Bron:
Leffingwell, D. e.a. (2003). Managing Software Requirements. 2de ed.
Boston: Pearson Education. ISBN 9780321122476

Chapter 2

Introduction to Requirements Management

Key Points

- A requirement is a capability that is imposed on the system.
- Requirements management is a process of systematically eliciting, organizing, and documenting requirements for a complex system.
- Our challenge is to understand *users'* problems in *their* culture and *their* language and to build systems that meet *their* needs.
- A feature is a service that the system provides to fulfill one or more stakeholder needs.
- A use case describes a sequence of actions, performed by a system, that yields a result of value to a user.

Based on the data presented in Chapter 1, you can see why we're interested in focusing on requirements management. However, before we can begin explaining the various techniques and strategies, we need to provide some definitions and examples. We'll start by defining what we mean by the term *software requirement.*

Definitions

What Is a Software Requirement?

Although many definitions of software requirements have been used throughout the years, the one provided by requirements engineering authors Dorfman and Thayer [1990] is quite workable:

1. *A software capability needed by the user to solve a problem to achieve an objective*
2. *A software capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documentation*

This definition may appear to be a little vague, but later we'll develop these concepts further. For now, this definition will do quite well.

What Is Requirements Management?

Requirements define capabilities that the systems must deliver, and conformance (or lack of conformance) to a set of requirements often determines the success (or failure) of projects. It makes sense, therefore, to find out what the requirements are, write them down, organize them, and track them in the event that they change. Stated another way, we'll define requirements management as

> *a systematic approach to eliciting, organizing, and documenting the requirements of the system, and a process that establishes and maintains agreement between the customer and the project team on the changing requirements of the system.*

Let's take a closer look at some key concepts contained in this definition.

- Anyone who has ever been involved with complex software systems—whether from the perspective of a customer or a developer—knows that a crucial skill is the ability to *elicit* the requirements from users and stakeholders.
- Since hundreds, if not thousands, of requirements are likely to be associated with a system, it's important to *organize* them.
- Since most of us can't keep more than a few dozen pieces of information in our heads, *documenting* the requirements is necessary to support effective communication among the various stakeholders. The requirements have to be recorded in an accessible medium: a document, a model, a database, or a list on the whiteboard.

What do these elements have to do with managing requirements? Project size and complexity are major factors here: nobody would bother talking about "managing" requirements in a two-person project that had only 10 requirements to fulfill. But to verify 1,000 requirements—a small purchased software product—or 300,000 requirements—a Boeing 777—it's obvious that we will face problems of organizing, prioritizing, controlling access to, and providing resources for the various requirements. On even a modest-size project, questions will naturally arise.

- Which project team members are responsible for the wind speed requirement (#278), and which ones are allowed to modify it or delete it?

- If requirement #278 is modified, what other requirements will be affected?
- How can we be sure that someone has written the code in a software system to fulfill requirement #278, and which test cases in the overall test suite are intended to verify that the requirements have indeed been fulfilled?

That, along with some other, similar activities, is what requirements management is all about.

This is not something new that we've invented on our own; it's one of those "commonsense" activities that most development organizations claim to do in some fashion or other. It's typically informal and carried out inconsistently from one project to the next, and some of the key activities are likely to be overlooked or short-changed because of the pressures and politics associated with many development projects. So, requirements management could be regarded as a set of organized, standardized, and systematic processes and techniques for dealing with the requirements of a significant, complex project.

We're certainly not the first to suggest the idea of organized, formalized processes; two well-known efforts of this kind are the Software Engineering Institute's Capability Maturity Model (SEI-CMM) and the ISO 9000 quality management standards. (Requirements practices in these processes are described briefly in Appendix F.)

APPLICATION OF REQUIREMENTS MANAGEMENT TECHNIQUES

Types of Software Applications

We suggest that software applications can be categorized as follows:

- Information systems and other applications developed for use within a company (such as the payroll system being used to calculate the take-home pay for our next paycheck). This category is the basis for the information system/information technology industry, or IS/IT.
- Software developed and sold as commercial products (such as the word processor we are using to write this chapter). Companies developing this type of software are often referred to as independent software vendors, or ISVs.

- Software that runs on computers embedded in other devices, machines, or complex systems (such as those contained in the airplane we are writing this in; the cell phones we just used to call our spouses; the automobile we'll use to get to our eventual destination). We'll call this type of software embedded-systems applications, or embedded applications.

The characteristics of the applications we develop for these three different types of systems are extremely diverse. They could consist of 5,000,000 lines of COBOL on a mainframe host environment developed over a period of ten or more years by fifty to a hundred individuals. They could consist of 10,000 lines of Java on a Web server application written in one year by a one- or two-person team. Or they could be 1,000,000 lines of extremely time-critical C code on a complex real-time telephony system.

We'll maintain that the requirements management techniques presented throughout this book can be applied to any of these types of systems. Many of the techniques are independent of application type; others may need to be tuned for the application-specific context before being applied. To enhance your understanding, we'll provide a mix of examples to illustrate the application of the various techniques.

Systems Applications

Requirements management can also be applied to systems development. Most of the techniques in this book will deliver value in managing requirements of arbitrarily complex systems consisting of mechanical subsystems, computer subsystems, and chemical subsystems and their interrelated pieces and parts. Clearly, this is a broad discipline, and we will have to show some discretion to be able to deliver value to the average software team member. Therefore, we'll focus on a requirements management process and specific techniques that can be applied most directly to significant software applications of the IS/IT, ISV, or embedded-systems types.

The Road Map

Since we are embarking on a journey to develop quality software—on time and on budget—that meets customers' real needs, it may well be helpful to have a map of the territory. This is a difficult challenge in that the variety of people you encounter on this particular journey, and the languages they speak, are quite diverse. Many questions will arise.

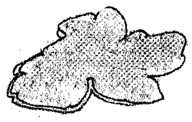
- Is this a need or a requirement?
- Is this a nice-to-have or a must-have?
- Is this a statement of the problem or a statement of the solution?
- Is this a goal of the system or a contractual requirement?
- Do we have to program in Java? Says who?
- Who doesn't like the new system, and where was that person when we visited here before?

In order to navigate successfully through the territory, we'll need to understand where we are at any point in time, whom we are meeting, what language they are speaking, and what information we need from them to complete our journey successfully. Let's start in the "land of the problem."

The Problem Domain

Most successful requirements journeys begin with a trip to the land of the problem. This *problem domain* is the home of real users and other stakeholders, people whose needs must be addressed in order for us to build the perfect system. This is the home of the people who need the rock or a new sales order entry system or a configuration management system good enough to blow the competition away. In all probability, these people are not like us. Their technical and economic backgrounds are different from ours, they speak in funny acronyms, they go to different parties and drink different beers, they don't wear T-shirts to work, and they have motivations that seem strange and unfathomable. (What, you never liked *Star Trek?*)

On rare occasions, they are just like us. They are programmers looking for a new tool or system developers who have asked you to develop a portion of the system. In these rare cases, this portion of the journey *might* be easier, but it might also be more difficult.


Problem domain

Typically, this is not the case, and we are in the land of the alien user. These users have *business* or *technical problems* that they need our help to solve. Therefore, it becomes *our* problem to understand *their* problems, in *their* culture and *their* language, and to build systems that meet *their* needs. Since this territory can seem a little foggy, we'll represent it as a cloud. This will be a constant reminder to us to make sure we are seeing all the issues in the problem space clearly.

Within the problem domain, we use a set of *team skills* as our map and compass to *understand the problem to be solved.* While we are here, we need to gain an understanding of the problem and the needs that must be filled to address this problem.

Stakeholder Needs

It is also our responsibility to understand the needs of users and other stakeholders whose lives will be affected by our solution. As we elicit those needs, we'll stack them in a little pile called *stakeholder needs*, which we represent as a pyramid.

Moving Toward the Solution Domain

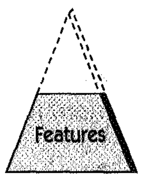
Fortunately, the journey through the problem domain is not necessarily difficult, and the artifacts collected there are not many. However, with even this little bit of data, we will be well provisioned for the part of the journey that we have perhaps been better prepared for: providing a solution to the problem at hand. In this solution space, we focus on defining a solution to the user's problem; this is the realm of computers, programming, operating systems, networks, and processing nodes. Here, we can apply the skills we have learned much more directly.

Features of the System

First, however, it will be helpful to state what we learned in the problem domain and how we intend to resolve those issues via the solution. This is not a very long list and consists of such items as the following.

- "The car will have power windows."
- "Defect-trending charts will provide a visual means of assessing progress."
- "The program will allow Web-enabled entry of sales orders."
- "An automatic step-and-repeat weld cycle is required."

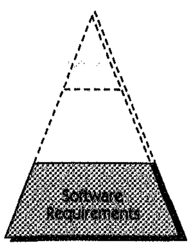
These are simple descriptions, in the user's language, that we will use as labels to communicate with the user how our system addresses the problem. These labels will become part of our everyday language, and much energy will be spent in defining them, debating them, and prioritizing them. We'll call these descriptions *features* of the system to be built and will define a feature as



a service provided by the system that fulfills one or more stakeholder needs.

Graphically, we'll represent features as a base for the previous needs pyramid.

Software Requirements



Once we have established the feature set and have gained agreement with the customer, we can move on to defining the more specific requirements we will need to impose on the solution. If we build a system that conforms to those requirements, we can be certain that the system we develop will deliver the features we promised. In turn, since the features address one or more stakeholder needs, we will have addressed those needs directly in the solution.

These more specific requirements are the *software requirements.* We'll represent them as a block within our pyramid in a manner similar to the features. We also note that these appear pretty far down on the pyramid, and this implies, correctly, that we have much work to do before we get to that level of specificity later in the book.

SUMMARY

Now let's take a look at the map we've built. In Figure 2–1, you can see that we have made a subtle yet important transition in our thinking in this process. We have moved from the problem domain, represented by the cloud and the

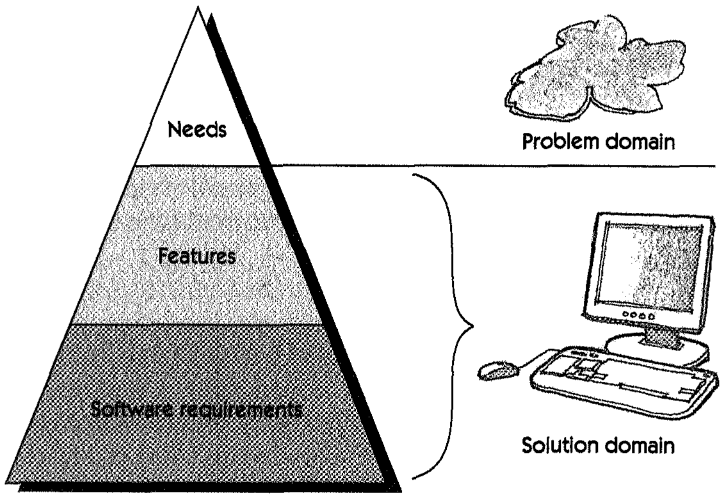


Figure 2–1 Overview of the problem domain and the solution domain

user needs we discovered, to a definition of a system that will constitute the solution domain, represented by the features of the system and the software requirements that will drive its design and implementation. Moreover, we have done so in a logical, stepwise fashion, making sure to understand the problem and the user's needs before we envision or define the solution. This road map, along with its significant distinctions, will continue to be important throughout this book.