



# M253 Resource Sheet

---

## *Specifying requirements*

---

### 1 Overview

Having spent some time identifying the context and scope of our proposed system and describing its major features, we need to get down to a lower level of detail in terms of our requirements for the system – what we expect our system to do.

We are still concerned with issues of what we are going to deliver rather than how we are going to deliver it. In this Resource Sheet we introduce some of the more important ideas and terminology used when specifying system requirements.

### 2 Requirements analysis

The term ‘software crisis’ first came into public use in 1968 to describe a situation that was causing concern at that time, and which has remained with us in some form ever since. This was the recognition that, as software-based systems became larger and more complex, and were introduced in new application areas, many of those systems were being delivered late, over budget, and failing to satisfy customer’s real needs.

Much work has been done over the intervening years in an attempt to improve this situation. Examples are the development and application of software life cycle models, software project management techniques, and software cost estimation models. However, the most difficult problem remains – that of ensuring software systems are developed which provide the features and facilities that the customers and the users actually expected.

Requirements analysis is the name given to the process of attempting to produce a complete, unambiguous, consistent, precise, verifiable, and implementation-independent description of the proposed system, which is readable, modifiable and well-organised for reference and review. A challenging task indeed!

### 3 Why do we need software requirements?

‘The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is as difficult to rectify later.’

Brooks (1987)

In the Resource Sheet *How to decompose a problem* we briefly introduced some ideas from a paper by Dean Leffingwell (Leffingwell 2000) about user and stakeholder needs, system features and use cases. These were seen as ways of describing the system to be developed in fairly general terms. The 'needs' are an expression of the issues associated with the problem domain, and give us a first approximation to what any satisfactory system would have to achieve. The 'features' fall between the users' real needs and the detailed way in which the system fulfils those needs, and are a first step towards the solution of the problem. They lack fine-grained detail but give a clear, natural language, non-technical statement of what the system is going to do to meet the identified needs. The 'use cases' are a vehicle for describing the ways in which users might interact with the system to accomplish one of the desired features, i.e. how the features are achieved in behavioural terms. Combining all these ways of describing the system produces what Leffingwell terms a 'vision' document for the system. He goes on to point out that this document, although (hope)fully communicating the intent of the system, is not in a form, or at a level of detail, where we would want to put it into the hands of a team of developers and send them off to create the system for us. We need to be a lot more definitive about what we want the system to do, and we need to take on board the needs of a whole new group of stakeholders – developers, testers, etc. So we need another layer of definition about what the system should do – the software requirements.

#### **4 What do we mean by requirements?**

'Requirement' is another everyday word that is overused and under-defined in the context of software engineering. Leffingwell uses a classic, but not very enlightening, definition of a software requirement, taken from an IEEE tutorial volume on requirements engineering.

'A software requirement is a software capability needed by the user to solve a problem that will achieve an objective, or, a software capability that must be met or possessed by a system or system component to satisfy a contract, standard, or other formally imposed documentation.'

Leffingwell (2000) p.6

Leffingwell suggests that we use this definition as an indication that we should develop our more specific set of requirements by refining or elaborating our list of features. Each requirement serves some feature. Most features will probably lead to several related requirements. Note that we do not introduce any new requirements that are not linked to already-identified features, but that as we write our requirements it may be that they suggest new or revised features and associated requirements. Like most things in life, the process is iterative; no matter how hard we try to think of everything up front, some things will not be immediately apparent.

The generally agreed purpose of the requirements phase of the software development life cycle is to determine exactly what we want the system to do, but still to avoid the temptation to specify how it will be done, which is left to the later design phase. This attempt to separate the 'what' from the 'how' – to separate the understanding and specification of the problem from the understanding of the solution – is not as easy in practice as it sounds in theory, and the boundaries are often blurred.

#### **5 Categories of requirements**

Most writers on software requirements suggest that there are three major categories into which requirements can be subdivided in order to describe different aspects of the proposed system. These are functional requirements, non-functional requirements and design constraints. To these we can add a fourth category, which is becoming increasingly important in modern interactive systems, namely user interface requirements. As with so many other areas of software development there is a blurring or overlapping of the boundaries between these categories, both in terms of their definition and their use. However, they can provide us with a reasonable initial framework for talking about and reasoning about a system.

## 5.1 Functional requirements

These are intended to capture the behavioural aspects of the system – what the system actually does. They describe what the inputs are, what the outputs are, what transforms are necessary in order to convert specified inputs to specified outputs in specified circumstances. They are generally our major area of interest in describing information systems.

As a simple example we might consider that, in a system which is to provide us with online facilities for booking flights, ‘the user should be able to search for available flights based on the date and time of flights, and range of ticket prices, and should be able to specify whether they only want direct flights or are prepared to take flights with intermediate changes of aircraft’. In a library information system ‘the user should be able to search for books based on title, author or ISBN’.

## 5.2 Non-functional requirements

These are intended to capture additional system ‘attributes’ such as performance requirements, throughput, usability, security and reliability. They are aspects of the overall quality of the system, and they may well place restrictions on both the product being developed and the process by which it is developed.

A simple example of a non-functional requirement might be a system security requirement that ‘all data shall be protected from unauthorised access’. It is easy to see that if we push this down to a lower level we could well find ourselves moving across into the area of functional requirements, since we may well demand that users can only access certain data areas on entry of a specified username and password, or that different data areas will be accessible to different groups of people depending on some internally checked authorisation status. In the process of defining and then decomposing fairly general non-functional requirements we may therefore generate new instances of more specific functional requirements.

## 5.3 Design constraints

Instead of defining the way in which the system behaves, this category of requirements is needed to cover restrictions on the design of a system that need to be fulfilled in order to meet external technical, business or contractual obligations.

A simple example might be a requirement that ‘the system has to be written in JavaScript and to work in a correct and consistent manner with (specified versions) of the Microsoft Internet Explorer and the Netscape Navigator browsers’. Another possibility may be that the system is to be designed for access by blind users and that there is a requirement ‘that the information on the user interface screens must be readable by a particular screen-reading tool’.

In systems where users have to provide personal data there may be a need to consider the privacy implications from the point of view of any relevant Data Protection legislation.

A further category of constraint may be imposed by the need for the proposed system to interact with other systems that already exist and which, for example, require data to be made available in particular formats if the new system needs to access the facilities provided by these existing systems.

## 5.4 User interface requirements

Interactive systems, i.e. systems that involve a significant amount of user interaction, bring a number of more specific concerns to the process of requirements specification. These include issues about the nature of the user-interface itself, and the modelling of the potential users of the system. There may be different classes of user, with conflicting expectations of how the system should present its features to them, in terms of the level of sophistication of the interactions required, or their degree of assumed knowledge about the information presented.

## 6 A hierarchy of (functional) requirements

In many cases there is a natural hierarchical relationship between the requirements that we have identified, and we can organise them in successive levels of abstraction in the way that we describe the overall system. This is particularly true for functional requirements. We can partition our system, i.e. decompose it into its constituent parts, in two directions: horizontally and vertically. A complex area of identified functionality may be decomposed horizontally into several simpler areas of functionality. Each one of these separate areas may itself be further decomposed vertically by describing it in more detail, so that eventually a tree-like structure of requirements emerges.

## 7 How do we document requirements?

The requirements document is an official statement of the system requirements for customers, end-users and software developers. The name this document is given differs between organisations, and it may be called variously a requirements definition, a functional specification or a software requirements specification. Its purpose is more important than its name, and it should be seen as the agreed final product of the analysis of requirements activity, forming the basis for all further system development. It should be equally meaningful to the customers, the project managers, the software designers and coders, and the system testers and maintainers. It acts as a source of reference and possibly even as the basis for any development contract, since it describes the common understanding of exactly what is to be built.

The form of the requirements document is also to some extent a matter of choice, and will vary between organisations, depending on the size and complexity of the system to be described. A good overall view of what a requirements document should contain is provided in the classic IEEE/ANSI 830-1993 standard (IEEE 1993) which suggests the following contents and structure:

- Introduction
  - Purpose of the requirements document
  - Scope of the product
  - Glossary (Definitions of technical terms, acronyms, abbreviations, etc)
  - References
  - Overview of the remainder of the document
- General description
  - Product perspective
  - Product functions
  - User characteristics
  - General constraints
  - Assumptions and dependencies
- Specific requirements
  - Functional requirements
  - Non-functional requirements
  - Interface requirements
  - Logical database requirements
  - etc
- Appendices
- Index

An alternative publicly available and downloadable requirements document format is the Volere Requirements Specification Template (see the *Further resources* section of this document) which has been produced and refined by The Atlantic Systems Guild. This follows the general approach of the IEEE standard, but is adapted to suit a particular requirements methodology. It is worth looking at, as it has a useful discussion of each major section under the appropriate sub-headings of Content, Motivation, Examples and Considerations, as well as a good set of definitions of the terms used in the template.

## 8 Prioritising requirements

When the whole system has been analysed and all possible requirements have been described and documented we may well need to stand back from what we have produced and let reality step in. There may not be enough time, money or even capability available for us to produce a system with all the features that have been identified, and some means of prioritising requirements is needed.

Dai Clegg of Oracle UK, one of the early participants in the Dynamic Systems Development Methodology (DSDM) consortium, came up with a useful acronym in this context, for prioritising requirements at the design stage. This is the **MoSCoW** rule, where M stands for Must have, S for Should have, C for Could have, and W for Won't have (or, in some versions of the rule, Want to have but...). We can expand the meaning of each of these headings as:

Must have applies to those requirements that are fundamental to the system, without which it would effectively be inoperable.

Should have applies to those requirements which would certainly be mandatory in the system if it was not under any time or money pressure, but without which the system would still be usable and useful.

Could have applies to requirements that would be valuable but are only to be included if time and money allows them to be.

Won't have (or Want to have, but not this time around) applies to other requirements identified as potentially valuable enhancements to the system at some later date.

As a team we can use ranking and cumulative voting techniques to categorise and label all the requirements we have identified and to ensure that an agreed set of requirements that **must** be provided exists at the end of the requirements analysis phase of a project.

## 9 Summary

This document is not intended to provide a full coverage of the area of systems requirements. Its intention is to give a general indication, for the purposes of this project, of some of the main issues involved, some pointers to the different types of requirement that need to be considered when attempting to define what a system should do, and some pointers to ways in which the results of analysing the identified requirements might be organised and presented. The area is broad enough to be the subject of complete courses, and there are many books specifically devoted to it, such as the volume by Kotonya and Sommerville (1998).

## 10 Further resources

The IEEE/ANSI standard 830-1993 referred to above in Section 7, as well as the IEEE/ANSI standard 830-1998 which superseded it, are available to subscribers to the IEEE Standards Online website. As a registered OU student you have access to this service through the OU Electronic Library facility.

The Volere Requirements Specification Template can be accessed online at:

[www.systemsguild.com/GuildSite/Robs/Template.html](http://www.systemsguild.com/GuildSite/Robs/Template.html)

A more general paper by Paul R. Read Jr., entitled "Rational unified process: 10 essential elements", includes a discussion of issues such as the MoScoW rules. It can be accessed at:

[www.jacksonreed.com/public/speaking/rupwnotes.pdf](http://www.jacksonreed.com/public/speaking/rupwnotes.pdf)

## 11 References

- Brooks, F. (1987) 'No silver bullet: Essence and accidents of software engineering.' *IEEE Computer* **20** (4), April 1987, pp.10–19.
- IEEE (1993) 'IEEE recommended practice for software requirements specifications'. In: R. H. Thayer and M. Dorfman (eds) *Software Requirements Engineering*. Chapter 5. Los Alamitos, CA, IEEE Computer Society Press.
- Kotonya, G. and Sommerville, I. (1998) *Requirements Engineering: Processes and Techniques*. New York, Wiley.
- Leffingwell, D, (2000) 'Features, use cases, requirements, oh my!' Rational Software White Paper.