5.2 Stating Requirements

Specification Notations

Once requirements are generated, they must be stated. The most widely used notation for recording software requirements specifications is natural language—plain English. The greatest advantage of stating requirements in natural language is that everyone understands it. There is also excellent tool support for it—a word processor. The greatest disadvantage of natural language is that it is not precise.

For example, suppose a requirement states that "Operation A will occur and operation B will occur, or operation C will occur." This specification leaves open the following questions:

- Must operation A occur before operation B, or does their order not matter?
- Must operation C occur only if neither A nor B occurs, or should C occur if either one of A or B does not occur?
- May operation C occur even if both A and B occur?

This sort of imprecision propagated over an entire SRS provides an enormous source of misunderstanding and error.

Another drawback of natural language is that it is not effective for specifying complicated relationships, especially spatial and temporal relationships. For example, it is very difficult to describe screen layouts in English and awkward to detail all the steps in a process. Graphical notations are better than natural language for such tasks.

Alternatives to natural language include a large collection of semi-formal and formal notations:

Semi-Formal Notations—These notations are more precise and concise than natural language, but they are not defined with mathematical rigor and precision. These include most graphical notations used in this book, such as class diagrams, tables, and pictures. Semi-formal notations avoid much of the imprecision of natural language, are better at describing complex relationships, and are usually fairly easy for most people to understand, especially when accompanied by notes and explanations.

Formal Notations—These notations are defined with mathematical rigor and precision. Examples of such notations are mathematical and logical notations, such as set notations and first-order logic; some graphical notations, such as state diagrams; and notations invented especially for specification, such as Z. Formal notations completely avoid the imprecision of natural language, but they are difficult to learn and use and inappropriate for communication with most stakeholders.

Use of formal notations is growing among software professionals, and they may eventually become the standard for stating requirements for use by professional software developers. For now, however, the standard remains

a mixture of natural language and semi-formal graphical notations with an occasional equation or two.

Stating Requirements

Requirements specifications are read and consulted over and over again, so they must be written and laid out on the page as clearly and concisely as possible. They should follow the rules of good technical writing:

Write complete, simple sentences in the active voice.

Define terms clearly and use them consistently.

Use the same word for a particular concept—that is, avoid synonyms.

Group related material into sections.

Provide a table of contents and perhaps an index.

Use tables, lists, indentation, white space, and other formatting aids to present and organize material clearly and concisely.

Leave margins ragged on the right, and use a kerned, medium-sized font.

In addition to rules of good writing and layout, there are a few heuristics especially for writing software requirements, which we consider in the next few subsections.

"Must" and "Shall"

Recall that a software product requirement is a statement that a software product *must* have a certain feature, function, capability, or property. It is therefore conventional to state all requirements using the words "must" or "shall." We can state this as a heuristic:

Express all requirements using the words "must" or "shall."

For example, a requirement should be expressed in one of the following ways:

The product must display all results to three decimal places.

The product shall display all results to three decimal places.

It is not correct to express such a requirement using other auxiliary verbs or no auxiliary verb at all, as in the following examples:

* The product will display all results to three decimal places.

* The product should display all results to three decimal places.

* The product displays all results to three decimal places.

Here and below, the * indicates that these are not acceptable expressions of requirements.

Verifiable Specifications

A specification is **verifiable** if there is a definitive procedure to determine whether it is met. Sometimes this property is referred to as a requirement's **testability**. The following statements are examples of requirements that are not verifiable.

* The product must produce reports in an acceptable amount of time.

* The product must respond to human users quickly.

* The product user interface must be user-friendly.

* The product must control several drill presses.

In each case, there is no way to tell decisively whether the requirement is satisfied. The following examples show how the previous requirements might be made verifiable:

The product must produce reports in five minutes or less from the time the report is requested.

The product must respond to human users in one second or less.

Eighty percent of first-time users must be able to formulate and enter a simple query within two minutes of starting to use the program.

The product must control up to seven drill presses concurrently.

Verifiability is crucial for all stakeholders. Verifiable requirements are specific and hence capture more exactly what the product must do to satisfy a client's needs and desires. They also give engineering designers, implementers, testers, and other developers the details they need to do their jobs. Hence, an important requirements writing heuristic is

Write verifiable requirements.

Requirements Atomization

Requirements must be realized through engineering design and implementation, and products must be reviewed and tested to ensure that their requirements are met. The ability to track requirements from their expression in an SRS to their realization in engineering design documentation, source code, and user documentation and their verification in reviews and tests is called **requirements traceability**. Requirements traceability is a fundamental software quality assurance capability.

Requirements traceability depends on being able to isolate and identify individual requirements. Isolating individual requirements makes it possible to associate parts of a design, some code, a test case, and so forth with just the relevant portions of an SRS. Associating individual requirements with an identifier simplifies making links to relevant requirements. We call requirements statements that meet these traceability needs *atomic.*

> A requirements statement is **atomic** if it states a single product function, feature, characteristic, or property, and it has a unique identifier.
>
> Splitting requirements statements until each one is atomic is called **atomizing requirements.**

Consider the requirements in Figure 5-2-1 from an SRS for a product to help technical support staff track computers and their assignments to employees.

Staff must be able to add computers to the tracking system.
When a computer is added, the tracking system must require the staff member to specify its type and allow the staff member to provide a description. Both these fields must be text of length greater than 0 and less than 512 characters.
The tracking system must respond with a unique serial number required for all further interactions with the tracking system about the added computer.

Figure 5-2-1 A Requirements Specification Example

These paragraphs contain many individual requirements and they are not numbered. Tracing these requirements as stated would be difficult. The atomized version of these requirements appears in Figure 5-2-2.

1. The tracking system must allow staff to add computers to the tracking system.
1.1 When a computer is added to the tracking system, it must require the staff member to provide type data for the added computer.
1.1.1 A computer's type data must be text of length greater than 0 and less than 512 characters.
1.2 When a computer is added to the tracking system, the tracking system must allow the staff member to provide description data for the added computer.
1.2.1 A computer's description data must be text of length greater than 0 and less than 512 characters.
1.3 The tracking system must respond to added computer input with a unique serial number identifying the computer in the tracking system.
1.4 All further interactions between staff members and the tracking system about a computer must use the unique serial number assigned by the tracking system.

Figure 5-2-2 Some Atomized Requirements

The atomized requirements are simple statements of single requirements, each numbered for identification and to show relationships to other requirements. The atomized requirements are much easier to trace through the development process.

Atomization Practices

Atomizing requirements lays the foundation for requirements traceability. Each labeled statement should express a single requirement. Operational- and physical-level atomized requirements will usually be verified by single test cases and implemented by fairly small parts of the design and code. As a rule of thumb, atomic requirements statements are expressed in simple declarative sentences rather than compound sentences, lists, or paragraphs.

Although any scheme for generating unique requirements identifiers will do, a hierarchical numbering scheme works well. Enumerate main requirements with whole numbers, related sub-requirements with a dotted

extension (1.1, 1.2, etc.), related sub-sub-requirements with another dotted extension (1.1.1, 1.1.2, etc.), and so on. This scheme is simple, shows connections between related requirements, and is infinitely expandable.

Non-natural-language specifications, such as equations, tables, trees, and diagrams, should be unchanged but included in the numbering scheme so that they can be cited like any other requirements.

We summarize these considerations in the following atomization heuristic:

Atomize requirements by stating each requirement in a numbered simple declarative sentence or in a numbered equation, tree, table, or diagram.

Heuristics Summary

Figure 5-2-3 summarizes our requirements specification heuristics.

- State requirements using formal or semi-formal notations when possible.
- Write complete, simple sentences in the active voice.
- Define terms clearly and use them consistently.
- Avoid synonyms.
- Group related material into sections.
- Provide a table of contents and perhaps an index.
- Use tables, lists, indentation, white space, and other formatting aids to present and organize material clearly and concisely.
- Leave margins ragged on the right and use a kerned, medium-sized font.
- Express all requirements using the words "must" or "shall."
- Write verifiable requirements.
- Atomize requirements by stating each requirement in a numbered simple declarative sentence or in a numbered equation, tree, table, or diagram.

Figure 5-2-3 Requirements Specification Heuristics

Section Summary

- Most requirements are expressed in plain English, but natural language is vague, ambiguous, and imprecise.
- Semi-formal notations (diagrams, tables, etc.) and formal notations (logic and mathematical notations) are more precise than natural language but harder to read.
- English requirements should be written in clear, precise, and simple technical prose.
- Requirements should be expressed in sentences using the auxiliary verbs "must" or "shall."
- Requirements should be **verifiable.**
- **Requirements traceability** is the ability to track requirements from their statement in an SRS to their realization in design, code, and user documentation and their verification in reviews and tests.