6 Requirements engineering processes

Objectives

The objective of this chapter is to describe a number of generic requirements engineering processes. When you have read this chapter, you will:

- understand the principal requirements engineering activities and their relationships;
- have been introduced to several techniques of requirements elicitation and analysis;
- understand the importance of requirements validation and how requirements reviews are used in this process;
- understand why requirements management is necessary and how it supports other requirements engineering activities.

Contents

Requirements engineering is a process that involves all of the activities required to create and maintain a system requirements document. There are four generic, high-level requirements engineering process activities. These are a system feasibility study, the elicitation and analysis of requirements, the specification of requirements and their documentation and, finally, the validation of these requirements. I cover all of these activities in this chapter apart from specification and documentation which I have already discussed in Chapter 5. Figure 6.1 illustrates the relationship between these activities. It also shows the documents produced at each stage of the requirements engineering (RE) process.

The requirements engineering activities shown in Figure 6.1 are concerned with the elicitation, documentation and checking of requirements. In virtually all systems, however, requirements change. The people involved develop a better understanding of what they want the software to do; the organisation buying the system changes; modifications are made to the system's hardware, software and organisational environment. Requirements management is an additional requirements engineering activity which is concerned with managing requirements change. This is covered in the final section of the chapter.

Some people consider requirements engineering to be the process of applying a structured method, such as object-oriented analysis (Rumbaugh *et al.*, 1991; Booch, 1994). This involves analysing the system and developing a set of graphical system models which then act as a system specification. The set of models describe the behaviour of the system and are annotated with additional information describing, for example, its required performance or reliability.

Although structured methods have a role to play in the requirements engineering process, there is much more to requirements engineering than is covered by these methods. They do not provide effective support for early RE process stages such as requirements elicitation. I therefore focus here on more general approaches to requirements engineering and I discuss structured analysis methods and system models in Chapter 7.

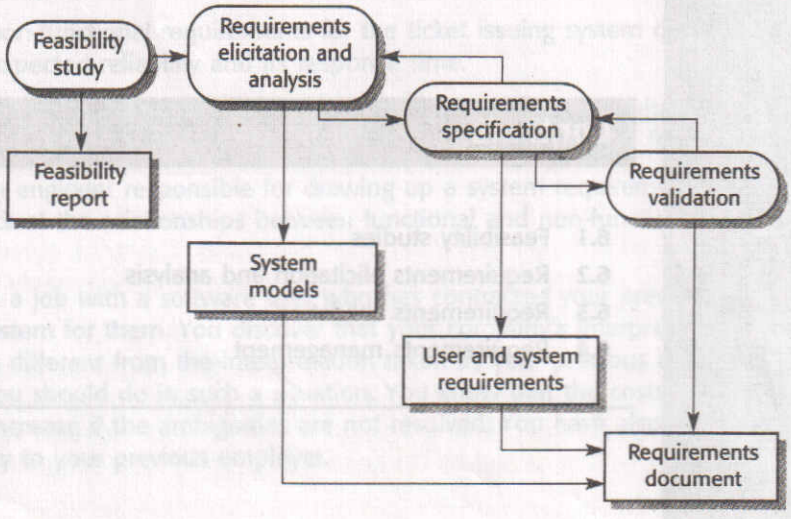


Figure 6.1
The requirements engineering process

6.1 Feasibility studies

For all new systems, the requirements engineering process should start with a feasibility study. The input to the feasibility study is an outline description of the system and how it will be used within an organisation. The results of the feasibility study should be a report which recommends whether or not it is worth carrying on with the requirements engineering and system development process.

A feasibility study is a short, focused study which aims to answer a number of questions:

1. Does the system contribute to the overall objectives of the organisation?
2. Can the system be implemented using current technology and within given cost and schedule constraints?
3. Can the system be integrated with other systems which are already in place?

The issue of whether or not the system contributes to business objectives is critical. If a system does not support these objectives, it has no real value to the business. While this may seem obvious, many organisations develop systems which do not contribute to their objectives either because they don't have a clear statement of these objectives or because other political or organisation factors influence the system procurement.

Carrying out a feasibility study involves information assessment, information collection and report writing. The information assessment phase identifies the information which is required to answer the three questions set out above. Once the information has been identified, you should question information sources to discover the answers to these questions. Some examples of possible questions that may be put are:

1. How would the organisation cope if this system was not implemented?
2. What are the problems with current processes and how would a new system help alleviate these problems?
3. What direct contribution will the system make to the business objectives?
4. Can information be transferred to and from other organisational systems?
5. Does the system require technology which has not previously been used in the organisation?
6. What must be supported by the system and what need not be supported?

Information sources may include the managers of departments where the system will be used, software engineers who are familiar with the type of system that is proposed, technology experts, end-users of the system, etc. They should be interviewed during the feasibility study to collect the required information.

When the information is available, the feasibility study report is prepared. This should make a recommendation about whether or not the system development should continue. It may propose changes to the scope, budget and schedule of the system and suggest further high-level requirements for the system.

6.2 Requirements elicitation and analysis

After initial feasibility studies, the next stage of the requirements engineering process is requirements elicitation and analysis. In this activity, technical software development staff work with customers and system end-users to find out about the application domain, what services the system should provide, the required performance of the system, hardware constraints, and so on.

Requirements elicitation and analysis may involve a variety of different kinds of people in an organisation. The term *stakeholder* is used to refer to anyone who should have some direct or indirect influence on the system requirements. Stakeholders include end-users who will interact with the system and everyone else in an organisation who will be affected by it. Engineers who are developing or maintaining other related systems, business managers, domain experts, trade union representatives, and so on may also be system stakeholders.

Elicitation and analysis is a difficult process for a number of reasons:

1. Stakeholders often don't really know what they want from the computer system except in the most general terms; they may find it difficult to articulate what they want from the system; they may make unrealistic demands because they are unaware of the cost of their requests.
2. Stakeholders in a system naturally express requirements in their own terms and with implicit knowledge of their own work. Requirements engineers, without experience in the customer's domain, must understand these requirements.
3. Different stakeholders have different requirements and they may express these in different ways. Requirements engineers have to discover all potential sources of requirements and discover commonalities and conflict.
4. Political factors may influence the requirements of the system. These may come from managers who demand specific system requirements because these allow them to increase their influence in the organisation.
5. The economic and business environment in which the analysis takes place is dynamic. It inevitably changes during the analysis process. Hence the importance of particular requirements may change. New requirements may emerge from new stakeholders who were not originally consulted.

A generic process model of the elicitation and analysis process is shown in Figure 6.2. Each organisation will have its own version or instantiation of this general model depending on local factors such as the expertise of the staff, the type of system being developed, the standards used, etc.

The process activities are:

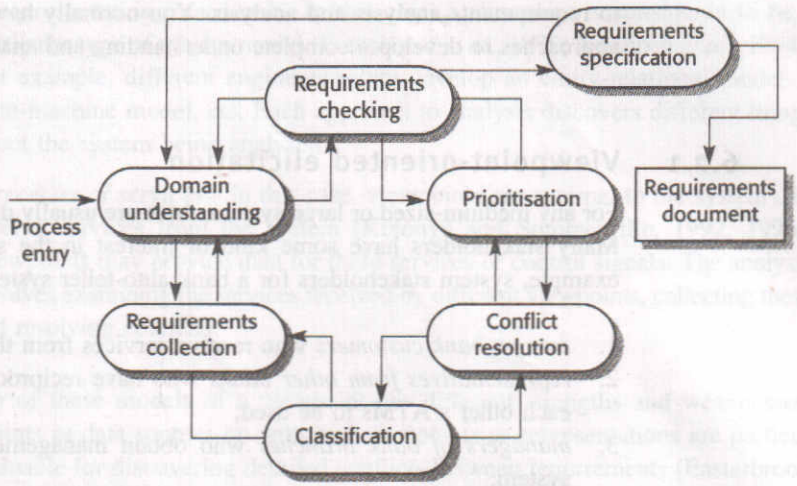


Figure 6.2 The requirements elicitation and analysis process

1. *Domain understanding* Analysts must develop their understanding of the application domain. For example, if a system for a supermarket is required, the analyst must find out how supermarkets operate.
2. *Requirements collection* This is the process of interacting with stakeholders in the system to discover their requirements. Obviously, domain understanding develops further during this activity.
3. *Classification* This activity takes the unstructured collection of requirements and organises them into coherent clusters.
4. *Conflict resolution* Inevitably, where multiple stakeholders are involved, requirements will conflict. This activity is concerned with finding and resolving these conflicts.
5. *Prioritisation* In any set of requirements some will be more important than others. This stage involves interaction with stakeholders to discover the most important requirements.
6. *Requirements checking* The requirements are checked to discover if they are complete, consistent and in accordance with what stakeholders really want from the system.

Figure 6.2 shows that requirements elicitation and analysis is an iterative process with continual feedback from each activity to other activities. The process cycle starts with domain understanding and ends with requirements checking. The analyst's understanding of the requirements improves with each round of the cycle.

In this section, I cover three techniques for requirements elicitation and analysis. These are viewpoint-oriented elicitation, scenarios and ethnography. Other techniques which may be used during this phase of the requirements engineering process include structured analysis methods, covered in Chapter 7, and prototyping, covered in Chapter 8. There is no perfect and universally applicable approach

to requirements analysis and analysis. You normally have to use several of these approaches to develop a complete understanding and analysis of the requirements.

6.2.1 Viewpoint-oriented elicitation

For any medium-sized or large system, there are usually different types of end-user. Many stakeholders have some kind of interest in the system requirements. For example, system stakeholders for a bank auto-teller system (ATM) include:

1. *current bank customers* who receive services from the system;
2. *representatives from other banks* who have reciprocal agreements that allow each other's ATMs to be used;
3. *managers of bank branches* who obtain management information from the system;
4. *counter staff at bank branches* who are involved in the day-to-day running of the system, handling customer complaints, etc.;
5. *database administrators* who are responsible for integrating the system with the bank's customer database;
6. *bank security managers* who must ensure that the system will not pose a security hazard of some kind;
7. *the bank's marketing department* who are likely be interested in using the system as a means of marketing the bank;
8. *hardware and software maintenance engineers* who are responsible for maintaining and upgrading the hardware and software.

This list shows that for even a relatively simple system, there are many different *viewpoints* that should be considered. Different viewpoints on a problem see the problem in different ways. However, their perspectives are not completely independent but usually overlap so that they have common requirements.

Viewpoint-oriented approaches to requirements engineering recognise these different viewpoints and use them to structure and organise both the elicitation process and the requirements themselves. A key strength of viewpoint-oriented analysis is that it recognises the existence of multiple perspectives and provides a framework for discovering conflicts in the requirements proposed by different stakeholders.

Different methods have different ideas of what is meant by a 'viewpoint'. A viewpoint may be considered as:

1. *A data source or sink* In this case, viewpoints are responsible for producing or consuming data. The analysis involves identifying all such viewpoints, identifying what data is produced or consumed and what processing is carried out. Methods such as SADT (Ross, 1977; Schoman and Ross, 1977) and CORE (Mullery, 1979), the first method to propose explicit viewpoints, use this type of viewpoint.

2. *A representation framework* In this case, a viewpoint is considered to be a particular type of system model (Finkelstein *et al.*, 1990; Nuseibeh *et al.*, 1994). For example, different engineers might develop an entity-relational model, a state-machine model, etc. Each approach to analysis discovers different things about the system being analysed.

3. *A receiver of services* In this case, viewpoints are external to the system and receive services from the system (Kotonya and Sommerville, 1992, 1996). Viewpoints may provide data for these services or control signals. The analysis involves examining the services received by different viewpoints, collecting these and resolving conflicts.

Each of these models of a viewpoint has different strengths and weaknesses. Viewpoints as data sources or sinks and viewpoints as representations are particularly valuable for discovering detailed conflicts between requirements (Easterbrook and Nuseibeh, 1996). However, they are less suited to structuring the requirements analysis process as there is no simple relationship between viewpoints and system stakeholders.

Interactive systems deliver services to end-users. Consequently, the most effective viewpoint-oriented approach for interactive systems analysis uses external viewpoints. These viewpoints interact with the system by receiving services from it and providing data and control signals to the system.

The advantages of this type of viewpoint are:

1. Viewpoints are external to the system so are a natural way to structure the requirements elicitation process.

2. It is relatively easy to decide if something is a valid viewpoint. Viewpoints must interact with the system in some way.

3. Viewpoints and services are a useful way of structuring non-functional requirements. Each service may have associated non-functional requirements. Multiple viewpoints allow the same service to have different non-functional requirements in different viewpoints.

The VORD (Viewpoint-Oriented Requirements Definition) method (Kotonya and Sommerville, 1996, 1998) has been designed as a service-oriented framework for requirements elicitation and analysis. The principal stages of the VORD method, shown in Figure 6.3, are:

1. Viewpoint identification, which involves discovering viewpoints that receive system services and identifying the specific services provided to each viewpoint.

2. Viewpoint structuring, which involves grouping related viewpoints into a hierarchy. Common services are provided at higher levels in the hierarchy and are inherited by lower-level viewpoints.

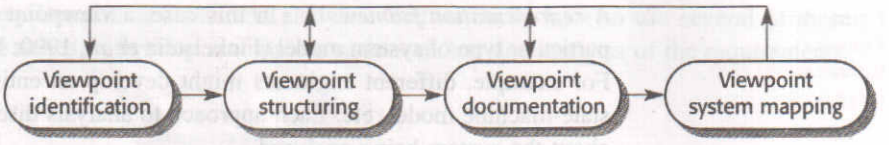


Figure 6.3 The VORD method

3. Viewpoint documentation, which involves refining the description of the identified viewpoints and services.
4. Viewpoint-system mapping, which involves identifying objects in an object-oriented design using service information which is encapsulated in viewpoints.

Viewpoint and service information in VORD is collected using standard forms. The forms used for viewpoint information (the viewpoint template) and service information (the service template) are shown in Figure 6.4. VORD also uses various diagrammatic notations, including viewpoint hierarchy diagrams (see example in Figure 6.8) and event scenarios (see example in Figure 6.10).

I illustrate the application of VORD by applying the first three steps in the analysis of requirements for a bank auto-teller (ATM) control system. Automated teller machines have an embedded software system to drive the machine hardware and to communicate with the bank's central account database.

The ATM accepts customer requests and delivers cash, account information, database updates, etc. Customers may withdraw and pay in cash, check their balance, transfer funds from one account to another, request an account statement, cheque book, etc. The machines provided by one bank may allow customers of other banks to use a subset (typically cash withdrawal and account balance querying) of their facilities.

The first step in viewpoint analysis is to identify possible viewpoints. As in all methods, this initial identification is probably the most difficult stage. One

Viewpoint template	
Reference:	The viewpoint name.
Attributes:	Attributes providing viewpoint information.
Events:	A reference to a set of event scenarios describing how the system reacts to viewpoint events.
Services:	A reference to a set of service descriptions.
Sub-VPs:	The names of sub-viewpoints.

Service template	
Reference:	The service name.
Rationale:	Reason why the service is provided.
Specification:	Reference to a list of service specifications. These may be expressed in different notations.
Viewpoints:	List of viewpoint names receiving the service.
Non-functional requirements:	Reference to a set of non-functional requirements which constrain the service.
Provider:	Reference to a list of system objects which provide the service.

Figure 6.4 Viewpoint and service template forms

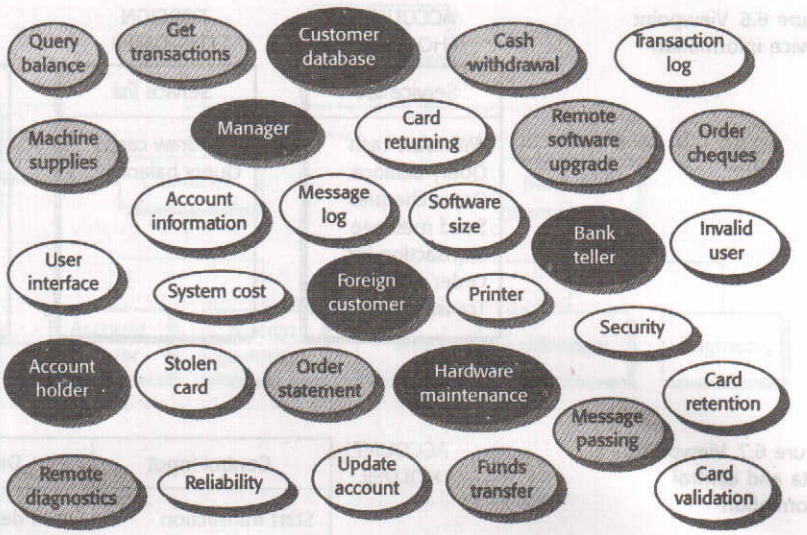


Figure 6.5 Brainstorming for viewpoint identification

approach is a brainstorming approach where potential services and entities which interact with the system are identified. Stakeholders meet and suggest possible viewpoints which are written down in a bubble diagram as shown in Figure 6.5. In this diagram, potential viewpoints for an ATM system are shown in separate bubbles.

When brainstorming, you should try to identify potential viewpoints, system services, data inputs, non-functional requirements, control events and exceptions. At this stage of the analysis, you should not try to impose a structure on the diagram. Sources of information which may be used in creating this initial view of the system may be documents setting out the high-level goals of the system, knowledge of software engineers from previous projects or experience as bank customers. Interviews may be held with bank managers, counter staff, consultants, engineers and customers.

The next stage of the process is to identify viewpoints (shown as dark blue bubbles in Figure 6.5) and services (shown as shaded bubbles). The services should be allocated to viewpoints. Unallocated services can suggest viewpoints that have not been identified in the initial brainstorming session. For example, the 'Remote software upgrade' and 'Remote diagnostics' services in Figure 6.5 imply that there may be a need for a software maintenance viewpoint.

Figure 6.6 illustrates, for some of the viewpoints identified in Figure 6.5, the allocation of services. The same service may be allocated to several viewpoints.

As well as receiving services, viewpoints also provide inputs to these services. For example, auto-teller users must specify the amount of money they want when they withdraw cash. Viewpoints also provide control information to determine if and when services are delivered.

During this early stage of the process, this data and control information is simply identified by name. Figure 6.7 shows this control information for the account holder viewpoint whose services are identified in Figure 6.6. Control information is

Figure 6.6 Viewpoint service information

ACCOUNT HOLDER	FOREIGN CUSTOMER	BANK TELLER
Service list	Service list	Service list
Withdraw cash Query balance Order cheques Send message Transaction list Order statement Transfer funds	Withdraw cash Query balance	Run diagnostics Add cash Add paper Send message

Figure 6.7 Viewpoint data and control information

ACCOUNT HOLDER

Control input	Data input
Start transaction Cancel transaction End transaction Select service	Card details PIN Amount required Message

provided through buttons on the machine; data through the user's card or the machine keyboard.

The viewpoint information is used to fill in viewpoint template forms and to organise the viewpoints into an inheritance hierarchy. To show viewpoint commonalities and to reuse viewpoint information, the inheritance hierarchy factors out viewpoints which provide common services. Services, data and control information are inherited by sub-viewpoints.

Figure 6.8 shows part of the viewpoint hierarchy for the ATM system. To avoid clutter, I have shown only the services associated with two viewpoints and I have left out a number of bank staff viewpoints. As services are inherited down the viewpoint hierarchy, the general services associated with the 'Customer' viewpoint (Query balance and Withdraw cash) are inherited by 'Account holder' and 'Foreign customer'.

The next process stage is to discover more detailed information about the services provided, the data which they require and how these are controlled. Requirements are elicited from the stakeholders associated with each viewpoint. The service needs of each different viewpoint are discussed with either end-users or with viewpoint experts if the viewpoint is another automated system. Figure 6.9 shows an example of a completed viewpoint template for the customer viewpoint and a template for the cash withdrawal service.

Notice that one of the fields in the viewpoint template is used to refer to event scenarios which describe how the system behaves in response to various events. These event scenarios are discussed in the following section.

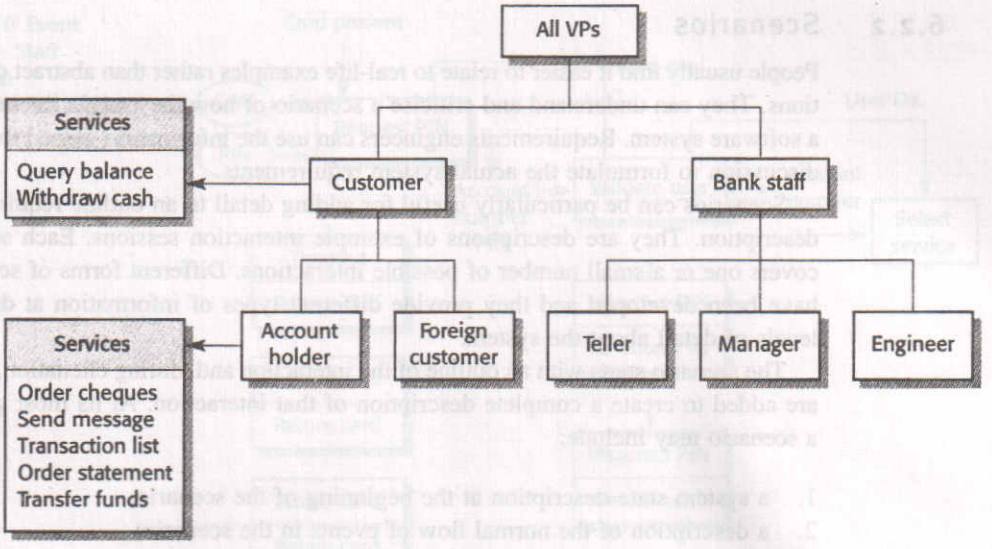


Figure 6.8
Viewpoint hierarchy

Viewpoint and service templates and event scenarios are developed for all identified viewpoints and services. The information may then be cross-checked to discover errors in the analysis and requirements conflicts. As this generates a great deal of information, VORD, like other analysis methods, is only practically usable with CASE tool support. A free set of CASE tools for VORD is available and can be downloaded through the book's web pages.

Figure 6.9 Customer viewpoint and cash withdrawal descriptions

Reference:	Customer
Attributes:	Account number PIN Start transaction
Events:	Select service Cancel transaction End transaction
Services:	Cash withdrawal Balance enquiry
Sub-VPs:	Account holder Foreign customer

Reference:	Cash withdrawal
Rationale:	To improve customer service and reduce paperwork
Specification:	Users choose this service by pressing the cash withdrawal button. They then enter the amount required. This is confirmed and, if funds allow, the balance is delivered.
VPs:	Customer
Non-funct. requirements:	Deliver cash within 1 minute of amount being confirmed
Provider:	*Filled in later*

6.2.2 Scenarios

People usually find it easier to relate to real-life examples rather than abstract descriptions. They can understand and criticise a scenario of how they might interact with a software system. Requirements engineers can use the information gained from this discussion to formulate the actual system requirements.

Scenarios can be particularly useful for adding detail to an outline requirements description. They are descriptions of example interaction sessions. Each scenario covers one or a small number of possible interactions. Different forms of scenarios have been developed and they provide different types of information at different levels of detail about the system.

The scenario starts with an outline of the interaction and, during elicitation, details are added to create a complete description of that interaction. At its most general, a scenario may include:

1. a system state description at the beginning of the scenario;
2. a description of the normal flow of events in the scenario;
3. a description of what can go wrong and how this is handled;
4. information about other activities which might be going on at the same time;
5. a description of the state of the system after completion of the scenario.

Scenario-based elicitation can be carried out informally where the requirements engineer works with stakeholders to identify scenarios and to capture details of these scenarios. Alternatively, a more structured approach such as event scenarios or use-cases may be used.

Event scenarios

Event scenarios are used in VORD to document the system behaviour when presented with specific events. Each distinct interaction event such as inserting a card into an ATM or selecting an ATM service may be documented with a separate event scenario. Event scenarios include a description of data flows and the actions of the system and document the exceptions which can arise. To illustrate event scenarios, consider Figure 6.10 which shows the scenario to a 'Start transaction' event. This is initiated by a customer inserting his or her card into the machine. This is a more abstract description than the Java-PDL description given in Chapter 5.

The diagrammatic conventions used in event scenarios are:

1. Data provided from a viewpoint or delivered to a viewpoint is shown in ellipses.
2. Control information enters and leaves at the top of each box.
3. Data leaves from the right of each box. If it is not enclosed, this means that it is internal to the system.
4. Exceptions are shown at the bottom of the box. Where there are several possible exceptions, these are enclosed in a box as shown on the left of Figure 6.10.
5. The name of the next expected event after completion of the scenario is shown in a shaded box.

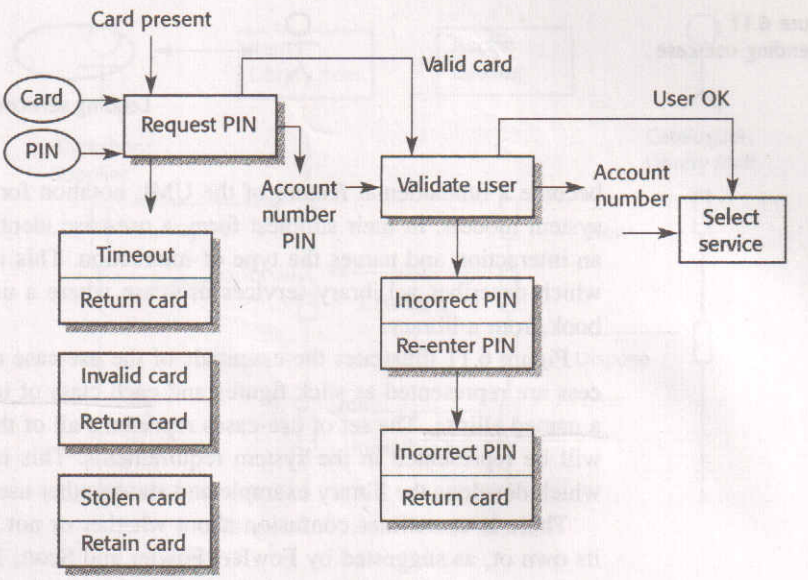


Figure 6.10 Event scenario – Start transaction

Figure 6.10 shows that when a card is entered, the customer's personal identification number (PIN) is requested. The customer inputs his or her card (the card must be present to trigger the request) and PIN. If the card is a valid card which can be processed by the machine, control can move to the next stage.

In the first stage, there are three possible exceptions:

1. *Timeout* The customer may fail to enter a PIN within the allowed time limit. The card is returned.
2. *Invalid card* The card is not recognised and is returned.
3. *Stolen card* The card is recognised as a stolen card and is retained by the machine.

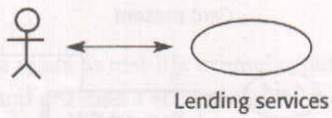
Each exception may be defined in more detail by constructing a separate data and control analysis diagram showing how that exception occurs. Alternatively, the general diagram may be annotated with extra information that explains when exceptions occur and the actions that should be taken.

The 'Validate user' stage checks that the PIN is associated with the customer's account number in which case it can trigger the next stage (Select service) with the 'User OK' event. The account number is also output from this stage. Possible exceptions are the input of an incorrect PIN in which case the PIN is requested again. The diagram shows that this repeated request can also have an exception. If an incorrect PIN is again input, the card is returned.

Use-cases

Use-cases are a scenario-based technique for requirements elicitation which were first introduced in the Objectory method (Jacobson *et al.*, 1993). They have now

Figure 6.11
A lending use-case



become a fundamental feature of the UML notation for describing object-oriented system models. In their simplest form, a use-case identifies the actors involved in an interaction and names the type of interaction. This is illustrated in Figure 6.11 which describes a Library-services use-case where a user can borrow or return a book from a library.

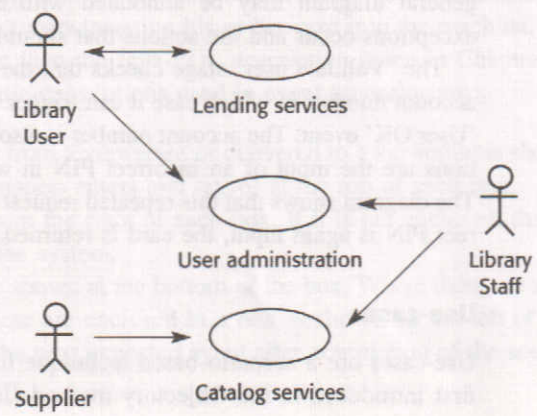
Figure 6.11 illustrates the essentials of the use-case notation. Actors in the process are represented as stick figures and each class of interaction is represented as a named ellipse. The set of use-cases represents all of the possible interactions that will be represented in the system requirements. This is illustrated in Figure 6.12 which develops the library example and shows other use-cases in that environment.

There is sometimes confusion about whether or not a use-case is a scenario on its own or, as suggested by Fowler (Fowler and Scott, 1997), a use-case encapsulates a set of scenarios where each scenario is a single thread through the use-case. In this case, there would be a scenario for the normal interaction plus scenarios for each possible exception.

Within the UML, sequence diagrams may be used to add information to a use-case. These sequence diagrams show the actors involved in the interaction, the objects within the system with which they interact and the operations which are associated with these objects. As an illustration of this, Figure 6.13 shows the interactions involved in acquiring and cataloguing a book for the library.

Figure 6.13 illustrates that there are two objects of classes Library Item and Catalog involved in the Catalogue Management use case. The sequence of actions is from top to bottom and the labels on the arrows between the actors and objects indicate the names of operations. Therefore, when a book is bought, the *New* operation is enacted on Catalog and the *Acquire* operation on Item. Once the book is available, the *Catalog item* operation is enacted on Item.

Figure 6.12
Library use-cases



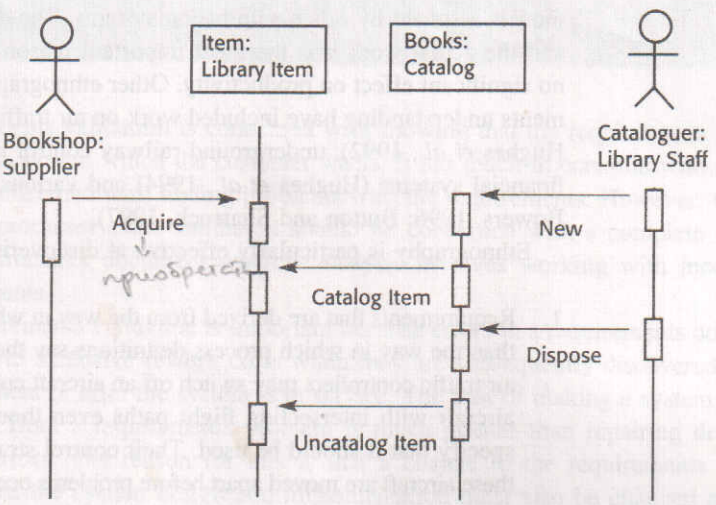


Figure 6.13
Sequence diagram for catalogue management

The UML is a *de facto* standard for object-oriented modelling so use-cases and use-case based elicitation are increasingly used for requirements elicitation. I discuss other aspects of the UML in Chapter 7 which covers system modelling and in Chapter 12 which covers object-oriented design.

6.2.3 Ethnography

Software systems do not exist in isolation. They are used in a social and organisational context and software system requirements may be derived or constrained by that context. Satisfying these social and organisational requirements is often critical for the success of the system. One reason why many software systems are delivered but never used is that they do not take proper account of the importance of this type of system requirement.

Ethnography is an observational technique that can be used to understand social and organisational requirements. An analyst immerses himself or herself in the working environment where the system will be used. The day-to-day work is observed and notes made of the actual tasks in which participants are involved. The value of ethnography is that it helps discover implicit system requirements which reflect the actual rather than the formal processes in which people are involved.

People often find it very difficult to articulate details of their work because it is second nature to them. They understand their own work but may not understand its relationship to other work in the organisation. Social and organisational factors which affect the work but which are not obvious to individuals may only become clear when noticed by an unbiased observer.

Suchman (1983) used ethnography to study office work and found that the actual work practices were far richer, more complex and more dynamic than the simple

models assumed by office automation systems. The difference between the assumed and the actual work was the most important reason why these office systems had no significant effect on productivity. Other ethnographic studies for system requirements understanding have included work on air traffic control (Bentley *et al.*, 1992; Hughes *et al.*, 1992), underground railway control rooms (Heath and Luff, 1991), financial systems (Hughes *et al.*, 1994) and various design activities (Pycock and Bowers, 1996; Button and Sharrock, 1997).

Ethnography is particularly effective at discovering two types of requirements:

1. Requirements that are derived from the way in which people actually work rather than the way in which process definitions say they ought to work. For example, air traffic controllers may switch off an aircraft conflict alert system which detects aircraft with intersecting flight paths even though normal control procedures specify that it should be used. Their control strategy is designed to ensure that these aircraft are moved apart before problems occur and they find that the conflict alert alarm distracts them from their work.
2. Requirements that are derived from cooperation and awareness of other people's activities. For example, air traffic controllers may use an awareness of other controllers' work to predict the number of aircraft which will be entering their control sector. They then modify their control strategies depending on that predicted workload. Therefore, an automated ATC system should allow controllers in a sector to have some visibility of the work in adjacent sectors.

Ethnography may be combined with prototyping (Figure 6.14). The ethnography informs the development of the prototype so that fewer prototype refinement cycles are required. Furthermore, the prototyping focuses the ethnography by identifying problems and questions which can then be discussed with the ethnographer. He or she should then look for the answers to these questions during the next phase of the system study (Sommerville *et al.*, 1993).

Ethnographic studies can reveal critical process details which are often missed by other requirements elicitation techniques. However, because of its focus on the end-user, this approach is not appropriate for discovering organisational or domain requirements. It cannot always identify new features which should be added to a system. Ethnography is not, therefore, a complete approach to elicitation and it should be used with other approaches such as use-case analysis.

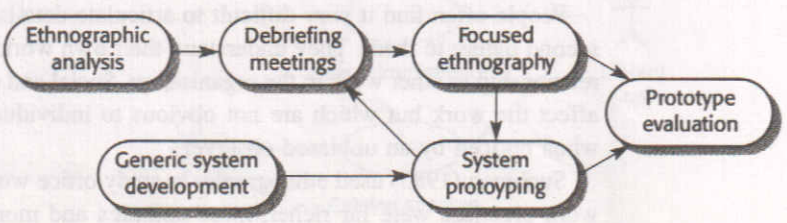


Figure 6.14 Ethnography and prototyping for requirements analysis

6.3 Requirements validation

Requirements validation is concerned with showing that the requirements actually define the system which the customer wants. It has much in common with analysis as it is concerned with finding problems with the requirements. However, they are distinct processes since validation should be concerned with a complete draft of the requirements document whereas analysis involves working with incomplete requirements.

Requirements validation is important because errors in a requirements document can lead to extensive rework costs when they are subsequently discovered during development or after the system is in service. The cost of making a system change resulting from a requirements problem is much greater than repairing design or coding errors. The reason for this is that a change to the requirements usually means that the system design and implementation must also be changed and that the system must be re-tested.

During the requirements validation process, different types of checks should be carried out on the requirements in the requirements document. These checks include:

1. *Validity checks* A user may think that a system is needed to perform certain functions. However, further thought and analysis may identify additional or different functions that are required. Systems have diverse users with different needs and any set of requirements is inevitably a compromise across the user community.
2. *Consistency checks* Requirements in the document should not conflict. That is, there should not be contradictory constraints or different descriptions of the same system function.
3. *Completeness checks* The requirements document should include requirements which define all functions and constraints intended by the system user.
4. *Realism checks* Using knowledge of existing technology, the requirements should be checked to ensure that they can actually be implemented. These checks should also take account of the budget and schedule for the system development.
5. *Verifiability* To reduce the potential for dispute between customer and contractor, system requirements should always be written so that they are verifiable. This means that a set of checks can be designed which can demonstrate that the delivered system meets that requirement.

There are a number of requirements validation techniques which can be used in conjunction or individually:

1. *Requirements reviews* The requirements are analysed systematically by a team of reviewers. This process is discussed in the following section.

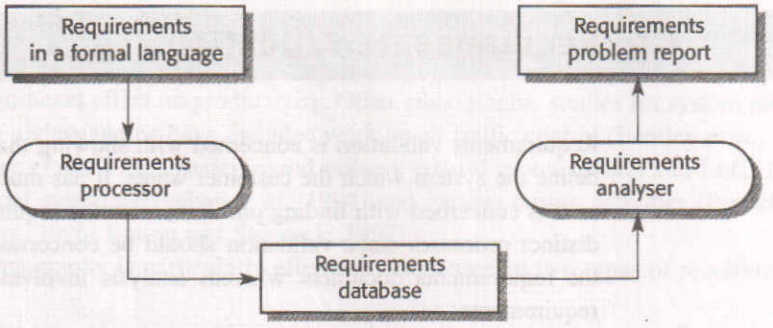


Figure 6.15 Automated consistency checking of requirements

2. *Prototyping* In this approach to validation, an executable model of the system is demonstrated to end-users and customers. They can experiment with this model to see if it meets their real needs. I discuss prototyping techniques in Chapter 8.

3. *Test-case generation* Requirements should, ideally, be testable. If the tests for the requirements are devised as part of the validation process, this often reveals requirements problems. If a test is difficult or impossible to design, this usually means that the requirements will be difficult to implement and should be reconsidered.

4. *Automated consistency analysis* If the requirements are expressed as a system model in a structured or formal notation then CASE tools may be used to check the consistency of the model. This is illustrated in Figure 6.15. To check the consistency, the CASE tool must build a requirements database and then, using the rules of the method or notation, check all of the requirements in this database. A requirements analyser produces a report of inconsistencies which it has discovered.

The difficulties of requirements validation should not be under-estimated. Demonstrating that a set of requirements meets a user's needs is difficult. Users must picture the system in operation and imagine how that system would fit into their work. It is hard for skilled computer professionals to perform this type of abstract analysis and even harder for system users. As a result, requirements validation rarely discovers all requirements problems and changes to correct omissions and misunderstandings after the requirements document has been agreed are inevitable.

6.3.1 Requirements reviews

A requirements review is a manual process which involves multiple readers from both client and contractor staff checking the requirements document for anomalies and omissions. The review process may be managed in the same way as program inspections (see Chapter 19). Alternatively, it may be organised on a larger scale with many participants involved in checking different parts of the document.

Requirements reviews can be informal or formal. Informal reviews simply involve contractors discussing requirements with as many system stakeholders as possible. It is surprising how often communication between system developers and stakeholders ends after elicitation and there is no confirmation that the documented requirements are what the stakeholders really said they wanted. Many problems can be detected simply by talking about the system to stakeholders before making a commitment to a formal review.

In a formal requirements review, the development team should 'walk' the client through the system requirements, explaining the implications of each requirement. The review team should check each requirement for consistency and should check the requirements as a whole for completeness. Reviewers may also check for:

1. *Verifiability* Is the requirement as stated realistically testable?
2. *Comprehensibility* Is the requirement properly understood by the procurers or end-users of the system?
3. *Traceability* Is the origin of the requirement clearly stated? You may have to go back to the source of the requirement to assess the impact of a change. Traceability is important as it allows the impact of change on the rest of the system to be assessed. I discuss it in more detail in the following section.
4. *Adaptability* Is the requirement adaptable? That is, can the requirement be changed without large-scale effects on other system requirements?

Conflicts, contradictions, errors and omissions in the requirements should be pointed out during the review and formally recorded. It is then up to the users, the system procurer and the system developer to negotiate a solution to these identified problems.

6.4 Requirements management

The requirements for large software systems are always changing. One reason for this is that these systems are usually developed to address 'wicked' problems (as discussed in Chapter 2). Because the problem cannot be fully defined, the software requirements are bound to be incomplete. During the software process, the developer's understanding of the problem is constantly changing and these changes feed back to the requirements.

Furthermore, large software systems are usually required to improve upon the *status quo*. The existing system may be a manual system or an out-of-date computer system. Although difficulties with the current system may be known, it is hard

to anticipate what effects the 'improved' system will have on the organisation. Once end-users have experience of a system, new requirements emerge for the following reasons:

1. Large systems usually have a diverse user community. Different users have different requirements and priorities. These may be conflicting or contradictory. The final system requirements are inevitably a compromise between them and, with experience, it is often discovered that the balance of support given to different users needs to be changed.
2. The people who pay for a system and the users of a system are rarely the same people. System customers impose requirements because of organisational and budgetary constraints. These may conflict with end-user requirements.
3. The business and technical environment of the system changes and these must be reflected in the system itself. New hardware may be introduced, it may be necessary to interface the system with other systems, business priorities may change with consequent changes in the system support which is needed and new legislation and regulations may be introduced which must be implemented by the system. Non-functional requirements are particularly affected by changes in hardware technology.

Requirements management is the process of understanding and controlling changes to system requirements. The process of requirements management is carried out in conjunction with other requirements engineering processes. Planning starts at the same time as initial requirements elicitation and active requirements management should start as soon as a draft version of the requirements document is available.

I discuss each of these activities below. Before going on to this description, however, I discuss why requirements inevitably change and explain why some types of requirements are more susceptible to change than others.

6.4.1 Enduring and volatile requirements

Developing software requirements focuses attention on software capabilities, business objectives and other business systems. As the requirements definition is developed, a better understanding of users' needs is achieved. This feeds information back to the user which causes the requirements to be changed (Figure 6.16). It may take several years to specify and develop a large system. Over that time, the system's environment and the business objectives will almost certainly change. The requirements must therefore evolve to reflect this.

From an evolution perspective, requirements fall into two classes:

1. *Enduring requirements* These are relatively stable requirements which derive from the core activity of the organisation and which relate directly to the domain

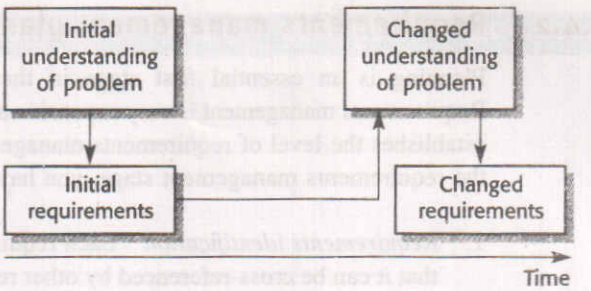


Figure 6.16 Requirements evolution

of the system. For example, in a hospital there will always be requirements concerned with patients, doctors, nurses, treatments, etc. These requirements may be derived from domain models that show the entities and relations which characterise an application domain (Prieto-Díaz and Arango, 1991; Easterbrook, 1993).

2. *Volatile requirements* These are requirements which are likely to change during the system development or after the system has been put into operation. Examples of volatile requirements are requirements resulting from government health-care policies.

Harker *et al.* (1993) have suggested that volatile requirements fall into five classes. However, I think that two of their classes are closely related. I prefer a classification as shown in Figure 6.17.

Figure 6.17 Classification of volatile requirements

Requirement Type	Description
Mutable requirements	Requirements which change because of changes to the environment in which the organisation is operating. For example, in hospital systems, the funding of patient care may change and thus require different treatment information to be collected.
Emergent requirements	Requirements which emerge as the customer's understanding of the system develops during the system development. The design process may reveal new emergent requirements.
Consequential requirements	Requirements which result from the introduction of the computer system. Introducing the computer system may change the organisation's processes and open up new ways of working which generate new system requirements.
Compatibility requirements	Requirements which depend on the particular systems or business processes within an organisation. As these change, the compatibility requirements on the commissioned or delivered system may also have to evolve.

6.4.2 Requirements management planning

Planning is an essential first stage in the requirements management process. Requirements management is very expensive and, for each project, the planning stage establishes the level of requirements management detail which is required. During the requirements management stage, you have to decide on:

1. *Requirements identification* Each requirement must be uniquely identified so that it can be cross-referenced by other requirements and so that it may be used in traceability assessments.
2. *A change management process* This is the set of activities which assess the impact and cost of changes. I discuss it in more detail in the following section.
3. *Traceability policies* These policies define the relationships between requirements and between the requirements and the system design that should be recorded and how these records should be maintained.
4. *CASE tool support* Requirements management involves the processing of large amounts of information about the requirements. Tools which may be used range from specialist requirements management system to spreadsheets and simple database systems.

There are many relationships between requirements and other requirements and between the requirements and the system design. There are also links between requirements and the underlying reasons why these requirements were proposed. When changes are proposed, you have to trace the impact of these changes on other requirements and the system design. Traceability is an overall property of a requirements specification which reflects the ease of finding related requirements.

There are three types of traceability information which may be maintained:

1. *Source traceability information* links the requirements to the stakeholders who proposed the requirements and to the rationale for these requirements. When a change is proposed, this information is used to discover the stakeholders so that they can be consulted about the change.
2. *Requirements traceability information* links dependent requirements within the requirements document. This information is used to assess how many requirements are likely to be affected by a proposed change and the extent of consequential requirements changes which may be necessary.
3. *Design traceability information* links the requirements to the design modules where these requirements are implemented. This information is used to assess the impact of proposed requirements changes on the system design and implementation.

Traceability information is often represented using traceability matrices which relate requirements to stakeholders, each other or design modules. If we consider

Figure 6.18
A traceability matrix

Req. id	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2
1.1		U	R					
1.2			U			R		U
1.3	R			R				
2.1			R		U			U
2.2								U
2.3		R		U				
3.1								R
3.2							R	

traceability matrices which link requirements to other requirements, each requirement is represented by both a row and a column in the matrix. Where a dependency between requirements exist, this is recorded in the cell at the row/column intersection.

This is illustrated in Figure 6.18 which shows a simple traceability matrix which records the dependencies between requirements. A U in the row/column intersection illustrates that the requirement in the row uses the facilities specified in the requirement named in the column; an R means that there is some other weaker relationship between the requirements. For example, they may both define the requirements for parts of the same sub-system.

Traceability matrices may be used when a small number of requirements have to be managed but they become very unwieldy and expensive to maintain for large systems with many requirements. For these systems, you have to capture traceability information in a requirements database where each requirement is explicitly linked to related requirements. The impact of changes can then be assessed by using the database browsing facilities. Alternatively, it may be possible to generate the traceability matrices automatically.

Requirements management needs some automated support and the CASE tools used should be chosen during the planning phase. Tool support is required for:

1. *Requirements storage* The requirements should be maintained in a secure, managed data store which is accessible to everyone involved in the requirements engineering process.

2. *Change management* The process of change management (Figure 6.19) is simplified if active tool support is available.

3. *Traceability management* As discussed above, tool support for traceability allows related requirements to be discovered. Some tools are available which use

natural language processing techniques to help discover possible relationships between the requirements.

For small systems, it may not be necessary to use specialised requirements management tools. The requirements management process may be supported using the facilities available in word processors, spreadsheets and PC databases. However, for larger systems, more specialised tool support is required. I have included links to information about requirements management tools such as DOORS and Requisite Pro in the book's web pages.

6.4.3 Requirements change management

Requirements change management (Figure 6.19) should be applied to all proposed changes to the requirements. The advantage of using a formal process for change management is that all change proposals are treated consistently and that changes to the requirements document are made in a controlled way. There are three principal stages to a change management process:

1. *Problem analysis and change specification* The process starts with an identified requirements problem or, sometimes, with a specific change proposal. During this stage, the problem or the change proposal is analysed to check that it valid. A more specific requirements change proposal may then be made.
2. *Change analysis and costing* The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. The cost of making the change is estimated both in terms of modifications to the requirements document and, if appropriate, to the system design and implementation. Once this analysis is completed, a decision is made whether or not to proceed with the requirements change.
3. *Change implementation* The requirements document and, where necessary, the system design and implementation is modified. The requirements document should be organised so that changes can be accommodated without extensive rewriting. As with programs, changeability in documents is achieved by minimising external references and making the document sections as modular as possible.

Figure 6.19 Requirements change management

If a requirements change to a system is urgently required, there is always a temptation to make that change to the system and then retrospectively modify

the requirements document. This almost inevitably leads to the requirements specification and the system implementation getting out of step. Once system changes have been made, requirements document changes may be forgotten or made in a way that is not consistent with the system changes.

KEY POINTS

- The requirements engineering process includes a feasibility study, requirements elicitation and analysis, requirements specification, requirements validation and requirements management.
- Requirements analysis is an iterative process which involves domain understanding, requirements collection, classification, structuring, prioritisation and validation.
- Different stakeholders in the system have different requirements. All complex systems should therefore be analysed from a number of different viewpoints. Viewpoints may be sources or sinks of data, different system representations or entities which are outside the system and receive services from it.
- Social and organisational factors have a strong influence on system requirements and may determine whether or not the software is actually used.
- Requirements validation is the process of checking the requirements for validity, consistency, completeness, realism and verifiability. Requirements reviews and prototyping are the principal techniques used for requirements validation.
- Business, organisational and technical changes inevitably lead to changes to the requirements for a software system. Requirements management is the process of managing and controlling these changes.
- The requirements management process includes management planning where policies and procedures for requirements management are specified and change management where changes are analysed and their impact assessed.

FURTHER READING

Mastering the Requirements Process. A readable book that is intended for practising requirements engineers. It gives specific guidance on developing an effective requirements engineering process. (S. Robertson and J. Robertson, 1999, Addison Wesley Longman.)

Requirements Engineering: Processes and Techniques. This book includes a more detailed look at the activities in the requirements engineering process and discusses the VORD method and its application. (G. Kotonya and I. Sommerville, 1998, John Wiley and Sons.)

Software Requirements: Objects, Functions and States. This book has a reasonable chapter on analysis in general and includes a good survey of methods of requirements analysis. However, it does not cover approaches to analysis based on multiple viewpoints. (A. M. Davis, 1993, Prentice-Hall.)

Exploring Requirements: Quality before Design. An excellent, very readable book that gives practical advice on the realities of requirements engineering. (D. C. Gause and G. M. Weinberg, 1989, Dorset House.)

EXERCISES

6.1 Suggest who might be stakeholders in a university student records system. Explain why it is almost inevitable that the requirements of different stakeholders will conflict in some ways.

6.2 A software system is to be developed to automate a library catalogue. This system will contain information about all the books in a library and will be usable by library staff and by book borrowers and readers. The system should support catalogue browsing, querying, and should provide facilities allowing users to send messages to library staff reserving a book which is on loan. Identify the principal viewpoints which might be taken into account in the specification of this system and show their relationships using a viewpoint hierarchy diagram.

6.3 For three of the viewpoints identified in the library cataloguing system, suggest services which might be provided to that viewpoint, data which the viewpoint might provide and events which control the delivery of these services.

6.4 For the services identified in Exercise 6.3, identify what might be the most important non-functional constraints.

6.5 Using your own knowledge of how an ATM is used, develop a set of use-cases that could be used to derive the requirements for an ATM system.

6.6 Discuss an example of a type of system where social and political factors might strongly influence the system requirements. Explain why these factors are important in your example.

6.7 Who should be involved in a requirements review? Draw a process model showing how a requirements review might be organised.

6.8 Why do traceability matrices become difficult to manage when there are many system requirements? Design a requirements structuring mechanism, based on viewpoints, which might help reduce the scale of this problem.

6.9 When emergency changes have to be made to systems, the system software may have to be modified before changes to the requirements have been approved. Suggest a model of a process for making these modifications which ensures that the requirements document and the system implementation do not become inconsistent.

6.10 Your company uses a standard analysis method which is normally applied in all requirements analyses. In your work, you find that this method cannot represent social factors which are significant in the system you are analysing. You point this out to your manager who makes clear that the standard should be followed. Discuss what you should do in such a situation.