Chapter 7

Identifying needs and establishing requirements

7.1 Introduction

An interaction design project may aim to replace or update an established system, or it may aim to develop a totally innovative product with no obvious precedent. There may be an initial set of requirements, or the project may have to begin by producing a set of requirements from scratch. Whatever the initial situation and whatever the aim of the project, the users' needs, requirements, aspirations, and expectations have to be discussed, refined, clarified, and probably re-scoped. This requires an understanding of, among other things, the users and their capabilities, their current tasks and goals, the conditions under which the product will be used, and constraints on the product's performance.

As we discussed in Chapter 6, identifying users' needs is not as straightforward as it sounds. Establishing requirements is also not simply writing a wish list of features. Given the iterative nature of interaction design, isolating requirements activities from design activities and from evaluation activities is a little artificial, since in practice they are all intertwined: some design will take place while requirements are being established, and the design will evolve through a series of evaluation–redesign cycles. However, each of these activities can be distinguished by its own emphasis and its own techniques.

This chapter provides a more detailed overview of identifying needs and establishing requirements. We introduce different kinds of requirements and explain some useful techniques.

The main aims of this chapter are to:

- Describe different kinds of requirements.
- Enable you to identify examples of different kinds of requirements from a simple description.
- Explain how different data-gathering techniques may be used, and enable you to choose among them for a simple description.
- Enable you to develop a "scenario," a "use case," and an "essential use case" from a simple description.
- Enable you to perform hierarchical task analysis on a simple description.

7.2 What, how, and why?

7.2.1 What are we trying to achieve in this design activiiy?

There are two aims. One aim is to understand as much as possible about the users, their work, and the context of that work, so that the system under development can support them in achieving their goals; this we call "identifying needs." Building on this, our second aim is to produce, from the needs identified, a set of stable requirements that form a sound basis to move forward into thinking about design. This is not necessarily a major document nor a set of rigid prescriptions, but you need to be sure that it will not change radically in the time it takes to do some design and get feedback on the ideas. Because the end goal is to produce this set of requirements, we shall sometimes refer to this as the requirements activity.

7.2.2 How can we achieve this?

The whole chapter is devoted to explaining how to achieve these aims, but first we give an overview of where we're heading.

At the beginning of the requirements activity, we know that we have a lot to find out and to clarify. At the end of the activity we will have a set of stable requirements that can be moved forward into the design activity. In the middle, there are activities concerned with gathering data, interpreting or analyzing[1] the data, and

---

'We use *interpretation* to mean the initial investigation of the data, while *analysis* is a more detailed study, using a particular frame of reference and notation.

capturing the findings in a form that can be expressed as requirements. Broadly speaking, these activities progress in a sequential manner: first gather some data, then interpret it, then extract some requirements from it, but it gets a lot messier than this, and the activities influence one another as the process iterates. One of the reasons for this is that once you start to analyze data, you may find that you need to gather some more data to clarify or confirm some ideas you have. Another reason is that the way in which you document your requirements may affect your analysis, since it will enable you to identify and express some aspects more easily than others. For example, using a notation which emphasizes the data-flow characteristics of a situation will lead the analysis to focus on this aspect rather than, for example, on data structure. Analysis requires some kind of framework, theory or hypothesis to provide a frame of reference, however informal, and this will inevitably affect the requirements you extract. To overcome this, it is important to use a complementary set of data-gathering techniques and data-interpretation techniques, and to constantly revise and refine the requirements. As we discuss below, there are different kinds of requirements, and each can be emphasized or de-emphasized by the different techniques.

Identifying needs and establishing requirements is itself an iterative activity in which the subactivities inform and refine one another. It does not last for a set number of weeks or months and then finish. In practice, requirements evolve and develop as the stakeholders interact with designs and see what is possible and how certain facilities can help them. And as shown in the lifecycle model in Chapter 6, the activity itself will be repeatedly revisited.

7.2.3 Why bother? The importance of getting it right

An article published in January 2000 (Taylor, 2000) investigated the causes of IT project failure. The article admits that "there is no single cause of IT project failure," but requirements issues figured highly in the findings. The research involved detailed questioning of 38 IT professionals in the UK. When asked about which project stages caused failure, respondents mentioned "requirements definition" more than any other phase. When asked about cause of failure, "unclear objectives and requirements" was mentioned more than anything else, and for critical success factors, "clear, detailed requirements" was mentioned most often.

As stressed in previous chapters, understanding what the product under development should do and ensuring that it supports stakeholders' needs are critically important activities in any product development. If the requirements are wrong then the product will at best be ignored and at worst be despised by the users, and will cause grief and lost productivity. In either case, the implications for both producer and customer are serious: anxiety and frustration, lost revenue, loss of customer confidence, and so on. However we look at it, getting the requirements of the product wrong is a very bad move and something to be avoided at all costs.

Taking a user-centered approach to development is one way to address this. If users' voices and needs are clearly heard and taken into account, then it is more likely that the end result will meet users' needs and expectations. Involving users isn't always easy, however, and we explore in more detail how to do this effectively

in Chapter 9. Here we focus on establishing the requirements, while keeping the emphasis clearly on users' needs.

7.2.4 Why *establish* requirements?

The activity of understanding what a product should do has been given various labels—for example, requirements gathering, requirements capture, requirements elicitation, requirements analysis, and requirements engineering. The first two imply that requirements exist out there and we simply need to pick them up or catch them. "Elicitation" implies that "others" (presumably the clients or users) know the requirements and we have to get them to tell us. Requirements, however, are not that easy to identify. You might argue that, in some cases, customers must know what the requirements are because they know the tasks that need to be performed, and may have asked for a system to be built in the first place. However, they may not have articulated requirements as yet, and even if they have an initial set of requirements, they probably have not explored them in sufficient detail for development to begin.

The term "requirements analysis" is normally used to describe the activity of investigating and analyzing an initial set of requirements that have been gathered, elicited, or captured. Analyzing the information gathered is an important step, since it is this interpretation of the facts, rather than the facts themselves, that inspires the design. Requirements engineering is a better term than the others because it recognizes that developing a set of requirements is an iterative process of evolution and negotiation, and one that needs to be carefully managed and controlled.

We chose the term establishing requirements to represent the fact that requirements arise from the data-gathering and interpretation activities and have been established from a sound understanding of the users' needs. This also implies that requirements can be justified by and related back to the data collected.

7.3 What are requirements?

Before we go any further, we need to explain what we mean by a requirement. Intuitively, you probably have some understanding of what a requirement is, but we should be clear. A requirement is a statement about an intended product that specifies what it should do or how it should perform. One of the aims of the requirements activity is to make the requirements as specific, unambiguous, and clear as possible. For example, a requirement for a website might be that the time to download any complete page is less than 5 seconds. Another less precise example might be that teenage girls should find the site appealing. In the case of this latter example, further investigation would be necessary to explore exactly what teenage girls would find appealing. Requirements come in many different forms and at many different levels of abstraction, but we need to make sure that the requirements are as clear as possible and that we understand how to tell when they have been fulfilled. The example requirement shown in Figure 7.1 is expressed using a template from the Volere process (Robertson and Robertson, 1999), which you'll hear more about later in this chapter and in Suzanne Robertson's interview at the end of this

Requirement #: 75 Requirement Type: 9 Eventluse case #: 6

Description: The product shall issue an alert if a weather station fails to transmit readings.

Rationale: Failure to transmit readings might indicate that the weather station is faulty and needs maintenance, and that the data used to predict freezing roads may be incomplete.

Source: Road Engineers
Fit Criterion: For each weather station the product shall communicate to the user when the recorded number of each type of reading per hour is not within the manufacturer's specified range of the expected number of readings per hour.

Customer Satisfaction: 3 Customer Dissatisfaction: 5
Dependencies: None Conflicts: None
Supporting Materials: Specification of Rosa Weather Station
History: Raised by GBS, 28 July 99

Volere

Figure 7.1 An example requirement using the Volere template.*

chapter. This template requires quite a bit of information about the requirement itself, including something called a "fit criterion," which is a way of measuring when the solution meets the requirement. In Chapter 6 we emphasized the need to establish specific usability criteria for a product early on in development, and this part of the template encourages this.

7.3.1 Different kinds of requirements

In software engineering, two different kinds of requirements have traditionally been identified: functional requirements, which say what the system should do, and non-functional requirements, which say what constraints there are on the system and its development. For example, a functional requirement for a word processor may be that it should support a variety of formatting styles. This requirement might then be decomposed into more specific requirements detailing the kind of formatting required such as formatting by paragraph, by character, and by document, down to a very specific level such as that character formatting must include 20 typefaces, each with bold, italic, and standard options. A non-functional requirement for a word processor might be that it must be able to run on a variety of platforms such as PCs, Macs and Unix machines. Another might be that it must be able to function on a computer with 64 MB RAM. A different kind of non-functional requirement would be that it must be delivered in six months' time. This represents a constraint on the development activity itself rather than on the product being developed.

If we consider interaction devices in general, other kinds of non-functional requirements become relevant such as physical size, weight, color, and production

*See Figure 7.5 for an explanation of these fields.

feasibility. For example, when the PalmPilot was developed (Bergman and Haitani, 2000), an overriding requirement was that it should be physically as small as possible, allowing for the fact that it needed to incorporate batteries and an LCD display. In addition, there were extremely tight constraints on the size of the screen, and that had implications for the number of pixels available to display information. For example, formatting lines or certain typefaces may become infeasible to use if they take up even one extra pixel. Figure 7.2 shows two screen shots from the PalmPilot development. As you can see, removing the line at the left-hand side of the display in the top window released sufficient pixels to display the missing "s" in the bottom window.

Interaction design requires us to understand the functionality required and the constraints under which the product must operate or be developed. However, instead of referring to all requirements that are not functional as simply "non-functional" requirements, we prefer to refine this into further categories. The following is not an exhaustive list of the different requirements we need to be looking out for (see the figure in Suzanne Robertson's interview at the end of this chapter for a more detailed list), nor is it a tight categorization, however, it does illustrate the variety of requirements that need to be captured.

Functional requirements capture what the product should do. For example, a functional requirement for a smart fridge might be that it should be able to tell when the butter tray is empty. Understanding the functional requirements for an interactive product is very important.

Data requirements capture the type, volatility, size/amount, persistence, accuracy, and value of the amounts of the required data. All interactive devices have to handle greater or lesser amounts of data. For example, if the system under consid-

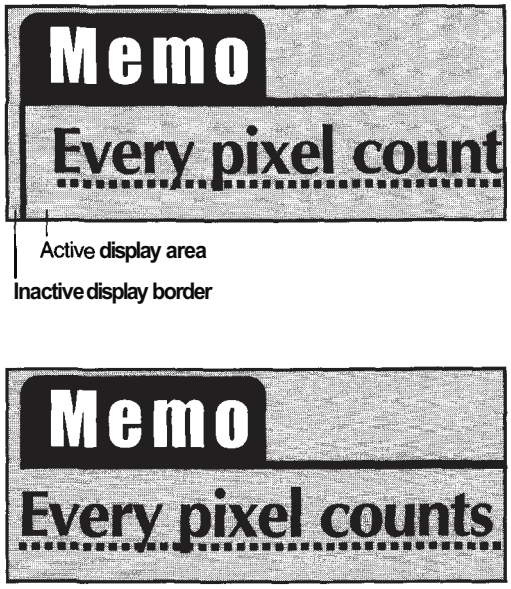


Figure 7.2 **Every pixel** counts.

eration is to operate in the share-dealing application domain, then the data must be up-to-date and accurate, and is likely to change many times a day. In the personal banking domain, data must be accurate, must persist over many months and probably years, is very valuable, and there is likely to be a lot of it.

Environmental requirements or *context of use* refer to the circumstances in which the interactive product will be expected to operate. Four aspects of the environment must be considered when establishing requirements. First is the physical environment such as how much lighting, noise, and dust is expected in the operational environment. Will users need to wear protective clothing, such as large gloves or headgear, that might affect the choice of interaction paradigm? How crowded is the environment? For example, an ATM operates in a very public physical environment. Using speech to interact with the customer is therefore likely to be problematic.

The second aspect of the environment is the social environment. The issues raised in Chapter 4 regarding the social aspects of interaction design, such as collaboration and coordination, need to be explored in the context of the current development. For example, will data need to be shared? If so, does the sharing have to be synchronous, e.g., does everyone need to be viewing the data at once, or asynchronous, e.g., two people authoring a report take turns in editing and adding to it? Other factors include the physical location of fellow team members, e.g., do collaborators have to communicate across great distances?

The third aspect is the organizational environment, e.g., how good is user support likely to be, how easily can it be obtained, and are there facilities or resources for training? How efficient or stable is the communications infrastructure? How hierarchical is the management? and so on.

Finally, the technical environment will need to be established: for example, what technologies will the product run on or need to be compatible with, and what technological limitations might be relevant?

User requirements capture the characteristics of the intended user group. In Chapter 6 we mentioned the relevance of a user's abilities and skills, and these are an important aspect of user requirements. But in addition to these, a user may be a novice, an expert, a casual, or a frequent user. This affects the ways in which interaction is designed. For example, a novice user will require step-by-step instructions, probably with prompting, and a constrained interaction backed up with clear information. An expert, on the other hand, will require a flexible interaction with more wide-ranging powers of control. If the user is a frequent user, then it would be important to provide short cuts such as function keys rather than expecting them to type long commands or to have to navigate through a menu structure. A casual or infrequent user, rather like a novice, will require clear instructions and easily understood prompts and commands, such as a series of menus. The collection of attributes for a "typical user" is called a *user profile.* Any one device may have a number of different user profiles.

Note that user requirements are not the same as usability requirements. We discuss the latter below.

Usability requirements capture the usability goals and associated measures for a particular product. In Chapter 6 we introduced the idea of usability engineering,

BOX 7.1 Underwater PCs

Developing a PC for undersea divers to take underwater has one major environmental factor: it's surrounded by water! However, waterproofing is not the main issue for the designers at WetPC, a company who have produced such a system. The interface has proved to be more of a problem. Divers typically have only one hand free to operate the computer, and are likely to be swimming and moving up and down in the water at the same time. So a traditional interface design is no good. Early prototypes of the computer used voice recognition, but the bubbles made too much noise and distorted the sound. Tracker balls were also inappropriate because the divers are not working on a flat surface. So the main developer at WetPC, Bruce Macdonald, devised a "keyboard" called a KordGrip that has five keys (see Figure 7.3(a)). Combinations of keys represent different symbols, so that divers can choose items from menus. They can perform operations such as controlling a camera and sending messages. The system is also linked to a GPS system that tells the divers where they are. This makes it much easier to mark the location of mines and other underwater discoveries (See Figure 7.3(b) on Color Plate 8).

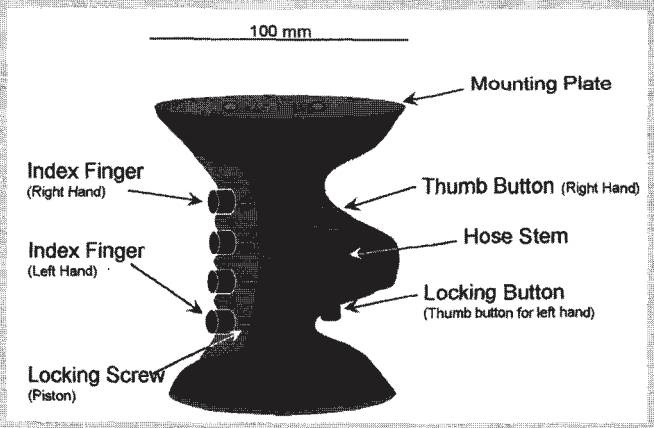


Figure 7.3 (a) The KordGrip interface; (b) the KordGrip in use under water.

an approach in which specific measures for the usability goals of the product are established and agreed upon early in the development process and are then revisited, and used to track progress as development proceeds. This both ensures that usability is given due priority and facilitates progress tracking. In Chapter 1 we described a number of usability goals: effectiveness, efficiency, safety, utility, learnability, and memorability. If we are to follow the philosophy of usability engineering and meet these usability goals, then we must identify the appropriate requirements. Chapter 1 also described some user experience goals, such as making products that are fun, enjoyable, pleasurable, aesthetically pleasing, and motivating. As we observed in Chapter 6, it is harder to identify quantifiable measures that allow us to track these qualities, but an understanding of how important each of these is to the current development should emerge as we learn more about the intended product.

Usability requirements are related to other kinds of requirement we must establish, such as the kinds of users expected to interact with the product.

ACTIVITY 7.1 Suggest one key functional, data, environmental, user and usability requirement for each of the following scenarios:

(a) A system for use in a university's self-service cafeteria that allows users to pay for their food using a credit system.

(b) A system to control the functioning of a nuclear power plant.

(c) A system to support distributed design teams, e.g., for car design.

Comment You may have come up with alternative suggestions; these are indicative of the kinds of answer we might expect.

(a) ***Functional:*** The system will calculate the total cost of purchases.

Data: The system must have access to the price of products in the cafeteria.

Environmental: Cafeteria users will be carrying a tray and will most likely be in a reasonable rush. The physical environment will be noisy and busy, and users may be talking with friends and colleagues while using the system.

User: The majority of users are likely to be under 25 and comfortable dealing with technology.

Usability: The system needs to be simple so that new users can use the system immediately, and memorable for more frequent users. Users won't want to wait around for the system to finish processing, so it needs to be efficient and to be able to deal easily with user errors.

(b) ***Functional:*** The system will be able to monitor the temperature of the reactors.

Data: The system will need access to temperature readings.

Environmental: The physical environment is likely to be uncluttered and to impose few restrictions on the console itself unless there is a need to wear protective clothing (depending on where the console is to be located).

User: The user is likely to be a well-trained engineer or scientist who is competent to handle technology.

Usability: Outputs from the system, especially warning signals and gauges, must be clear and unambiguous.

(c) ***Functional:*** The system will be able to communicate information between remote sites.

Data: The system must have access to design information that will be captured in a common file format (such as AutoCAD).

Environmental: Physically distributed over a wide area. Files and other electronic media need to be shared. The system must comply with available communication protocols and be compatible with network technologies.

User: Professional designers, who may be worried about technology but who are likely to be prepared to spend time learning a system that will help them perform their jobs better. The design team is likely to be multi-lingual.

Usability: Keeping transmission error rate low is likely to be of high priority.

7.4 Data gathering

So how do we go about determining requirements? Data gathering is an important part of the requirements activity and also of evaluation. In this chapter, we concentrate on data gathering for the requirements activity. Further information about the techniques we present here and how to apply them in evaluation is in Chapters 12 through 14.

The purpose of data gathering is to collect sufficient, relevant, and appropriate data so that a set of stable requirements can be produced. Even if a set of initial requirements exists, data gathering will be required to expand, clarify, and confirm those initial requirements. Data gathering needs to cover a wide spectrum of issues because the different kinds of requirement we need to establish are quite varied, as we saw above. We need to find out about the tasks that users currently perform and their associated goals, the context in which the tasks are performed, and the rationale for why things are the way they are.

There is essentially a small number of basic techniques for data gathering, but they are flexible and can be combined and extended in many ways; this makes the possibilities for data gathering very varied, to give full leverage on understanding the variety of requirements we seek. These techniques are questionnaires, interviews, focus groups and workshops, naturalistic observation, and studying documentation. Some of them, such as the interview, require active participation from stakeholders, while others, such as studying documentation, require no involvement at all. In addition, various props can be used in data-gathering sessions, such as descriptions of common tasks and prototypes of possible new functionality. See Section 7.6 and Chapter 8 for further information on how to develop these props. Box 7.2 gives an

BOX 7.2 Combining Data-Gathering Techniques and Props to Understand Different Requirements

Rudman and Engelbeck (1996) describe how they used different techniques to establish the requirements for a complex graphical user interface for a telephone company, and how different methods resulted in understanding different requirements. They used five different techniques:

1. On-site observation allowed them to understand the nature of the current business.
2. Participatory prototyping, i.e., active involvement of the stakeholders in designing a prototype, allowed them to take advantage of the employees' knowledge.
3. Interviews aimed at understanding the background business of the company allowed them to understand the complex nature of the wider domain.
4. Interviews aimed at understanding the decision sequences of employees allowed them to create dialogs to support two-party negotiations.
5. Role-playing prototype walkthroughs using simulated scenarios also helped to create dialogs to support two-party negotiations.

The difference between the third and fourth techniques lies in the focus of the questioning and in the notation used to capture data. In the third technique, interviewers focused on understanding the domain of the application and captured information using semantic nets, which are specifically designed to represent such information. In the fourth technique, decision trees were used to understand the goals, decision points, and options considered by employees when dealing with a customer.

example of how different methods and props can be combined to gain maximum advantage, while Box 7.3 describes a very different approach aimed at prompting inspiration rather than simple data gathering.

7.4.1 Data-gathering techniques

In addition to the most common forms of data-gathering techniques listed above, if a system is currently operational then data logging may be used. This involves instrumenting the software to record users' activity in a log that can be examined later. Each of the techniques will yield different kinds of data and are useful in different circumstances. In most cases, they are also used in evaluation, and how to implement them is described in Chapters 12 and 13. Here we describe what each technique involves and explain the circumstances for which they are most suitable, in the context of the requirements activity. The discussion is summarized in Table 7.1 on page 214.

Questionnaires. Most of us are familiar with questionnaires. They are a series of questions designed to elicit specific information from us. The questions may require different kinds of answers: some require a simple YES/NO, others ask us to choose from a set of pre-supplied answers, and others ask for a longer response or comment. Sometimes questionnaires are sent in electronic form and arrive via email or are posted on a website, and sometimes they are given to us on paper. In most cases the questionnaire is administered at a distance, i.e., no one is there to help you answer the questions or to explain what they mean.

Well-designed questionnaires are good at getting answers to specific questions from a large group of people, and especially if that group of people is spread across a wide geographical area, making it infeasible to visit them all. Questionnaires are often used in conjunction with other techniques. For example, information obtained through interviews might be corroborated by sending a questionnaire to a wider group of stakeholders to confirm the conclusions.

Interviews. Interviews involve asking someone a set of questions. Often interviews are face-to-face, but they don't have to be. Companies spend large amounts of money conducting telephone interviews with their customers finding out what they like or don't like about their service. If interviewed in their own work or home setting, people may find it easier to talk about their activities by showing the interviewer what they do and what systems and other artifacts they use. The context can also trigger them to remember certain things, for example a problem they have downloading email, which they would not have recalled had the interview taken place elsewhere.

Interviews can be broadly classified as structured, unstructured or semi-structured, depending on how rigorously the interviewer sticks to a prepared set of questions.

In the requirements activity, interviews are good at getting people to explore issues and unstructured interviews are often used early on to elicit scenarios (see Section 7.6 below). Interacting with a human rather than a sterile, impersonal piece of paper or electronic questionnaire encourages people to respond, and can make the exercise more pleasurable. In the context of establishing requirements, it is equally important for development team members to meet stakeholders and for users to feel involved. This on its own may be sufficient motivation to arrange interviews.

BOX 7.3 An Artist-designer Approach to Users

An alternative approach to understanding users and their needs was taken in a European Union-funded project called the Presence Project (Gaver et al., 1999). This work arose from research looking at novel interaction techniques to increase the presence of elderly people in their local community. Three different groups were studied: one in Oslo, Norway, one near Amsterdam, The Netherlands, and one near Pisa, Italy. One of the problems with designing for an unknown culture is that it can be difficult to understand or appreciate the needs of that culture. Rather than take a more traditional approach of questionnaires, interviews or ethnographic studies, this project used "cultural probes." These probes consisted of a wallet containing a variety of items: 8 to 10 postcards, about seven maps, a disposable camera, a photo album, and a media diary (see Figure 7.4). The intent was for the recipients to look through the wallet and answer questions associated with certain probes that they contained, then to return the items directly to the researchers when they had finished with them.

Figure 7.4 A cultural probe package.

The postcards had pictures on the front and questions on the back, and were pre-addressed and stamped so that they could be easily returned. Questions included "Please tell us a piece of advice or insight that has been important to you," "What place does art have in your life?" and "Tell us about your favorite device." The maps and associated inquiries were designed to find out about the participants' attitudes towards their environment. They were printed on various textured papers and were in the form of folding envelopes, also to facilitate their return. On local maps, participants were asked to mark sites where they would go to meet people, to be alone, to daydream and where they would like to go, but can't. On a map of the world, they were asked to mark places where they had been.

Participants were asked to use the camera to take pictures of their home, what they will wear today (whenever "today" was), the first person you see today, something desirable, and something boring. In the photo album they were asked to tell the researchers their story in pictures. The media diary was to record their use of television and radio.

The approach taken by these researchers was not to identify specific user needs but to seek inspiration that would lead to new opportunities, new pleasures, new forms of sociability, and new cultural forms. Hence they were seeking inspiration rather than requirements.

The probes were returned over a period of a month or so, at different rates and in different quantities for each group. The data were not analyzed *per se*, but the resulting designs reflect what the designers learned.

For the Dutch site, they proposed building a network of computer displays with which the elderly could help inhabitants communicate their values and attitudes about the culture.

For the Norwegians, they proposed that the elders should lead a community-wide conversation about social issues, publishing questions

from the library that would be sent for public response to electronic systems in cafes, trams, or public spaces.

For the Italian village near Pisa, they plan to create social and pastoral radioscapes allowing them to create flexible communications networks and to listen to the surrounding countryside.

"What we learned about the elders is only half the story, however. The other half is what the elders learned from the probes. They provoked the groups to think about the roles they play and the pleasures they experience, hinting to them that our designs might suggest new roles and new experiences." (Gaver et al., 1999, p. 29)

However, interviews are time consuming and it may not be feasible to visit all the people you'd like to see.

Focus groups and workshops. Interviews tend to be one on one, and elicit only one person's perspective. As an alternative or as corroboration, it can be very revealing to get a group of stakeholders together to discuss issues and requirements. These sessions can be very structured with set topics for discussion, or can be unstructured. In this latter case, a facilitator is required who can keep the discussion on track and can provide the necessary focus or redirection when appropriate. In some development methods, workshops have become very formalized. For example, the workshops used in Joint Application Development (Wood and Silver, 1995) are very structured, and their contents and participants are all prescribed.

In the requirements activity, focus groups and workshops are good at gaining a consensus view and/or highlighting areas of conflict and disagreement. On a social level it also helps for stakeholders to meet designers and each other, and to express their views in public. It is not uncommon for one set of stakeholders to be unaware that their views are different from another's even though they are in the same organization. On the other hand, these sessions need to be structured carefully and the participants need to be chosen carefully. It is easy for one or a few people to dominate discussions, especially if they have control, higher status, or influence over the other participants.

Naturalistic observation. It can be very difficult for humans to explain what they do or to even describe accurately how they achieve a task. So it is very unlikely that a designer will get a full and true story from stakeholders by using any of the techniques listed above. The scenarios and other props used in interviews and workshops will help prompt people to be more accurate in their descriptions, but observation provides a richer view. Observation involves spending some time with the stakeholders as they go about their day-to-day tasks, observing work as it happens, in its natural setting. A member of the design team shadows a stakeholder, making notes, asking questions (but not too many), and observing what is being done in the natural context of the activity. This is an invaluable way to gain insights into the tasks of the stakeholders that can complement other investigations. The level of involvement of the observer in the work being observed is variable along a spectrum with no involvement (outside observation) at one end and full involvement (participant observation) at the other.

Table 7.1 Overview of data-gathering techniques used in the requirements activity

Technique	Good for	Kind of data	Advantages	Disadvantages	Detail for designing in
Questionnaires	Answering specific questions	Quantitative and qualitative data	Can reach many people with low resource	The design is crucial. Response rate may be low. Responses may not be what you want	Chapter 13
Interviews	Exploring issues	Some quantitative but mostly qualitative data	Interviewer can guide interviewee if necessary. Encourages contact between developers and users	Time consuming. Artificial environment may intimidate interviewee	Chapter 13
Focus groups and workshops	Collecting multiple viewpoints	Some quantitative but mostly qualitative data	Highlights areas of consensus and conflict. Encourages contact between developers and users	Possibility of dominant characters	Chapter 13
Naturalistic observation	Understanding context of user activity	Qualitative	Observing actual work gives insights that other techniques can't give	Very time consuming. Huge amounts of data	Chapter 12
Studying documentation	Learning about procedures, regulations and standards	Quantitative	No time commitment from users required	Day-to-day working will differ from documented procedures	N/A

Not only can naturalistic observation help fill in details and nuances that simply did not come out of the other investigations, it also provides context for tasks. Contextualizing the work or behavior that a device is to support provides data that other techniques cannot, and from which we can evolve requirements.

In the requirements activity, observation is good for understanding the nature of the tasks and the context in which they are performed. However, it requires more time and commitment from a member of the design team, and it can result in a huge amount of data.

Studying documentation. Procedures and rules are often written down in manuals and these are a good source of data about the steps involved in an activity and

any regulations governing a task. Such documentation should not be used as the only source, however, as everyday practices may augment them and may have been devised by those concerned to make the procedures work in a practical setting. Taking a user-centered view of development means that we are interested in the everyday practices rather than an idealized account.

Other documentation that might be studied includes diaries or job logs that are written by the stakeholders during the course of their work.

In the requirements activity, studying documentation is good for understanding legislation and getting some background information on the work. It also doesn't involve stakeholder time, which is a limiting factor on the other techniques.

7.4.2 Choosing between techniques

Table 7.1 provides some information to help you choose a set of techniques for a specific project. It tells you the kind of information you can get, e.g., answers to specific questions, and the kind of data it yields, e.g., qualitative or quantitative. It also includes some advantages and disadvantages for each technique. The kind of information you want will probably be determined by where you are in the cycle of iterations. For example, at the beginning of the project you may not have any specific questions that need answering, so it's better to spend time exploring issues through interviews rather than sending out questionnaires. Whether you want qualitative or quantitative data may also be affected by the point in development you have reached, but is also influenced by the kind of analysis you need to do.

The resources available will influence your choice, too. For example, sending out questionnaires nationwide requires sufficient time, money, and people to do a good design, try it out (i.e., pilot it), issue it, collate the results and analyze them. If you only have three weeks and no one on the team has designed a survey before, then this is unlikely to be a success.

Finally, the location and accessibility of the stakeholders need to be considered. It may be attractive to run a workshop for a large group of stakeholders, but if they are spread across a wide geographical area, it is unlikely to be practical.

Olson and Moran (1996) suggest that choosing between data-gathering techniques rests on two issues: the nature of the data gathering technique itself and the task to be studied.

Data-gathering techniques differ in two main respects:

1. The amount of time they take and the level of detail and risk associated with the findings. For example, they claim that a naturalistic observation will take two days of effort and three months of training, while interviews take one day of effort and one month of training (p. 276).
2. The knowledge the analyst must have about basic cognitive processes.

Tasks can be classified along three scales:

1. Is the task a set of sequential steps or is it a rapidly overlapping series of subtasks?

2. Does the task involve high information content with complex visual displays to be interpreted, or low information content where simple signals are sufficient to alert the user?
3. Is the task intended to be performed by a layman without much training or by a practitioner skilled in the task domain?

Box 7.4 summarizes two examples to show how techniques can be chosen using these dimensions.

So, when choosing between techniques for data gathering in the requirements activity, you need to consider the nature of the technique, the knowledge required of the analyst, the nature of the task to be studied, the availability of stakeholders and other resources, and the kind of information you need.

7.4.3 Some basic data-gathering guidelines

Organizing your first data-gathering session may seem daunting, but if you plan the sessions well, and know what your objectives are then this will increase your confidence and make the whole exercise a lot more comfortable. Below we list some data-gathering guidelines to support the requirements activity.

- Focus on identifying the stakeholders' needs. This may be achieved by studying their existing behavior and support tools, or by looking at other products,

BOX 7.4 Coordinated Methods *(Olson and Moran, 1996)*

For a walk-up-and-use system. An ATM is an example of a system with a simple task flow and relatively low information content that is targeted for the layman. Because of the user base, the emphasis will be on the ease with which the user can learn to operate the device. An understanding of the user's mental model may also yield insights, as evidenced by the assignment set at the end of Chapter 3.

To establish the components of the task, simple questionnaires might suffice, supplemented with naturalistic observation, i.e., observing current users at existing machines. The initial design guided by guidelines and checklists could be documented as a storyboard. A mockup of the entire system using a rapid prototyping system such as Visual Basic can be used to observe users' difficulties. After a series of such prototyping sessions, the system could be installed in a friendly site and logging data could be gathered.

For a high-performance system. The example used here is a system to support back-room workers at a bank who are reconciling the machine register with the information written on the back of the deposit slip by the customer. The task requires overlapping activation of physical actions and mental capabilities, is relatively high in information content, and is targeted for a skilled user.

This task is less obvious to the designer and we need to employ several techniques to understand it. If there is an existing system in place, then naturalistic observation and interview can be used. More detailed discovery of the objects, actions, and kinds of thinking can come from using interviews. Task analysis will help to understand the details of the task, and once understood, a series of design and evaluation steps follow, including prototyping, detailed analysis of the visual display and usability tests. The design would then iterate until it meets preset target criteria.

such as a competitor's product or an earlier release of your product under development.

- Involve all the stakeholder groups. It is very important to make sure that you get all the views of the right people. This may seem an obvious comment, but it is easy to overlook certain sections of the stakeholder population if you're not careful. We were told about one case where a large distribution and logistics company reimplemented their software systems and were very careful to involve all the clerical, managerial, and warehouse staff in their development process, but on the day the system went live, the productivity of the operation fell by 50%. On investigation it was found that the bottleneck was not in their own company, but in the suppliers' warehouses that had to interact with the new system. No one had asked them how they worked, and the new system was incompatible with their working routines.
- Involving only one representative from each stakeholder group is not enough, especially if the group is large. Everyone you involve in data gathering will have their own perspective on the situation, the task, their job and how others interact with them. If you only involve one representative stakeholder then you will only get a narrow view.
- Use a combination of data gathering techniques. Each technique will yield a certain kind of information, from a certain perspective. Using different techniques is one way of making sure that you get different perspectives (called triangulation, see Chapter 10), and corroboration of findings. For example, use observation to understand the context of task performance, interviews to target specific user groups, questionnaires to reach a wider population, and focus groups to build a consensus view.
- Support the data-gathering sessions with suitable props, such as task descriptions and prototypes if available. Since the requirements activity is iterative, prototypes or descriptions generated during one session may be reused or revisited in another with the same or a different set of stakeholders. Using props will help to jog people's memories and act as a focus for discussions.
- Run a pilot session if possible to ensure that your data-gathering session is likely to go as planned. This is particularly important for questionnaires where there is no one to help the users with ambiguities or other difficulties, but also applies to interview questions, workshop formats, and props. Any data collected during pilot sessions cannot be treated equally with other data, so don't mix them up. After running the pilot it is likely that some changes will be needed before running the session "for real."
- In an ideal world, you would understand what you are looking for and what kinds of analysis you want to do, and design the data-capture exercise to collect the data you want. However, data gathering is an expensive and time-consuming activity that is often tightly constrained on resources. Sometimes pragmatic constraints mean that you have to make compromises on the ideal

situation, but before you can make sensible compromises, you need to know what you'd ***really*** like.

- How you record the data during a face-to-face data-gathering session is just as important as the technique(s) you use. Video recording, audio recording, and note taking are the main options. Video and audio recording provide the most accurate record of the session, but they can generate huge amounts of data. You also need to decide on practical issues that can have profound effects on the data collected, such as where to position the camera. Note taking can be harder unless this is the person's only role in the session, but note taking always involves an element of interpretation. Taking impartial, accurate notes is difficult but can be improved with practice.

ACTIVITY 7.2 For each of the situations below, consider what kinds of data gathering would be appropriate and how you might use the different techniques introduced above. You should assume that you are at the beginning of the development and that you have sufficient time and resources to use any of the techniques.

(a) You are developing a new software system to support a small accountant's office. There is a system running already with which the users are reasonably happy, but it is looking dated and needs upgrading.

(b) You are looking to develop an innovative device for diabetes sufferers to help them record and monitor their blood sugar levels. There are some products already on the market, but they tend to be large and unwieldy. Many diabetes sufferers rely on manual recording and monitoring methods involving a ritual with a needle, some chemicals, and a written scale.

(c) You are developing a website for a young person's fashion e-commerce site.

Comment

(a) As this is a small office, there are likely to be few stakeholders. Some period of observation is always important to understand the context of the new and the old system. Interviewing the staff rather than giving them questionnaires is likely to be appropriate because there aren't very many of them, and this will yield richer data and give the developers a chance to meet the users. Accountancy is regulated by a variety of laws and it would also pay to look at documentation to understand some of the constraints from this direction. So we would suggest a series of interviews with the main users to understand the positive and negative features of the existing system, a short observation session to understand the context of the system, and a study of documentation surrounding the regulations.

(b) In this case, your user group is spread about, so talking to all of them is infeasible. However, it is important to interview some, possibly at a local diabetic clinic, making sure that you have a representative sample. And you would need to observe the existing manual operation to understand what is required. A further group of stakeholders would be those who use or have used the other products on the market. These stakeholders can be questioned to find out the problems with the existing devices so that the new device can improve on them. A questionnaire sent to a wider group in order to back up the findings from the interviews would be appropriate, as might a focus group where possible.

(c) Again, you are not going to be able to interview all your users. In fact, the user group may not be very well defined. Interviews backed up by questionnaires and focus groups would be appropriate. Also, in this case, identifying similar or competing sites and evaluating them will help provide information for producing an improved product.

The problems of choosing among data-gathering techniques for the requirements activity have been recognized in requirements engineering. For example ACRE (Acquisition REquirements) is a quite extensive set of guidance to help requirements engineers choose between a variety of techniques for data gathering, including interviews and observation. The framework also includes other techniques from software engineering, knowledge engineering, and the social sciences. For more information on this framework, see Maiden and Rugg (1996).

7.5 Data interpretation and analysis

Once the first data-gathering session has been conducted, interpretation and analysis can begin. It's a good idea to start interpretation as soon after the gathering session as possible. The experience will be fresh in the minds of the participants and this can help overcome any bias caused by the recording approach. It is also a good idea to discuss the findings with others to get a variety of perspectives on the data.

The aim of the interpretation is to begin structuring and recording descriptions of requirements. Using a template such as the one suggested in Volere (Figure 7.5) highlights the kinds of information you should be looking for and guides the data interpretation and analysis. Note that many of the entries are concerned with trace-

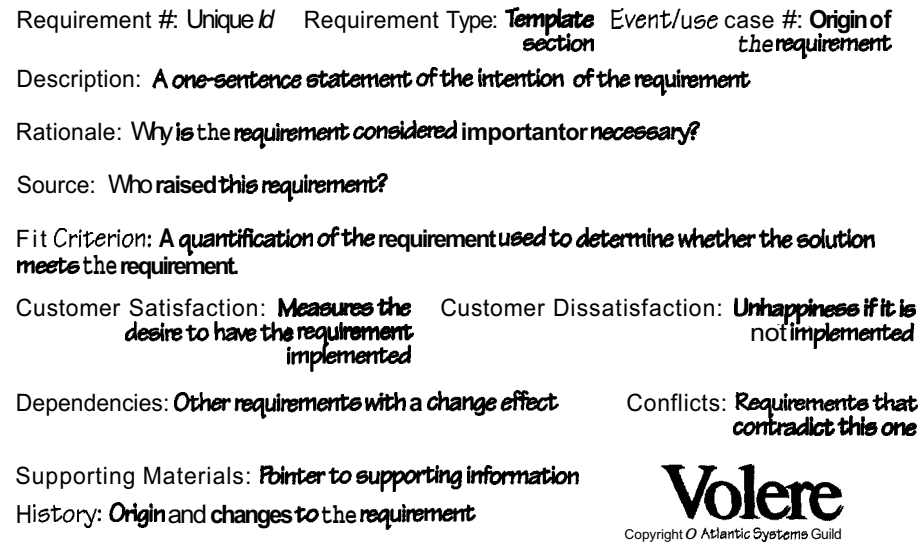
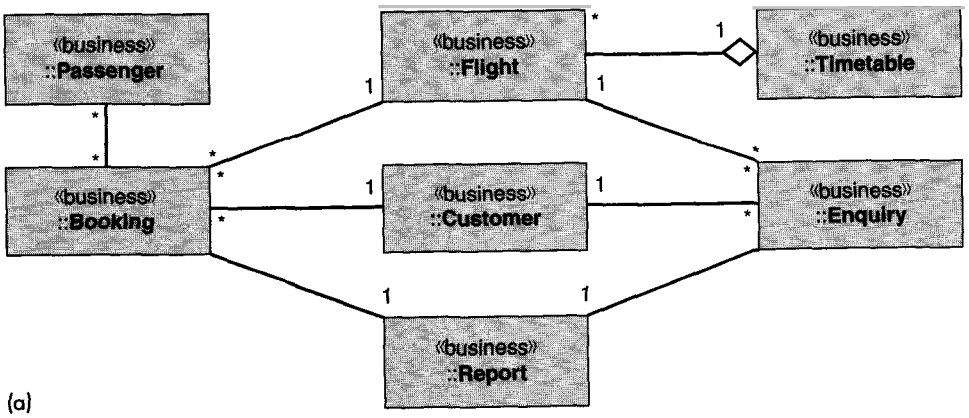


Figure 7.5 The Volere shell for requirements.

ability. For example, who raised the requirement and where can more information about it be found. This information may be captured in documents or in diagrams drawn during analysis. Providing links with raw data as captured on video or audio recordings can be harder, although just as important. Haumer et al. (2000) have developed a tool that records concrete scenarios using video, speech, and graphic media, and relates these recorded observations to elements of a corresponding design. This helps designers to keep track of context and usage information while analyzing and designing for the system.

More focused analysis of the data will follow initial interpretation. Different techniques and notations exist for investigating different aspects of the system that will in turn give rise to the different requirements. For example, functional requirements have traditionally been analyzed and documented using data-flow diagrams,



(a)

(b)

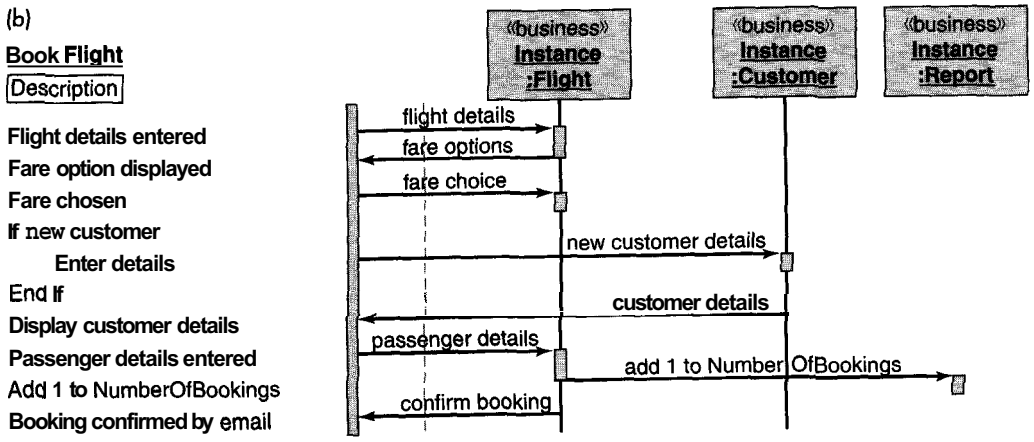
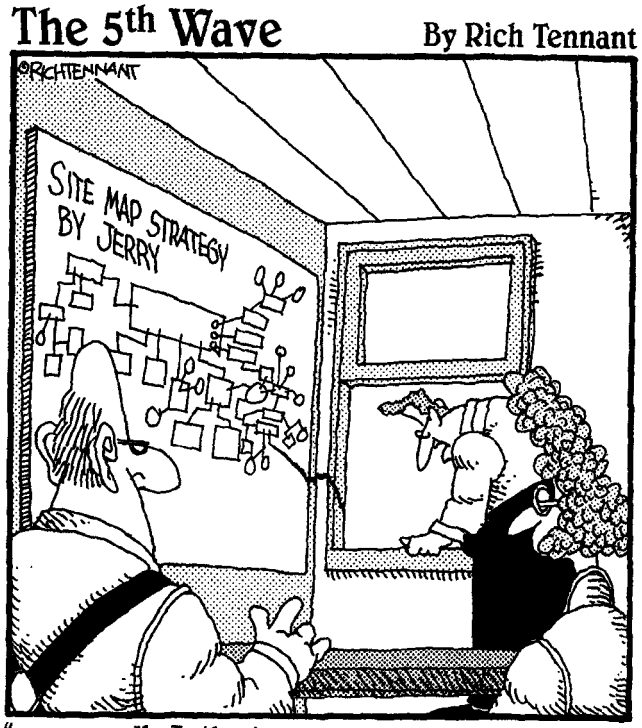


Figure 7.6 (a) Class diagram and (b) sequence diagram that might be used to analyze and capture static structure and dynamic behavior (respectively) if the system is being developed using an object-oriented approach.

state charts, work-flow charts, etc. (see e.g., Sommerville, 2001). Data requirements can be expressed using entity-relationship diagrams, for example. If the development is to take an object-oriented approach, then functional and data requirements are combined in class diagrams, with behavior being expressed in state charts and sequence diagrams, among others. Examples of two such diagrams representing a portion of a holiday booking system are given in Figure 7.6. These diagrams can be linked to the requirements through the "Event/use case" field in the template in Figure 7.5.

We don't go into the detail of how diagrams such as these might be developed, as whole books are dedicated to them. Instead, we describe four techniques that have a user-centered focus and are used to understand users' goals and tasks: scenarios, use cases, essential use cases, and task analysis. **All** of them may be produced during data-gathering sessions, and their output used as props in subsequent data-gathering sessions.

The requirements activity iterates a number of times before a set of stable requirements evolves. As more interpretation and analysis techniques are applied, a deeper understanding of requirements will emerge and the requirements descriptions will expand and clarify.



"Okay, well, I think we **all** get the gist of where Jerry **was** going **with the** site map."

DILEMMA How formal should your notation be?

Many forms of notation are used in design activities. Each discipline has its own set of symbols, graphs, and mnemonics that communicate precisely and clearly among people from the same discipline. But, in the early stages of design, designers are well known for their "back-of-an-envelope" sketches that capture the essence of an idea. At what stage should these sketches be transformed into more formal notations?

When we have identified needs and established requirements, they must be documented somehow. Whether this is in a purely textual form, or in prototypical form, or more formal box and line notations, our findings must be documented. When Verplank (1994) speaks about producing software-based prototypes, he talks emphatically about the importance of allowing ideas to flourish before they are formalized in the computer medium. Once cast in this way, we "get sucked into thinking that the design already works and all we have to do is fix it." The same could be said of formal paper-based notations. In interaction design, we have many notations to choose from, arising from the various disciplines that underpin our field (see Figure 1.3). How quickly should we formalize our ideas in structured notation, and for how long should we leave the ideas fluid and flexible?

A counterargument to Verplank's position is that trying to write our findings in a more structured fashion also helps us to understand better what we've found and what's missing. The problem is that any notation has its strengths and weaknesses, and we must be aware of these when we commit our ideas to a specific notation so that our thinking and our ideas don't become too influenced by the foibles of the notation itself.

Yet again, there is also a question of who the requirements are being documented for. If users are to read and understand them, then the notation shouldn't contain technical jargon or symbols. On the other hand, if it's for communicating precise meaning within a team of developers, a more formal, specialized notation may be more appropriate.

Choosing the medium for the message can affect how the message is received and hence the meaning that is communicated, so it's important to get the medium right.

7.6 Task description

Descriptions of business tasks have been used within software development for many years. During the 1970s and 1980s, "business scenarios" were commonly used as the basis for acceptance testing, i.e., the last testing stage before the customer paid the final fee installment and "accepted" the system. In more recent years, due to the emphasis on involving users earlier in the development lifecycle and the large number of new interaction devices now being developed, task descriptions are used throughout development, from early requirements activities through prototyping, evaluation, and testing. Consequently, more time and effort has been put into understanding how best to structure and use them.

There are different flavors of task descriptions, and we shall introduce three of them here: scenarios, use cases, and essential use cases. Each of these may be used to describe either existing tasks or envisioned tasks with a new device. They are not mutually exclusive and are often used in combination to capture different perspectives or to document different stages during the development lifecycle.

In this section and the next, we use two main examples to illustrate the application of techniques. These are a library catalog service and a shared diary or calendar system. The library catalog is similar to any you might find in a public or

university library, and allows you to access the details of books held in the library: for example, to search for books by a particular author, or by subject, to identify the location of a book you want to borrow, and to check on a member's current loans and status.

The shared calendar application is to support a university department. Members of the department currently keep their own calendars and communicate their whereabouts to the department's administrator, who keeps the information in a central paper calendar. Unfortunately, the central calendar and the individuals' calendars easily become out of step as members of the department arrange their own engagements. It is hoped that having a shared calendar in which individuals can enter their own engagements will help overcome the confusion that often ensues due to this mismatch. Shared calendars raise some interesting aspects of collaboration and coordination, as discussed in Chapter 4, Box 4.2. In particular, people don't usually like to have their time filled with appointments without their consent, and so a mechanism is needed for people to protect some time from being booked by others.

7.6.1 Scenarios

A scenario is an "informal narrative description" (Carroll, 2000). It describes human activities or tasks in a story that allows exploration and discussion of contexts, needs, and requirements. It does not explicitly describe the use of software or other technological support to achieve a task. Using the vocabulary and phrasing of users means that the scenarios can be understood by the stakeholders, and they are able to participate fully in the development process. In fact, the construction of scenarios by stakeholders is often the first step in establishing requirements.

Imagine that you have just been invited along to talk to a group of users who perform data entry for a university admissions office. You walk in, and are greeted by Sandy, the supervisor, who starts by saying something like:

> ***Well, this is where the admissions forms arrive. We receive about 50 a day during the peak application period. Brian here opens the forms and checks that they are complete, that is, that all the documentation has been included. You see, we require copies of relevant school exam results or evidence of work experience before we can process the application. Depending on the result of this initial inspection, the forms getpassed to....***

Telling stories is a natural way for people to explain what they are doing or how to achieve something. It is therefore something that stakeholders can easily relate to. The focus of such stories is also naturally likely to be about what the users are trying to achieve, i.e., their goals. Understanding why people do things as they do and what they are trying to achieve in the process allows us to concentrate on the human activity rather than interaction with technology.

This is not to say that the human activity should be preserved and reflected in any new device we are trying to develop, but understanding what people do now is a good starting point for exploring the constraints, contexts, irritations, facilitators and so on under which the humans operate. It also allows us to identify the stakeholders and the products involved in the activity. Repeated reference to a particular

form, book, behavior, or location indicates that this is somehow central to the activity being performed and that we should take care to understand what it is and the role it plays.

A scenario that might be generated by potential users of a library catalog service is given below:

> *Say I want to find a book by George Jeffries. I don't remember the title but I know it was published before 1995. I go to the catalog and enter my user password. I don't understand why I have to do this, since I can't get into the library to use the catalog without passing through security gates. However, once my password has been confirmed, I am given a choice of searching by author or by date, but not the combination of author and date. I tend to choose the author option because the date search usually identifies too many entries. After about 30 seconds the catalog returns saying that there are no entries for George Jeffries and showing me the list of entries closest to the one I've sought. When I see the list, I realize that in fact I got the author's first name wrong and it's Gregory, not George. I choose the entry I want and the system displays the location to tell me where to find the book.*

In this limited scenario of existing system use, there are some things of note: the importance of getting the author's name right, the annoyance concerning the need to enter a password, the lack of flexible search possibilities, and the usefulness of showing a list of similar entries when an exact match isn't clear. These are all indicators of potential design choices for the new catalog system. The scenario also tells us one (possibly common) use of the catalog system: to search for a book by an author when we don't know the title.

The level of detail present in a scenario varies, and there is no particular guidance about how much or how little should be included. Often scenarios are generated during workshop or interview sessions to help explain or discuss some aspect of the user's goals. They can be used to imagine potential uses of a device as well as to capture existing behavior. They are not intended to capture a full set of requirements, but are a very personalized account, offering only one perspective.

A simple scenario for the shared-calendar system that was elicited in an informal interview describes how one function of the calendar might work: to arrange a meeting between several people.

> *The user types in all the names of the meeting participants together with some constraints such as the length of the meeting, roughly when the meeting needs to take place, and possibly where it needs to take place. The system then checks against the individuals' calendars and the central departmental calendar and presents the user with a series of dates on which everyone is free all at the same time. Then the meeting could be confirmed and written into peoples' calendars. Some people, though, will want to be asked before the calendar entry is made. Perhaps the system could email them automatically and ask that it be conjirmed before it is written in."*

An example of a futuristic scenario, devised by Symbian, showing one vision of how wireless devices might be used in the future is shown in Figure 7.7.

In this chapter, we refer to scenarios only in their role of helping to establish requirements. They have a continuing role in the design process that we shall return to in Chapter 8.

A businesswoman traveling to Paris from the US

A businesswoman is traveling from San Francisco to Paris on a business trip. On her way to the airport she narrowly misses a traffic delay. She avoids the traffic jam because her Srnartphone beeps, then sends her a text message warning her of the traffic accident on her normal route from her office to the airport.

Upon arrival at the airport, the location-sensitive Srnartphone notifies the airline that she will be checking in shortly, and an airline employee immediately finds her and takes her baggage. Her on-screen display shows that her flight is on time and provides a map to her gate. On her way to the gate she downloads tourist information such as maps and events occurring in Paris during her stay.

Once she finds her seat on the plane, she begins to review all the information she has downloaded. She notices than an opera is playing in Paris that she has been wanting to see, and she books her ticket. Her Srnartphone can make the booking using her credit card number, which it has stored in its memory. This means that she does not need to re-enter the credit card number each time she uses wCommerce (i.e., wireless commerce), facilities. The security written into the software of the Smartphone protects her against fraud.

The Srnartphone stores the opera booking along with several emails that she writes on the plane. As soon as she steps off the plane, the Smartphone makes the calls and automatically sends the emails.

As she leaves the airport, a map appears on her Smartphone's display, guiding her to her hotel.

Figure 7.7 A scenario showing how two technologies, a Smartphone and wCommerce (wireless commerce), might be used.

Capturing scenarios of existing behavior and goals helps in determining new scenarios and hence in gathering data useful for establishing the new requirements. The next activity is intended to help you appreciate how a scenario of existing activity can help identify the requirements for a future application to support the same user goal.

ACTIVITY 7.3 **Write a scenario of how you would currently go about choosing a new car. This should be a brand new car, not a second-hand car. Having written it, think about the important aspects of the task, your priorities and preferences. Then imagine a new interactive product that supports you in your goal and takes account of these issues. Write a futuristic scenario showing how this product would support you.**

Comment **The following example is a fairly generic view of this process. Yours will be different, but you may have identified similar concerns and priorities.**

The first thing I would do is to observe cars on the road and identify ones that I like the look of. This may take some weeks. I would also try to identify any consumer reports that will include an assessment of car performance. Hopefully, these initial activities will result in me identifying a likely car to buy. The next stage will be to visit a car showroom and see at first hand what the car looks like, and how comfortable it is to sit in. If I still feel positive about the car, then I'll ask for a test drive. Even a short test drive helps me to

understand how well the car handles, how noisy is the engine, how smooth are the gear changes, and so on. Once I've driven the car myself, I can usually tell whether I would like to own it or not.

From this scenario, it seems that there are broadly two stages involved in the task: researching the different cars available, and gaining first-hand experience of potential purchases. In the former, observing cars on the road and getting actual and maybe critical information about them has been highlighted. In the latter, the test drive seems to be quite significant.

For many people buying a new car, the smell and touch of the car's exterior and interior, and the driving experience itself are often the most influential factors in choosing a particular model. Other more factual attributes such as fuel consumption, amount of room inside, colors available, and price may rule out certain makes and models, but at the end of the day, cars are often chosen according to how easy they are to handle and how comfortable they are inside. This makes the test drive a vital part of the process of choosing a new car.

Taking these comments into account, we've come up with the following scenario describing how a new "one-stop" shop for new cars might operate. This product makes use of immersive virtual reality technology that is already used for other applications such as designing buildings and training bomb disposal experts.

I want to buy a new car, so I go down the street to the local "one-stop car shop." The shop has a number of booths in it, and when I go in I'm directed to an empty booth. Inside there's a large seat that reminds me of a racing car seat, and in front of that a large display screen, keyboard and printer. As I sit down, the display jumps into life. It offers me the options of browsing through video clips of new cars which have been released in the last two years, or of searching through video clips of cars by make, by model, or by year. I can choose as many of these as I like. I also have the option of searching through and reading or printing consumer reports that have been produced about the cars I'm interested in. I spend about an hour looking through materials and deciding that I'd like to experience a couple that look promising. I can of course go away and come back later, but I'd like to have a go with some of those I've found. By flicking a switch in my armrest, I can call up the options for virtual reality simulations for any of the cars I'm interested in. These are really great as they allow me to take the car for a test drive, simulating everything about the driving experience in this car, from road holding, to windscreen display, and front pedal pressure to dash board layout. It even re-creates the atmosphere of being inside the car.

Note that the product includes support for the two research activities mentioned in the original scenario, as well as the important test drive facility. This would be only a first cut scenario which would then be refined through discussion and further investigation.

7.6.2 Use cases

Use cases also focus on user goals, but the emphasis here is on a user–system interaction rather than the user's task itself. They were originally introduced through the object-oriented community in the book *Object-Oriented Software Engineering* (Jacobson et al., 1992). Although their focus is specifically on the interaction between the user (called an "actor") and a software system, the stress is still very much on the user's perspective, not the system's. The term "scenario" is also used in the context of use cases. In this context, it represents one path through the use

case, i.e., one particular set of conditions. This meaning is consistent with the definition given above in that they both represent one specific example of behavior.

A use case is associated with an actor, and it is the actor's goal in using the system that the use case wants to capture. In this technique, the main use case describes what is called the "normal course" through the use case, i.e., the set of actions that the analyst believes to be most commonly performed. So, for example, if through data gathering we have found that most users of the library go to the catalog to check the location of a book before going to the shelves, then the normal course for the use case would include this sequence of events. Other possible sequences, called alternative courses, are then listed at the bottom of the use case.

A use case for arranging a meeting using the shared calendar application, with the normal course being that the meeting is written into the calendar automatically, might be:

1. The user chooses the option to arrange a meeting.
2. The system prompts user for the names of attendees.
3. The user types in a list of names.
4. The system checks that the list is valid.
5. The system prompts the user for meeting constraints.
6. The user types in meeting constraints.
7. The system searches the calendars for a date that satisfies the constraints.
8. The system displays a list of potential dates.
9. The user chooses one of the dates.
10. The system writes the meeting into the calendar.
11. The system emails all the meeting participants informing them of the appointment.

Alternative courses:

5. If the list of people is invalid,
   5.1 The system displays an error message.
   5.2 The system returns to step 2.
8. If no potential dates are found,
   8.1 The system displays a suitable message.
   8.2 The system returns to step 5.

Note that the number associated with the alternative course indicates the step in the normal course that is replaced by this action or set of actions. Also note how specific the use case is about how the user and the system will interact.

Use cases may be described graphically. Figure 7.8 shows the use case diagram for the above calendar example. The actor "Administrator" is associated with the use case "Arrange a meeting." Another actor we might identify for the calendar system is the "Departmental member" who updates his own calendar entries, also shown in Figure 7.8. Actors may be associated with more than one use case, so for

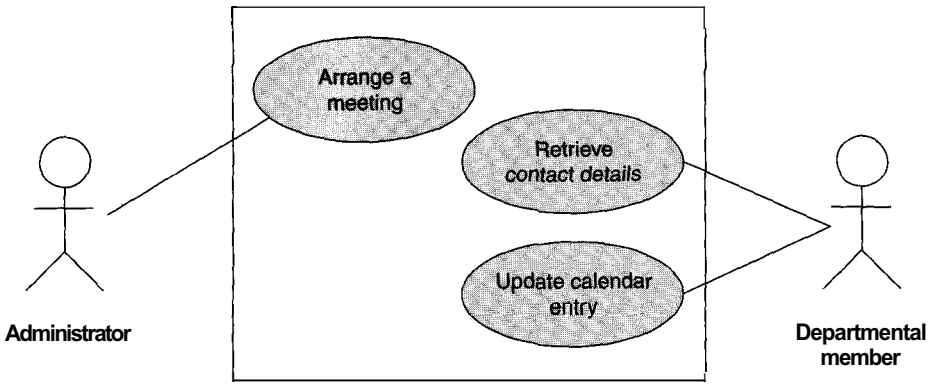


Figure 7.8 Use case diagram for the shared calendar system showing three use cases and two actors.

example the actor "Departmental member" can be associated with a use case "Retrieve contact details" as well as the "Update calendar entry" use case. Each use case may also be associated with more than one actor.

This kind of description has a different style and a different focus from the scenarios described above. The layout is more formal, and the structure of "good" use cases has been discussed by many (e.g., Cockburn, 1995; Gough et al., 1995; Ben Achour, 1999). The description also focuses on the user–system interaction rather than on the user's activities; thus a use case presupposes that technology is being used. This kind of detail is more useful at conceptual design stage than during requirements or data gathering, but use cases have been found to help some stakeholders express their views on how existing systems are used and how a new system might work.

To develop a use case, first identify the actors, i.e., the people or other systems that will be interacting with the system under development. Then examine these actors and identify their goal or goals in using the system. Each of these will be a use case.

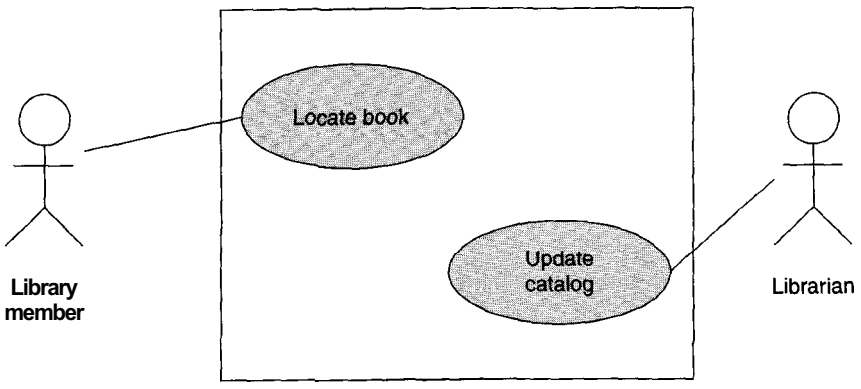


Figure 7.9 Use case diagram for the library catalog service.

ACTIVITY 7.4 Consider the example of the library catalog service again. One use case is "Locate book," and this would be associated with the "Library member" actor. Identify one other main actor and an associated use case, and draw a use case diagram.

Write out the use case for "Locate book" including the normal and some alternative courses. You may assume that the normal course is for users to go to the catalog to find the location, and that the most common path to find this is through a search by author.

Comment

One other main actor is the "Librarian." **A** use case for the "Librarian" would be "Update catalog." Figure 7.9 is the associated use case diagram. There are other use cases you may have identified.

The use case for "Locate book" might be something like this:

1. The system prompts for user name and password.

2. The user enters his or her user name and password into the catalog system.

3. The system verifies the user's password.

4. The system displays a menu of choices.

5. The user chooses the search option.

6. The system displays the search menu.

7. The user chooses to search by author.

8. The system displays the search author screen.

9. The user enters the author's name.

10. The system displays search results.

11. The user chooses the required book.

12. The system displays details of chosen book.

13. The user notes location.

14. The user quits catalog system.

Alternative courses:

4. If user password is not valid
4.1 The system displays error message.
4.2 The system returns to step **1.**

5. If user knows the book details
5.1 The user chooses to enter book details.
5.2 The system displays book details screen.
5.3 The user enters book details.
5.4 The system goes to step **12.**

7.6.3 Essential use cases

Essential use cases were developed by Constantine and Lockwood (1999) to combat what they see as the limitations of both scenarios and use cases as described

arrangeMeeting

USER INTENTION	SYSTEM RESPONSIBILITY
arrange a meeting	
	request meeting attendees and constraints
identify meeting attendees and constraints	
	suggest potential dates
choose preferred date	
	book meeting

Figure 7.10 An essential use case for arranging a meeting in the shared calendar application.

above. Scenarios are concrete stories that concentrate on realistic and specific activities. They therefore can obscure broader issues concerned with the wider organizational view. On the other hand, traditional use cases contain certain assumptions, including the fact that there is a piece of technology to interact with, and also assumptions about the user interface and the kind of interaction to be designed.

Essential use cases represent abstractions from scenarios, i.e., they represent a more general case than a scenario embodies, and try to avoid the assumptions of a traditional use case. An essential use case is a structured narrative consisting of three parts: a name that expresses the overall user intention, a stepped description of user actions, and a stepped description of system responsibility. This division between user and system responsibilities can be very helpful during conceptual design when considering task allocation and system scope, i.e., what the user is responsible for and what the system is to do.

An example essential use case based on the library example given above is shown in Figure 7.10. Note that the steps are more generalized than those in the use case in Section 7.6.2, while they are more structured than the scenario in Section 7.6.1. For example, the first user intention does not say anything about typing in a list of names, it simply states that the user identifies meeting attendees. This could be done by identifying roles, rather than people's names, from an organizational or project chart, or by choosing names from a list of people whose calendars the system keeps, or by typing in the names. The point is that at the time of creating this essential use case, there is no commitment to a particular interaction design.

Instead of actors, essential use cases are associated with user roles. One of the differences is that an actor could be another system, whereas a user role is just that: not a particular person, and not another system, but a role that a number of different people may play when using the system. Just as with actors, though, producing an essential use case begins with identifying user roles.

ACTIVITY 7.5 Construct an essential use case "locateBook" for the user role "Library member" of the library catalog service discussed in Activity 7.4.

Comment

locateBook

USER INTENTION	SYSTEM RESPONSIBILITY
identify self	
	verify identity
	request appropriate details
offer known details	
	offer search results
note search results	
quit system	
	close

Note that here we don't talk about passwords, but merely state that the users need to identify themselves. This could be done using fingerprinting, or retinal scanning, or any other suitable technology. The essential use case does not commit us to technology at this point. Neither does it specify search options or details of how to initiate the search.

7.7 Task analysis

Task analysis is used mainly to investigate an existing situation, not to envision new systems or devices. It is used to analyze the underlying rationale and purpose of what people are doing: what are they trying to achieve, why are they trying to achieve it, and how are they going about it? The information gleaned from task analysis establishes a foundation of existing practices on which to build new requirements or to design new tasks.

Task analysis is an umbrella term that covers techniques for investigating cognitive processes and physical actions, at a high level of abstraction and in minute detail. In practice, task analysis techniques have had a mixed reception. The most widely used version is Hierarchical Task Analysis (HTA) and this is the technique we introduce in this chapter. Another well-known task analysis technique called GOMS (goals, operations, methods, and selection rules) that models procedural knowledge (Card et al., 1983) is described in Chapter 14.

7.7.1 Hierarchical task analysis

Hierarchical Task Analysis (HTA) was originally designed to identify training needs (Annett and Duncan, 1967). It involves breaking a task down into subtasks and then into sub-subtasks and so on. These are then grouped together as plans that specify how the tasks might be performed in an actual situation. HTA focuses on the physical and observable actions that are performed, and includes looking at actions that are not related to software or an interaction device at all. The starting point is a user goal. This is then examined and the main tasks associated with achieving that goal are identified. Where appropriate, these tasks are subdivided into subtasks.

Consider the library catalog service, and the task of borrowing a book. This task can be decomposed into other tasks such as accessing the library catalog, searching by name, title, subject, or whatever, making a note of the location of the book, going to the correct shelf, taking it down off the shelf (provided it is there) and finally tak-

0. In order to borrow a book from the *library*
   1. go to the library
   2. find the required book
      2.1 access library catalog
      2.2 access the search screen
      2.3 enter search criteria
      2.4 identify required book
      2.5 note location
   3. go to correct shelf and retrieve book
   4. take book to checkout counter

plan 0: do **1-3-4.** If book isn't on the shelf expected, do **2-3-4.**
plan 2: do **2.1-2.4-2.5.** If book not identified do **2.2-2.3-2.4-2.5.**

Figure 7.11 An **HTA** for borrowing a book from the library.

it to the check-out counter. This set of tasks and subtasks might be performed in a different order depending on how much is known about the book, and how familiar the user might be with the library and the book's likely location. Figure 7.11 shows these subtasks and some plans for different paths through those subtasks. Indentation shows the hierarchical relationship between tasks and subtasks.

Note how the numbering works for the task analysis: the number of the plan corresponds to the number of the step to which the plan relates. For example, plan 2 shows how the subtasks in step 2 can be ordered; there is no plan 1 because step 1 has no subtasks associated with it.

An alternative expression of an HTA is a graphical box-and-line notation. Figure 7.12 shows the graphical version of the HTA in Figure 7.11. Here the subtasks are represented by named boxes with identifying numbers. The hierarchical relationship between tasks is shown using a vertical line. If a task is not decomposed any further then a thick horizontal line is drawn underneath the corresponding box.

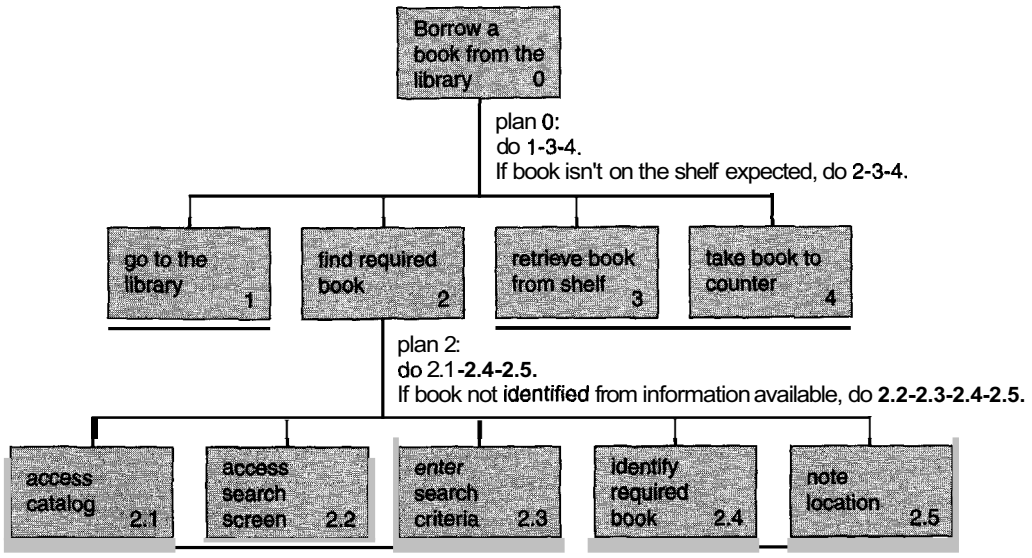


Figure 7.12 A graphical representation of the task analysis for borrowing a book.

Plans are also shown in this graphical form. They are written alongside the vertical line emitting from the task being decomposed. For example, in Figure 7.12 plan 2 is specified next to the vertical line from box 2 "find required book."

ACTIVITY 7.6 Look back at the scenario for arranging a meeting in the shared calendar application. Perform hierarchical task analysis for the goal of arranging a meeting. Include all plans in your answer. Express the task analysis textually and graphically.

Comment

The main tasks involved in this are to find out who needs to be at the meeting, find out the constraints on the meeting such as length of meeting, range of dates, and location, find a suitable date, enter details into the calendar, and inform attendees. Finding a suitable date can be decomposed into other tasks such as looking in the departmental calendar, looking in individuals' calendars, and checking potential dates against constraints. The textual version of the HTA is shown below. Figure 7.13 shows the corresponding graphical representation.

0. In order to arrange a meeting
   1. compile a list of meeting attendees
   2. compile a list of meeting constraints
   3. find a suitable date
      3.1 identify potential dates from departmental calendar
      3.2 identify potential dates from each individual's calendar
      3.3 compare potential dates
      3.4 choose one preferred date
   4. enter meeting into calendars
   5. inform meeting participants of calendar entry

plan 0: do 1-2-3. If potential dates are identified, do 4-5. If no potential dates can be identified, repeat 2-3.

plan 3: do 3.1-3.2-3.3-3.4 or do 3.2-3.1 -3.3-3.4

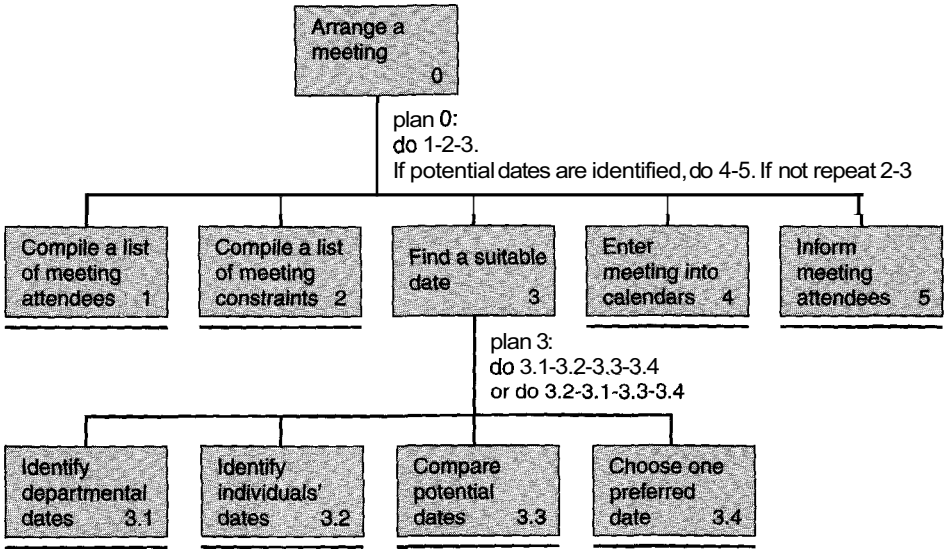


Figure 7.13 A graphical representation of the meeting HTA.

ACTIVITY 7.7 What do you think are the main problems with using task analysis on real problems? Think of more complex tasks such as scheduling delivery trucks, or organizing a large conference.

Comment

Real tasks are very complex. One of the main problems with task analysis is that it does not scale very well. The notation soon becomes unwieldy, making it difficult to follow. Imagine what it would be like to produce a task analysis in which there were hundreds or even thousands of subtasks.

A second problem is that task analysis is limited in the kind of tasks it can model. For example, it cannot model tasks that are overlapping or parallel, nor can it model interruptions. Most people work through interruptions of various kinds, and many significant tasks happen in parallel.

Assignment

This assignment is the first of four assignments that together take you through the complete development lifecycle for an interactive product. This assignment requires you to use techniques described in this chapter for identifying needs and establishing requirements. The further three assignments are at the end of ***Chapters*** *8, 13, and 14.*

The overall assignment is for you to design and evaluate an interactive website for booking tickets online for events like concerts, the theatre and the cinema. This is currently an activity that in many instances, can be difficult or inconvenient to achieve using traditional means (e.g., waiting for ages on the phone to get hold of an agent, queuing for hours in the rain at a ticket office).

For this assignment, you should:

(a) Identify users' needs for this website. You could do this in a number of ways. For example, you could observe people using ticket agents, think about your own experience of purchasing tickets, look at existing websites for booking tickets, talk to friends and family about their experiences, and so on. Record your data carefully.

(b) Based on your user requirements, choose two different user profiles and produce one main scenario for each one, capturing how the user is expected to interact with the system.

(c) Using the scenarios generated from your data gathering, perform a task analysis on the main task associated with the ticket booking system, i.e., booking a ticket.

(d) Based on the data gathered in part (a) and your subsequent interpretation and analysis, identify different kinds of requirements for the website, according to the headings introduced in Section 7.3 above. Write up the requirements in the style of the Volere template.

Summary

In this chapter, we have looked in more detail at how to identify users' needs and establish requirements for interaction design. Various data-gathering techniques can be used to collect data for interpretation and analysis. The most common of these are questionnaires, interviews, focus groups, workshops, naturalistic observation, and studying documentation. Each of these has advantages and disadvantages that must be balanced against your constraints when choosing which techniques to use for a particular project. They can be combined in many different ways, and can be supported by props such as scenarios and prototypes. How

to carry out these techniques is covered in Chapters 12 through 14, Scenarios, use cases, and essential use cases are helpful techniques for beginning to document the findings from the data-gathering sessions. Task analysis is a little more structured, but does not scale well.

Key points

- Getting the requirements right is crucial to the success of the interactive product.
- There are different kinds of requirements: functional, data, environmental, user, and usability. Every system will have requirements under each of these headings.
- The most commonly used data-gathering techniques for this activity are: questionnaires, interviews, workshops or focus groups, naturalistic observation, and studying documentation.
- Descriptions of user tasks such as scenarios, use cases, and essential use cases help users to articulate existing work practices. They also help to express envisioned use for new devices.
- Task analysis techniques help to investigate existing systems and current practices.

Further reading

ROBERTSON, SUZANNE, AND ROBERTSON, JAMES (1999) ***Mastering the Requirements Process.*** Boston: Addison-Wesley. In this book, Robertson and Robertson explain a useful framework for software requirements work (see also the interview with Suzanne Robertson after this chapter).

CONSTANTINE, LARRY L., AND LOCKWOOD, LUCY A. D. (1999) ***Software for Use.*** Boston: Addison-Wesley. This very readable book provides a concrete approach for modeling and analyzing software systems. The approach has a user-centered focus and contains some useful detail. It also includes more information about essential use cases.

JACOBSON, I., BOOCH, G., AND RUMBAUGH, J. (1992) ***The Unified Software Development Process.*** Boston: Addison-Wesley. This is not an easy book to read, but it is the definitive guide for developing object-oriented systems using use cases and the modeling language Unified Modeling Language (UML).

BRUEGGE, BERND, AND DUTOIT, ALLEN H. (2000) ***Object-oriented Software Engineering.*** Upper Saddle River, NJ: Prentice-Hall. This book is a comprehensive treatment of the whole development process using object-oriented techniques such as use cases. The book is organized to help those involved in project work.

SOMMERVILLE, IAN (2001) ***Software Engineering*** (6th ed.). Boston: Addison-Wesley. If you are interested in pursuing notations for functional and data requirements, then this book introduces a variety of notations and techniques used in software engineering.

INTERVIEW with Suzanne Robertson

Suzanne Roberston is a principal of The Atlantic Systems Guild, an international think tank producing numerous books and seminars whose aim is to make good ideas to do with systems engineering more accessible. Suzanne is particularly well known for her work in systems analysis and requirements gathering activities.

HS: What are requirements?

SR: Well the problem is that "requirements" has turned into an elastic term. Requirements is an enormously wide field and there are so many different types of requirements. One person may be talking about budget, somebody else may be talking about interfacing to an existing piece of software, somebody else may be talking about a performance requirement, somebody else may be talking about the calculation of an algorithm, somebody else may be talking about a data definition, and I could go on for hours as to what requirement means. What we advise people to do to start with is to look for something we call "linguistic integrity" within their own project. When all people who are connected with the project are talking about requirements, what do they mean? This gets very emotional, and that's why we came up with our framework. We gathered together all this experience of different types of requirements, tried to pick the most common organization, and then wrote them down in a framework.

HS: Please would you explain your framework? (The version discussed in this interview is shown in the figure on page 238. The most recent version may be downloaded from www.systemsguild.com.)

SR: Imagine a huge filing cabinet with 27 drawers, and in each drawer you've got a category of knowledge that is related to requirements. In the very first drawer for example you've got the goals, i.e., the reason for doing the project. In the second drawer you've got the stakeholders. These are roles because they could be played by more than one person, and one person may play more than one role. You've got the client who's going to pay for the development, and the customer who's making the decision about buying it. Then you've got stakeholders like the project leader, the developers, the requirements engineers, the designers, the quality people, and the testers. Then you've got the less obvious stakeholders like surrounding organizations, professional bodies, and other people in the organization whose work might be affected by the project you're doing, even if they're never going to use the product.

HS: So do you find the stakeholders by just asking questions?

SR: Yes, partly that and partly by using the domain model of the subject matter, which is in drawer 9, as the driver to ask more questions about the stakeholders. For example, for each one of the subject matter areas, ask who have we got to represent this subject matter? For each one of the people that we ***come*** across, ask what subject matter are we expecting from them?

Drawer 3 contains the end users. I've put them in a separate drawer because an error that a lot of people make when they're looking for requirements is that the only stakeholder they talk about is the end user. They decide on the end user too quickly and they miss opportunities. So you end up building a product that is possibly less competitive. I keep them a bit fuzzy to start with, and as you start to fix on them then you can go into really deep analysis about them: What is their psychology? What are their characteristics? What's their subject-matter knowledge? How do they feel about their work? How do they feel about technology? All of these things help you to come up with the most competitive non-functional requirements for the product.

HS: How do you resolve conflict between stakeholders?

SR: Well, part of it is to get the conflicts out in the open up front, so people stop blaming each other, but that certainly doesn't resolve it. One of the ways is to make things very visible all the way through and to keep reminding people that conflict is respectable, that it's a sign of creativity, of people having ideas. The other thing that we do is that in our individual requirements (that is atomic requirements), which end up living in drawers 9 to 17 of this filing cabinet, we've got a place to say "Conflict: Which other requirement is this in conflict with?" and we encourage people to

identify them. Sometimes these conflicts resolve themselves because they're on people's back burners, and some of the conflicts are resolved by people just talking to one another. We have a point at which we cross-check requirements and look for conflicts and if we find some that are just not sorting themselves out, then we stop and have a serious negotiation.

In essence, it's bubbling the conflicts up to the surface. Keep on talking about them and keep them visible. De-personalize it as much as you can. That helps.

HS: What other things are associated with these atomic requirements?

SR. Each one has a unique number and a description that is as close as you can get to what you think the thing means. It also has a rationale that helps you to figure out what it really is. Then the next component is the fit criterion, which is, "If somebody came up with a solution to this requirement, how would you know whether or not it satisfies the requirement?" So this means making the requirement quantifiable, measurable. **And** it's **very** powerful because it **makes** you think about the requirement. One requirement quite often turns into several when you really try and quantify it. It also provides a wonderful opportunity for involving testers, because at that point if you write the fit criterion you can get a tester and ask whether this can be used as input to writing a cost-effective test. Now this is different from the way we usually use the testers, which is to build tests that test our solutions. Here I want to get them in much earlier, I want them to test whether this requirement really is a requirement.

***HS: So* what's in drawers 18 through 27?**

SR: Well here you can get into serious quarrels. The overall category is "project issues," and people often say they're not really requirements, and they aren't. But if the project is not being managed according to the real work that's being done, in other words the contents of the drawers, then the project goes off the rails. In project issues we create links so that a project manager can manage the project according to what's happening to the requirements.

In the last drawer we have design ideas. People say when you're gathering requirements you should not be concerned with how you're going to solve the problem. But mostly people tell you requirements in the form of a solution anyway. The key thing is to learn how to separate the real requirements from solution ideas, and when you get a solution idea, pop it in this drawer. This helps requirements engineers, I think, because we are trained to think of solutions, not to dig behind and find the real problem.

***HS:* How do you go about identifying requirements?**

SR. For too long we've been saying the stakeholders should give us their requirements: we'll ask them and they'll give them to us. We've realized that this is not practical—partly because there are many requirements people don't know they've got. Some requirements are conscious and they're usually because things have gone wrong or they'd like something extra. Some requirements are unconscious because maybe people are used to it, or maybe they haven't a clue because they don't see the overall picture. And then there are undreamed-of requirements that people just don't dream they could ever have, because we've all got boundaries based on what we think technology is capable of doing or what we know about technology or what our experience is. So it's not just asking people for things, it's also inventing requirements. I think that's where prototyping comes in and scenario modeling and storyboarding and all of those sorts of techniques to help people to imagine what they could have.

If you're building a product for the market and you want to be more competitive you should be inventing requirements. Instead of constricting yourself within the product boundary, say, "Can I push myself out a bit further? Is there something else I could do that isn't being done?"

***HS: So* what kinds of techniques can people use to push out further?**

SR: One of the things is to learn how to imagine what it's like to be somebody else, and this is why going into other fields, for example family therapy, is helpful. They've learned an awful lot about how to imagine you might be somebody else. And that's not something that software engineers are taught in college normally and this is why it's very healthy for us to be bringing together the ideas of psychology and sociology and so on with software and systems engineering. Bringing in these human aspects—the performance, the usability features, the "look and feel" features—that's going to make our products more competitive. I always tell people to read a lot of novels. If you're having trouble relating to some stakeholders, for example, go and read some Jane Austen and then try to

imagine what it would have been like to have been the heroine in ***Pride and Prejudice.*** What would it have been like to have to change your clothes three times a day? I find this helps me a lot, it frees your mind and then you can say, "OK, what's it really like to be that other person?" There's a lot to learn in that area.

HS: So what you're saying really is that it's not easy.

SR. It's not easy. I don't think there's any particular technique. But what we have done is we have come up with a lot of different "trawling" techniques, along with recommendations, that can help you.

HS: Do you have any other tips for gathering requirements?

SR: It's important for people to feel that they've been heard. The waiting room (drawer number 26) was invented because of a very enthusiastic high-level stakeholder in a project we were doing. She was very enthusiastic and keen and very involved. Wonderful! She really gave us tremendous ideas and support. The problem was she kept having ideas, and we didn't know what to do. We didn't want to stop her having ideas, on the other hand we couldn't always include them because then we would never get anything built. So we invented the waiting room. All the good ideas we have we put in there and every so often we go into the waiting room and review the ideas. Some of them get added to the product, some are discarded, and some are left waiting. The psychology of it is very good because the idea's in the waiting room, everyone knows it's in there, but it's not being ignored. When people feel heard, they feel better and consequently they're more likely to cooperate and give you time.

The Template

PROJECT DRIVERS

1. The Purpose of the Product
2. Client, Customer and other Stakeholders
3. Users of the Product

PROJECT CONSTRAINTS

4. Mandated Constraints
5. Naming Conventions and Definitions
6. Relevant Facts and Assumptions

FUNCTIONAL REQUIREMENTS

7. The Scope of the Work
8. The Scope of the Product
9. Functional and Data Requirements

NON-FUNCTIONAL REQUIREMENTS

10. Look and Feel Requirements
11. Usability Requirements
12. Performance Requirements
13. Operational Requirements
14. Maintainability and Portability Requirements
15. Security Requirements
16. Cultural and Political Requirements
17. Legal Requirements

PROJECT ISSUES

18. Open Issues
19. Off-the-shelf Solutions
20. New Problems
21. Tasks
22. Cutover
23. Risks
24. Costs
25. User Documentation and Training
26. Waiting Room
27. Ideas for Solutions