference number (Guideline 9.1, Uniquely identify each requirement), the sources of the requirement (Guideline 4.4, Record requirements sources} and the rationale for the requirement (Guideline 4.8, Record requirements rationale).

You should provide word processor templates for these forms to all those who are likely to have to collect requirements. Use your standard word processor for this rather than any special-purpose tool. **You can then be sure that the forms can be supported on all computers in your organization.**

However, you should not force analysts to fill in all fields on a form and you should not use special-purpose software which requires the fields in a form to be completed in a particular order. Requirements discovery is a difficult process and different people collect information in different ways. **Imposing a single approach is likely to be resisted**. Always allow for a 'get out' where analysts may simply note the requirement in any way, they wish in some comments field of the form. Predefined forms cannot cover everything and there needs to be some flexibility in your system.

Some readers of requirements simply want a high-level description and do not want to read pages and pages of forms. You should allow for this by including fields on the form which may be extracted (automatically if possible) to create a shorter (perhaps incomplete) description of the requirement. Make sure, however, that you maintain links from this shorter description to the complete requirements specification in the standard template. This will then allow you to generate a new short description when changes to the requirement are agreed.

Costs and problems:

If you do not already have standards for presenting requirements, there is a significant **cost in introducing this guideline and developing an appropriate standard**

for your organization. Although you can use the suggestions here as a basis for your standard, you need to look at your existing requirements to check how many different styles (and hence standards) that you use and to decide what information you need to include in a requirements description.

After designing an initial set of standards you will have to assess them and, almost certainly, modify them after you have experience of their use. If you already have standards, you may wish to think about whether you should modify them to incorporate some of the advice given in this book. Once a set of standard templates has been established, there are few additional costs involved in applying the guideline.

A problem which you may face is resistance from **requirements engineers to standards.** They may have evolved their own way of working and do not wish to be constrained by a standard way of recording requirements. You should try to avoid this problem by disseminating information about the standard during its development and by involving engineers in the standard development process. They will then feel committed to the standard and therefore more likely to use it.

B. Use Language Simply, Consistently and Concisely.

When you are writing detailed system requirements, you should define a set of standard templates which you use for organizing the description of these requirements. The template should include fields to collect the detailed information necessary to completely specify the requirement.

Benefits:

- When requirements are written using **simple language, they are easier to read and understand.** The principal costs of requirements engineering are the costs of paying people to read requirements. If you can reduce the time required for requirements reading without losing understanding, you will have an immediate cost saving in your requirements process.
- More people can understand requirements that are written in a simple way. It takes less time to explain the requirements to stakeholders and a wider range of people can participate in the validation of the requirements.

Implementation:

Many people find it difficult to write clear and concise natural language. They write long and convoluted sentences and express the requirements in an unnecessarily verbose way. To improve requirements descriptions, we recommend that you should write a **short style guide which describes how to use language in writing requirements in your organization.**

This guide must be short and easy to read. Otherwise, people will not have the time or the inclination to read it. Ideally, your style guide should include four or five pages of guidelines with examples of good and bad writing drawn from your own experience.

Some basic writing style guidelines which might be included in your guide are as follows.

1. **Keep sentences short.** Because of the limitation of our short-term memory, you often need to read long sentences more than once before you can understand them.
2. **Never express more than one requirement in a single sentence.** Avoid the use of 'and' in sentences. This implies that more than one requirement or concept is being discussed.
3. Avoid the use of **jargon, abbreviations and acronyms** unless you are completely confident that they will be understood by all readers of your document. Even then be very careful; the same acronym may have different meanings in different domains. For example, the acronym **'ATM'** is used in both banking systems and networking application domains. In a requirements document which describes how ATMs (Automatic Teller Machines) should be connected using ATM (Asynchronous Transfer Mode), you can imagine the confusion which could arise.
4. **Keep paragraphs short.** As a general rule, no paragraph should be made up of more than **seven sentences**. Again, the reason for this is due to the limits on people's short-term memory.
5. Use **lists and tables** wherever possible to present information sequences. Lists (like this one) are much easier to understand than sequences which are presented as a single paragraph.
6. **Use terminology consistently.** Do not use a term to mean one thing in one place in the document and something different somewhere else. This is very difficult to achieve, especially when different people are responsible for writing different parts of the document. Using a data dictionary to define the names of system entities can be helpful here (see Guideline 7.5, Use a data dictionary).
7. Use words such as 'shall', 'should', 'will' and 'must' in a consistent way with the following meanings:
 - **'shall'** indicates that the requirement is **mandatory**,
 - **'should'** indicates that the requirement is **desirable but not mandatory**
 - **'will'** indicates something that will be **externally provided**
 - **'must'** is best avoided. If used, it should be a synonym for 'shall'.
8. Do not express requirements using **nested conditional clauses** (i.e. if X then if Y then R1a else if Z then R1b else R1c}. These are very easy to misunderstand. If you cannot find a convenient way of expressing a requirement in natural language without nested conditional clauses, use a different notation such as a decision table.
9. Use the **active rather than the passive voice**, particularly when describing actions taken by people or the system. Unfortunately, most technical writing is written impersonally and many people find it difficult to change to a more personal style.
10. Do not try to express complex relationships in a natural language description. **Diagrams** are much more effective for this purpose.
11. **Never use anonymous references.** When you refer to other requirements, tables or diagrams, give a brief description of what you are referring to, as well as the

reference number. If you simply use a reference number, the reader may need to look it up specially to remind themselves what it means.

12. **Pay attention to spelling and grammar.** Poor spelling and grammar can obscure your meaning. A spelling checker should always be used. The style guide should be included as part of your standard for specifying requirements. When you organize requirements reviews, you should ask reviewers to comment on the writing style used and to highlight requirements which are particularly difficult to understand.

Costs and problems:

The principal costs of introducing this guideline are the **writing of a style guide** and, if necessary, **short training sessions** for all involved in the RE process. These should introduce them to the style guide and clear writing style. There is also a slightly increased review cost but this is minimal so long as your style guide is not too long.

Initially, you will probably find that it is quite expensive to introduce this guideline into your requirements engineering process. Surprisingly, perhaps, it takes longer to write requirements simply than in a more complex way. However, as people gain experience with clear writing style, these costs will rapidly decline.

The problem that you are likely to face is that many people from a technical background find it very difficult to write clearly and concisely. Introducing a style guide is all very well but we have a **long tradition of writing requirements in an opaque and obscure way and this cannot be changed overnight**. Although reviews can highlight significant deviations from the style guides, this will not remove all obscure writing.

C. Use Diagrams Appropriately

You should use diagrams in a requirements description wherever it is necessary to show **structural information or to show relationships between information in the requirements description**. Diagrams may also be used to summarize numeric information and to describe sequences of events or activities.

Benefits:

- Diagrams are far more effective than text for presenting relationships between items of information. Most people find diagrams **easier to understand** than textual descriptions of structure or relationships.
- Diagrams break up sections of text into **smaller, more readable fragments**. This means that you get a double bonus from the diagram. Information is presented more comprehensibly and the surrounding text is easier to understand.
- Diagrams from a requirements document may be reused when making requirements presentations to customers.

Implementation:

As part of your document style guide, we recommend including a section which describes the circumstances where diagrams are most likely to be helpful. These include the following.

1. Where something (a system, a document, etc.) is made up of a number of **modules or components and you wish to illustrate relationships between these components**. An example of this is where you use a block diagram to illustrate the overall structure of a system.
2. Where you need to show a **set of activities where each activity has a number of inputs and outputs**. Diagrams can be used to show the activity sequence and where activities can be carried out in parallel.
3. Where you need to illustrate spatial organization, for example, of a control panel.
4. Where you wish to show some decomposition structure such as an organization chart.

A good general rule for drawing diagrams is to keep them **as simple as possible**. Complex diagrams are often worse than no diagram at all. You should use simple labelled boxes and lines to show structure and relationships. Diagrams should fit onto a single page and should not have 'connectors' to other diagrams. Diagrams should always be labelled with a figure number and a title. The figure number should always be used to refer to the diagram.

You should avoid the use of icons which do not have an obvious meaning. Many people find the commercial 'clip art' pictures provided with word processing packages to be trite and irritating. Unless these are particularly appropriate, these are also best avoided.

If color or shading is used, it should be used to distinguish different parts of the diagram or for emphasis. **Color should be used sparingly. You should not associate meaning with specific colors as people from different backgrounds may interpret color in different ways.** For example, red may mean 'hot' in some application domains and 'danger' in other domains. You should also remember that about 10% of men are color-blind.

Some people think that diagrams are always easier to understand than textual presentation. They therefore introduce diagrams everywhere even when the same information could be presented more clearly using a list or a table. You should not introduce diagrams unless they provide meaningful information about structure or relationships. They should not be over-used or used to present information which could be more clearly expressed in some other way.

Costs and problems:

The costs of introducing this guideline are minimal. Diagrams can be drawn using facilities which are now embedded in most word-processing systems. More complex drawing packages can be used but these are designed for professional illustrators and sometimes encourage the production of over-complex diagrams. There may be some training costs for people to use the drawing facilities but most packages are intuitive and can be self-taught with little effort. The costs of applying the guideline are the costs of drawing diagrams. These should not usually take much more time than textual requirements description.

While you are unlikely to encounter any user resistance to this guideline, you will probably find that some people are much better than others at expressing their ideas in diagrams. This seems to reflect fundamental differences in thinking and there is not much you can do about this. Some people think in pictures, but people who do not think this way often prefer textual descriptions to diagrams.

D. Supplement Natural Languages with Other Descriptions of Requirements.

Some requirements are best described using specialized notations, such as mathematical formulae, decision tables, design notations or programming languages, rather than natural language. Although you will almost always want to have some description in natural language, you should supplement this, where appropriate, with more precise descriptions in an appropriate notation.

Benefits:

- Specialized notations allow for descriptions whose meaning is less likely to be misinterpreted than a natural language description. The description is often more concise than a corresponding natural language description.
- If the notation used is familiar to application domain experts, they are less likely to make mistakes when specifying their requirements and more likely to find any errors which have been made.

Implementation:

Customers and systems engineers have different needs. Customers need to be able to understand the requirements for the system. They need to make an assessment of how this will contribute to their overall organizational objectives, fit into existing working processes and integrate with other systems which are used. Systems engineers need a detailed and unambiguous statement of requirements on which their system design should be based. It should be possible to trace from the requirements to the design through to the system implementation.

It is often hard to meet these conflicting objectives if you only use natural language. You should therefore supplement this with other notations such as the following.

- **Decision tables** where actions have to be taken depending on a complex set of conditions. Text which includes many conditional clauses is often ambiguous. It is notoriously difficult to read and understand.
- **Programming languages or program design languages** which may be used where you wish to describe a number of actions which must take place in sequence and where some of these actions are conditional or may be repeated. You may also use language descriptions for detailed interface specification.
- **Algebra** when you wish to describe how particular numeric values must be transformed by an operation or particular values should be computed.
- **Data-flow diagrams** where you need to define a sequence of transformations which take place during some end-to-end processing of data.
- **Timing diagrams** to illustrate time-critical system actions.
- **System models such as object models, entity-relation** models or finite state models where you need to describe detailed system requirements.

The additional information provided by these notations is usually necessary when the requirements must define detailed information such as a sequence of events which must take place, the interfaces between sub-systems, etc. They are therefore most likely to be used when your requirements document is a detailed functional specification of the system to be implemented.

We do not know of any book where all these notations are described but many of them are covered in Software Engineering, 5th edition by Ian Sommerville (Addison Wesley, 1996).

Costs and problems:

In most cases, introducing this guideline does not require engineers to learn new notations but to use techniques which they already know. In such cases, there are no introductory costs for this guideline. **We advise against introducing new unfamiliar notations as these are likely to cause more problems than they solve.** There are obviously some costs in applying the guideline as tables, models, etc., have to be created. Like diagrams, however, these should not normally cost much more than writing the equivalent technical description.

A possible problem may arise if mathematical or unfamiliar notations such as decision tables are used. Some readers of the requirements may think that a requirement is harder to understand because they do not know the notation. They may therefore oppose the use of these notations. This is a particular problem when the people against the idea of specialized notation are managers who can decide whether or not they should be used. To overcome this resistance, you may find it helpful to **demonstrate that the more concise notations can be paraphrased in natural language.**

For further contents on the topic please refer a book entitled REQUIREMENTS ENGINEERING A Good Practice Guide by Ian Sommerville and Pete Sawyer Lancaster University