4. Requirements engineering

Objectives

4.1 Functional and non-functional requirements
4.2 The software requirements document
4.3 Requirements specification
4.4 Requirements engineering processes
4.5 Requirements elicitation and analysis
4.6 Requirements validation
4.7 Requirements management

4.1 Cerințe funcționale și nefuncționale
4.2 Documentul de cerințe software
4.3 Specificația cerințelor
4.4 Procese de inginerie a cerințelor
4.5 Eliberarea și analiza cerințelor
4.6 Validarea cerințelor
4.7 Managementul cerințelor

The requirements for a system are the descriptions of what the system should do—the services that it provides and the constraints on its operation. These requirements reflect the needs of customers for a system that serves a certain purpose such as controlling a device, placing an order, or finding information. The process of finding out, analyzing, documenting and checking these services and constraints is called

requirements engineering (RE).

The term 'requirement' is not used consistently in the software industry. In some cases, a requirement is simply a high-level, abstract statement of a service that a system should provide or a constraint on a system. At the other extreme, it is a detailed, formal definition of a system function. Davis (1993) explains why these differences exist:

If a company wishes to let a contract for a large software development project, it must define its needs in

1. ***a sufficiently* abstract way *that a solution is not predefined*.**

The requirements must be written so that several contractors can bid for the contract, offering, perhaps, different ways of meeting the client organization's needs.

Once a contract has been awarded, the contractor

2. ***must write a system definition for the client in more detail so that the client understands and can validate what the software will do.***

Both of these documents may be called the requirements document for the system.

Some of the problems that arise during the requirements engineering process are a result of failing to **make a clear separation between these different levels of description**

a) **'user requirements'** to mean the high-level abstract requirements and

b) **'system requirements'** to mean the detailed description of what the system should do. User requirements and system requirements may be defined as follows:

1**. User requirements are statements**, in a natural language plus diagrams, of what services the system is expected to provide to system users and the constraints under which it must operate.

2. **System requirements are more detailed descriptions** of the software system's **functions, services, and operational constraints**. The system requirements document (**sometimes called a functional specification**) should define exactly what is to be implemented. It may be part of the **contract between the system buyer and the software developers.**

Different levels of requirements are useful because they communicate information about the system to different types of reader. **Figure 4.1 illustrates the distinction between user and system requirements. This example from a mental health care patient management system (MHC-PMS) shows how a user requirement may be expanded into several system requirements. You can see from Figure 4.1 that the user requirement is quite general.** The system requirements provide more specific information about the services and functions of the system that is to be implemented.

User Requirement Definition

1. The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System Requirements Specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost, and the prescribing clinics shall be generated.
1.2 The system shall automatically generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed, and the total cost of the prescribed drugs.
1.4 If drugs are available in different dose units (e.g., 10 mg, 20 mg) separate reports shall be created for each dose unit.
1.5 Access to all cost reports shall be restricted to authorized users listed on a management access control list.

Figure 4.1 User and system requirements

4.1 Functional and non-functional requirements

Software system requirements are often classified as functional requirements or nonfunctional requirements:

1. *Functional requirements* These are statements of services the system should provide, **how the system should react to particular inputs**, and how the system should behave in particular situations. In some cases, the functional requirements may also explicitly state what the system should not do.

2. Non-functional requirements These are constraints on the services or functions offered by the system. They include **timing constraints, constraints on the development process, and constraints imposed by standards**. Non-functional requirements often apply to the system as a whole, rather than individual system features or services.

4.1.1 Functional requirements

The functional requirements for a system describe what the system should do. These requirements depend on the type of software being developed, the expected users of the software, and the general approach taken by the organization when writing requirements. When expressed as user requirements, functional requirements are usually described in an abstract way that can be understood by system users. However, more specific functional system requirements describe the system functions, its inputs and outputs, exceptions, etc., in detail.
Functional system requirements vary from general requirements covering what the system should do to very specific requirements reflecting local ways of working or an organization's existing systems. For example, here are examples of functional requirements for the MHC-PMS system, used to maintain information about patients receiving treatment for mental health problems:
1. A user shall be able to search the appointments lists for all clinics.
2. The system shall generate each day, for each clinic, a list of patients who are

expected to attend appointments that day.
3. Each staff member using the system shall be uniquely identified by his or her eight-digit employee number.
These functional user requirements

4.1.2 Non-functional requirements

Non-functional requirements, as the name suggests, are requirements that are not directly concerned with the specific services delivered by the system to its users. They may relate to emergent system properties such as reliability, response time, and store occupancy. Alternatively, they may define constraints on the system implementation such as the capabilities of I/O devices or the data representations used in interfaces with other systems.

1. Non-functional requirements **may affect the overall architecture** of a system rather than the individual components. For example, to ensure that performance requirements are met, you may have to organize the system to minimize communications between components.
2. **A single non-functional requirement**, such as a security requirement, **may generate a number of related functional requirements that define new system services** that are required. In addition, it may also generate requirements that restrict existing requirements.
Non-functional requirements arise through user needs, because of budget constraints, organizational policies, **the need for interoperability with other software**

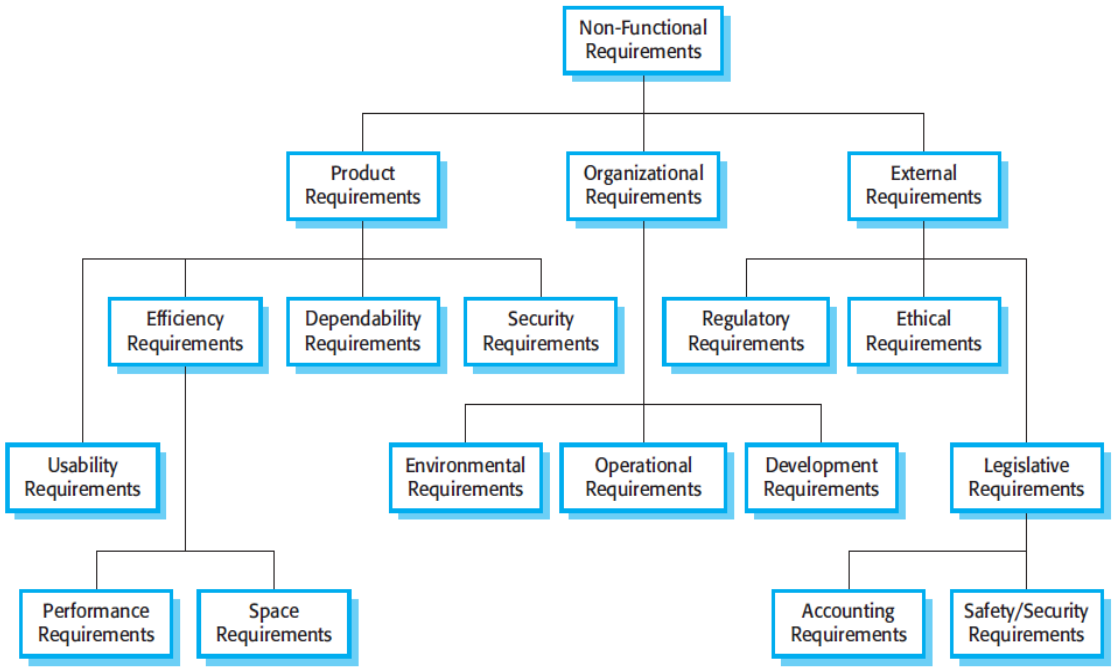

Figure 4.3 Types of non-functional requirement

The system should be easy to use by medical staff and should be organized in such a way that user errors are minimized.

I have rewritten this to show how the goal could be expressed as a 'testable' nonfunctional requirement. It is impossible to objectively verify the system goal, but in the description below you can at least include software instrumentation to count the errors made by users when they are testing the system.

Medical staff shall be able to use all the system functions after four hours of training. After this training, the average number of errors made by experienced users shall not exceed two per hour of system use.

Whenever possible, you should write non-functional requirements quantitatively so that they can be objectively tested.

Figure 4.5 **shows metrics that you can use to specify non-functional system properties. You can measure these characteristics**

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Figure 4.5 Metrics for specifying non-functional requirements

Requirements document standards
A number of large organizations, such as the U.S. Department of Defense and the IEEE, have defined standards for requirements documents. These are usually very generic but are nevertheless useful as a basis for developing more detailed organizational standards. The U.S. Institute of Electrical and Electronic Engineers (IEEE) is one of the best-known standards providers and they have developed a standard for the structure of requirements documents. This standard is most appropriate for systems such as military command and control systems that have a long lifetime and are usually developed by a group of organizations.
http://www.SoftwareEngineering-9.com/Web/Requirements/IEEE-standard.html

4.2 The software requirements document

The software requirements document (**sometimes called the software requirements specification or SRS**) is an official statement of what the system developers should implement. It should include **both the user requirements for a system and a detailed specification of the system requirements**. Sometimes, the user and system requirements are integrated into a single description. In other cases, the user requirements are defined in an introduction to the system requirements specification.

If there are a large number of requirements, the detailed **system requirements may be presented in a separate document**.

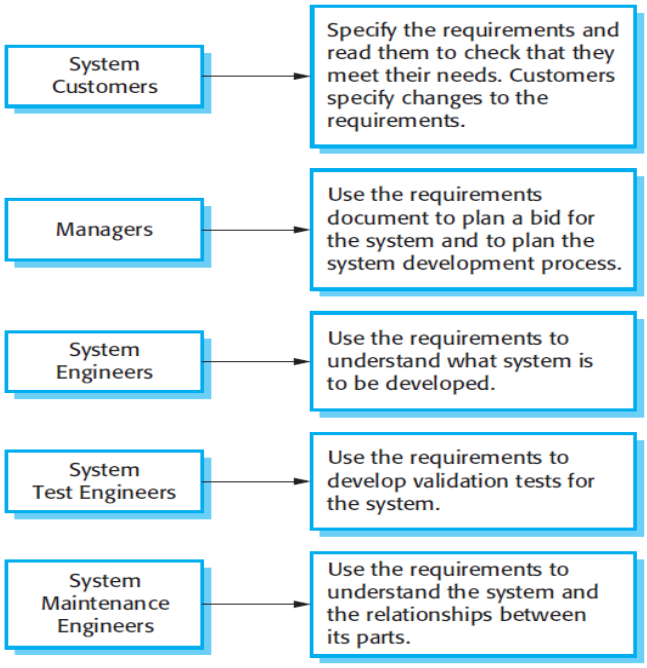

Figure 4.6 Users of a requirements document

Figure 4.7 shows one possible organization for a requirements document that is based on an IEEE **standard for requirements documents (IEEE, 1998**). **This standard is a generic standard that can be adapted to specific uses.** In this case, I have extended the standard to include information about predicted system evolution. This information helps the maintainers of the system and allows designers to include support for future system features.

Naturally, the information that is included in a requirements document depends on the type of software being developed and the approach to development that is to be used. If an evolutionary approach is adopted for a software product (say), the

Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	Here, you describe the services provided for the user. The non-functional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.
System requirements specification	This should describe the functional and non-functional requirements in more detail. If necessary, further detail may also be added to the non-functional requirements. Interfaces to other systems may be defined.
System models	This might include graphical system models showing the relationships between the system components, the system, and its environment. Examples of possible models are object models, data-flow models, or semantic data models.
System evolution	This should describe the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.
Appendices	These should provide detailed, specific information that is related to the application being developed; for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.

Figure 4.7 The structure of a requirements document

4.3 Requirements specification

Requirements specification is :

1. **the process of writing down the user**
2. **and system requirements in a requirements document.**

Ideally, the user and system requirements
should be clear, unambiguous, easy to understand, complete, and consistent. In practice, this is difficult to achieve as stakeholders interpret the requirements in different

ways and there are often inherent conflicts and inconsistencies in the requirements.
The user requirements for a system should describe the functional and nonfunctional requirements so that they are understandable by system users who don't have detailed technical knowledge. Ideally, they should specify only the external behavior of the system. The requirements document should not include details of the system architecture or design. Consequently, if you are writing user requirements, you should not use software jargon, structured notations, or formal notations. You should write user requirements in natural language, with simple tables, forms, and intuitive diagrams.
System requirements are expanded versions of the user requirements that are used by software engineers as the starting point for the system design. They add detail and explain how the user requirements should be provided by the system. They may be used as part of the contract for the implementation of the system and should therefore be a complete and detailed specification of the whole system.
Ideally, the system requirements should simply describe the external behavior of the system and its operational constraints. They should not be concerned with how the system should be designed or implemented. However, at the level of detail required to completely specify a complex software system, it is practically impossible to exclude all design information. There are several reasons for this:
1. You may have to design an initial architecture of the system to help structure the requirements specification. The system requirements are organized according to the different sub-systems that make up the system. As I discuss in Chapters 6 and 18, this architectural definition is essential if you want to reuse software components when implementing the system.
2. In most cases, systems must interoperate with existing systems, which constrain the design and impose requirements on the new system.
3. The use of a specific architecture to satisfy non-functional requirements (such as N-version programming to achieve reliability, discussed in Chapter 13) may be necessary. An external regulator who needs to certify that the system is safe may specify that an already certified architectural design be used.

Notation	Description
Natural language sentences	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Design description languages	This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This approach is now rarely used although it can be useful for interface specifications.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract.

Figure 4.8 Ways of writing a system requirements specification

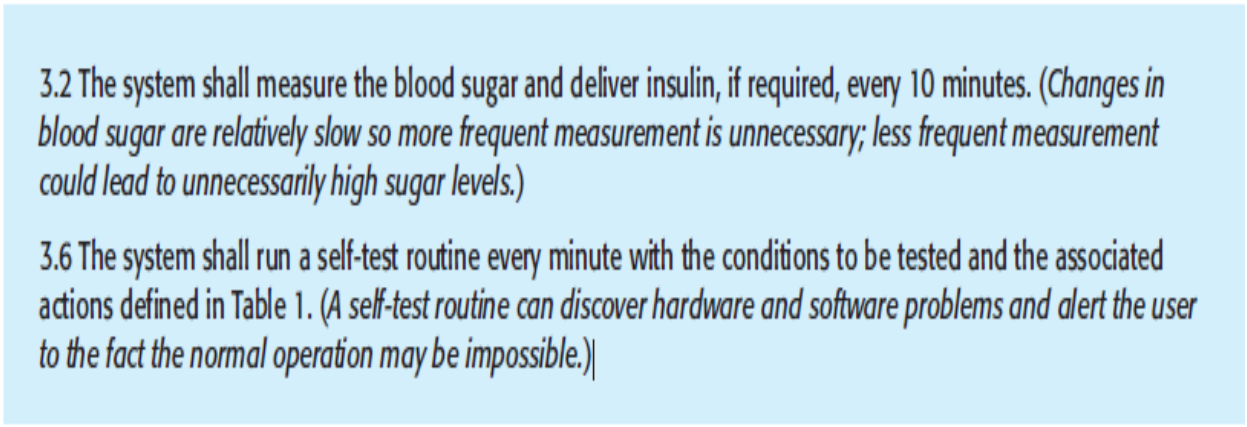

Figure 4.9 Example requirements for the insulin pump software system

4.4 Requirements engineering processes

As I discussed in Chapter 2, requirements engineering processes may include four high-level activities. These focus on assessing if the system is useful to the business (feasibility study), discovering requirements (elicitation and analysis), converting these requirements into some standard form (specification), and checking that the requirements actually define the system that the customer wants (validation). I have shown these as sequential processes in Figure 2.6. However, in practice, requirements engineering is an iterative process in which the activities are interleaved.

Figure 4.12 shows this interleaving. The activities are organized as an iterative process around a spiral, with the output being a system requirements document. The amount of time and effort devoted to each activity in each iteration depends on the stage of the overall process and the type of system being developed. Early in the process, most effort will be spent on understanding high-level business and non-functional requirements, and the user requirements for the system. Later in the process, in the outer rings of the spiral, more effort will be devoted to eliciting and understanding the detailed system requirements.

This spiral model accommodates approaches to development where the requirements are developed to different levels of detail. The number of iterations around the spiral can vary so the spiral can be exited after some or all of the user requirements have been elicited. Agile development can be used instead of prototyping so that the requirements and the system implementation are developed together.

Some people consider requirements engineering to be the process of applying a structured analysis method, such as object-oriented analysis (Larman, 2002). This involves analyzing the system and developing a set of graphical system models, such as use case models, which then serve as a system specification. The set of models describes the behavior of the system and is annotated with additional information describing, for example, the system's 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. Requirements elicitation, in particular, is a human-centered activity and people dislike the constraints imposed on it by rigid system models.

In virtually all systems, requirements change. The people involved develop a better understanding of what they want the software to do; the organization buying the system changes; modifications are made to the system's hardware, software, and organizational environment. The process of managing these changing requirements is called requirements management, which I cover in Section 4.7.

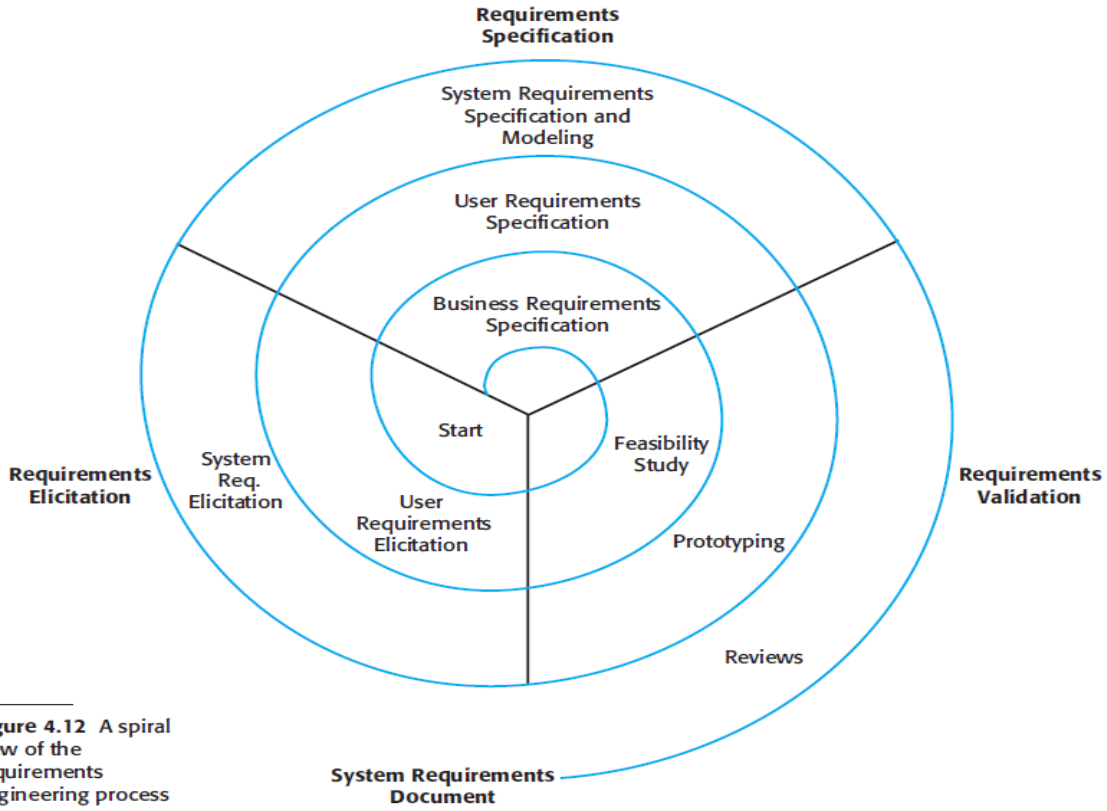

Figure 4.12 A spiral view of the requirements engineering process

4.5 Requirements elicitation and analysis

After an initial feasibility study, the next stage of the requirements engineering process is requirements elicitation and analysis. In this activity, software engineers 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.

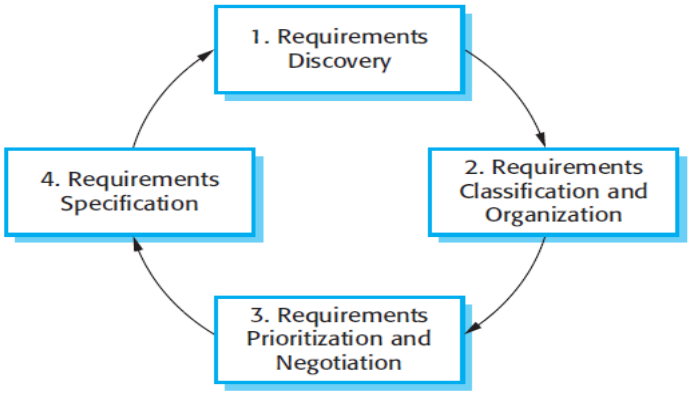

Figure 4.13 The requirements elicitation and analysis process

Requirements elicitation and analysis may involve a variety of different kinds of people in an organization. A system stakeholder is anyone who should have some direct or indirect influence on the system requirements. Stakeholders include endusers who will interact with the system and anyone else in an organization who will be affected by it. Other system stakeholders might be engineers who are developing or maintaining other related systems, business managers, domain experts, and trade union representatives.

A process model of the elicitation and analysis process is shown in Figure 4.13. Each organization 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:

1. ***Requirements discovery*** This is the process of interacting with stakeholders of the system to discover their requirements. Domain requirements from stakeholders and documentation are also discovered during this activity. There are several complementary techniques that can be used for requirements discovery, which I discuss later in this section.

2. Requirements classification and organization This activity takes the unstructured collection of requirements, groups related requirements, and organizes them into coherent clusters. The most common way of grouping requirements is to use a model of the system architecture to identify sub-systems and to associate requirements with each sub-system. In practice, requirements engineering and architectural design cannot be completely separate activities.

3. Requirements prioritization and negotiation Inevitably, when **multiple stakeholders are involved, requirements will conflict**. This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiation. Usually, stakeholders have to meet to resolve differences and agree on compromise requirements.

4. ***Requirements specification*** The requirements are documented and input into the next round of the spiral. Formal or informal requirements documents may be produced, as discussed in Section 4.3.

Figure 4.13 shows that requirements elicitation and analysis is an iterative

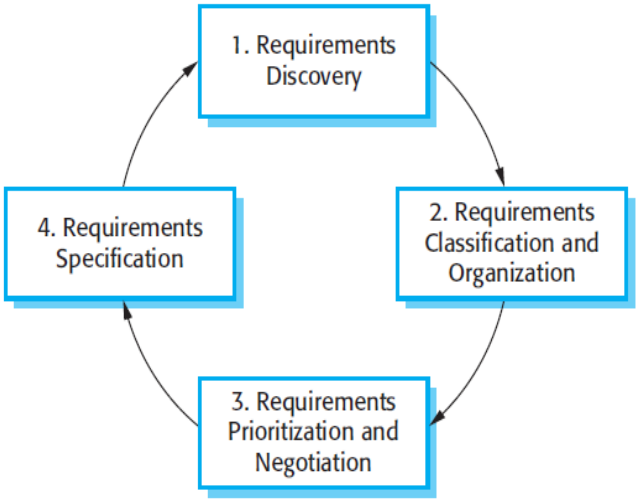

Figure 4.13 The requirements elicitation and analysis process

process with continual feedback from each activity to other activities. The process cycle starts with requirements discovery and ends with the requirements documentation. The analyst's understanding of the requirements improves with each round of the cycle. The cycle ends when the requirements document is complete.

Eliciting and understanding requirements from system stakeholders is a difficult process for several reasons:

1. Stakeholders often don't know what they want from a computer system except in the most general terms; they may find it difficult to articulate what they want the system to do; they may make unrealistic demands because they don't know what is and isn't feasible.

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, may not 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 a system. Managers** may demand specific system requirements because these will allow them to increase their influence in the organization.

5**. The economic and business environment in which the analysis takes place is dynamic**. It inevitably changes during the analysis process. The importance of particular requirements may change. New requirements may emerge from new stakeholders who were not originally consulted.

Inevitably, different stakeholders have different views on the importance and priority of requirements and, sometimes, these views are conflicting. During the process, you should organize regular stakeholder negotiations so that compromises can be reached. It is impossible to completely satisfy every stakeholder but if some stakeholders feel that their views have not been properly considered then they may deliberately attempt to undermine the RE process.

At the requirements specification stage, the requirements that have been elicited so far are documented in such a way that they can be used to help with requirements discovery. At this stage, an early version of the system requirements document may be produced with missing sections and incomplete requirements. Alternatively, the requirements may be documented in a completely different way (e.g., in a spreadsheet or on cards). Writing requirements on cards can be very effective as these are easy for stakeholders to handle, change, and organize.

Viewpoints
A viewpoint is way of collecting and organizing a set of requirements from a group of stakeholders who have something in common. Each viewpoint therefore includes a set of system requirements. **Viewpoints might come from end-users, managers, etc**. They help identify the people who can provide information about their requirements and structure the requirements for analysis.
http://www.SoftwareEngineering-9.com/Web/Requirements/Viewpoints.html

4.6 Requirements validation

Requirements validation is the process of checking that requirements actually define the system that the customer really wants. It overlaps with analysis as it is concerned with finding problems with the requirements. Requirements validation is important because errors in a requirements document can lead to extensive rework costs when these problems are discovered during development or after the system is in service. The cost of fixing a requirements problem by making a system change is usually 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. Furthermore the system must then 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 stakeholders with different needs and **any set of requirements is inevitably a compromise across the stakeholder** 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 that define all functions and the 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 you **should be able to write a set of tests that can demonstrate that the delivered system meets each specified requirement**.

4.7 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—problems that cannot be completely defined. Because the problem cannot be fully defined, the software requirements are bound to be incomplete. **During the software process, the stakeholders' understanding of the problem is constantly changing (Figure 4.17). The system requirements must then also evolve to reflect this changed problem view.**

4.7.1 Requirements management planning

Planning is an essential first stage in the requirements management process. The planning stage establishes the level of requirements management detail that 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 with other requirements and used in traceability assessments.

2. ***A change management process*** This is the **set of activities that assess the impact and cost of changes.**.
3. ***Traceability policies*** These policies define the relationships between each requirement and between the requirements and the system design that should be recorded. The traceability policy should also define how these records should be maintained.
4. *Tool support* Requirements management involves the processing of large amounts of information about the requirements. Tools that may be used range from specialist requirements management systems to spreadsheets and simple database systems.

Requirements management needs automated support and the software tools for this should be chosen during the planning phase. You need tool support for:

1. *Requirements storage* The requirements should be maintained in a secure, managed data store that is accessible to everyone involved in the requirements engineering process.
2. *Change management* The process of change management (Figure 4.18) 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 requirements.

For small systems, it may not be necessary to use specialized 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 specialized tool support is required. I have included links to information about requirements management tools in the book's web pages.

4.7.2 Requirements change management

Requirements change management (Figure 4.18) should be applied to all proposed changes to a system's requirements after the requirements document has been approved. **Change management is essential because you need to decide if the benefits of implementing new requirements are justified by the costs of implementation.**

The advantage of using a formal process for change management is that all change proposals are treated consistently and 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 analyzed to check that it is valid. This analysis is fed back to the change requestor who may respond with a more specific requirements change proposal, or decide to withdraw the request.
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, are modified. You should organize the requirements document so that you can make changes to it without extensive rewriting or reorganization. As with programs, changeability in documents is achieved by minimizing external references and making the document sections

as modular as possible. Thus, individual sections can be changed and replaced without affecting other parts of the document.

If a new requirement has to be urgently implemented, there is always a temptation to change the system and then retrospectively modify the requirements document. You should try to avoid this as it almost inevitably leads to the requirements specification and the system implementation getting out of step. Once system changes have been made, it is easy to forget to include these changes in the requirements document or to add information to the requirements document that is inconsistent with the implementation.

Agile development processes, such as extreme programming, have been designed to cope with requirements that change during the development process. In these processes, when a user proposes a requirements change, this change does not go through a formal change management process. Rather, the user has to prioritize that change and, if it is high priority, decide what system features that were planned for the next iteration should be dropped.

KEY POINTS

- **Requirements for a software system set out what the system should do and define constraintson its operation and implementation.**
- **Functional requirements are statements of the services that the system must provide or are descriptions of how some computations must be carried out.**
- **Non-functional requirements often constrain the system being developed and the development process being used. These might be product requirements, organizational requirements, or external requirements. They often relate to the emergent properties of the system and therefore apply to the system as a whole.**
- **The software requirements document is an agreed statement of the system requirements. It should be organized so that both system customers and software developers can use it.**
- **The requirements engineering process includes a feasibility study, requirements elicitation and analysis, requirements specification, requirements validation, and requirements management.**
- **Requirements elicitation and analysis is an iterative process that can be represented as a spiral of activities—requirements discovery, requirements classification and organization, requirements negotiation, and requirements documentation.**
- **Requirements validation is the process of checking the requirements for validity, consistency,completeness, realism, and verifiability.**
- **Business, organizational, 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.**