Course : Software Engineering and Project Management

Unit 2

Software Requirements Engineering & Analysis

Syllabus

Requirements Engineering : User and system requirements, Functional and non-functional requirements, Types & Metrics, A spiral view of the requirements engineering process.

Software Requirements Specification (SRS): The software requirements Specification document, The structure of SRS, Ways of writing a SRS, structured & tabular SRS for an insulin pump case study,

Requirements elicitation & Analysis: Process, Requirements validation, Requirements management.

Case Studies: The information system case study - Mental health care patient management system (MHC-PMS)

Requirement Engineering

- 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).**

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

- Distinguish between them by using the term **‘user requirements’** to mean the high-level abstract requirements and **‘system requirements’** to mean the detailed description of what the system should do.

- User requirements and system requirements may be defined as follows:

User and system requirements

- **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.

- **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.

- Figure 4.2 shows possible readers of the user and system requirements.

- The readers of the user requirements are not usually concerned with how the system will be implemented and may be managers who are not interested in the detailed facilities of the system.

- The readers of the system requirements need to know more precisely what the system will do because they are concerned with how it will support the business processes or because they are involved in the system implementation.

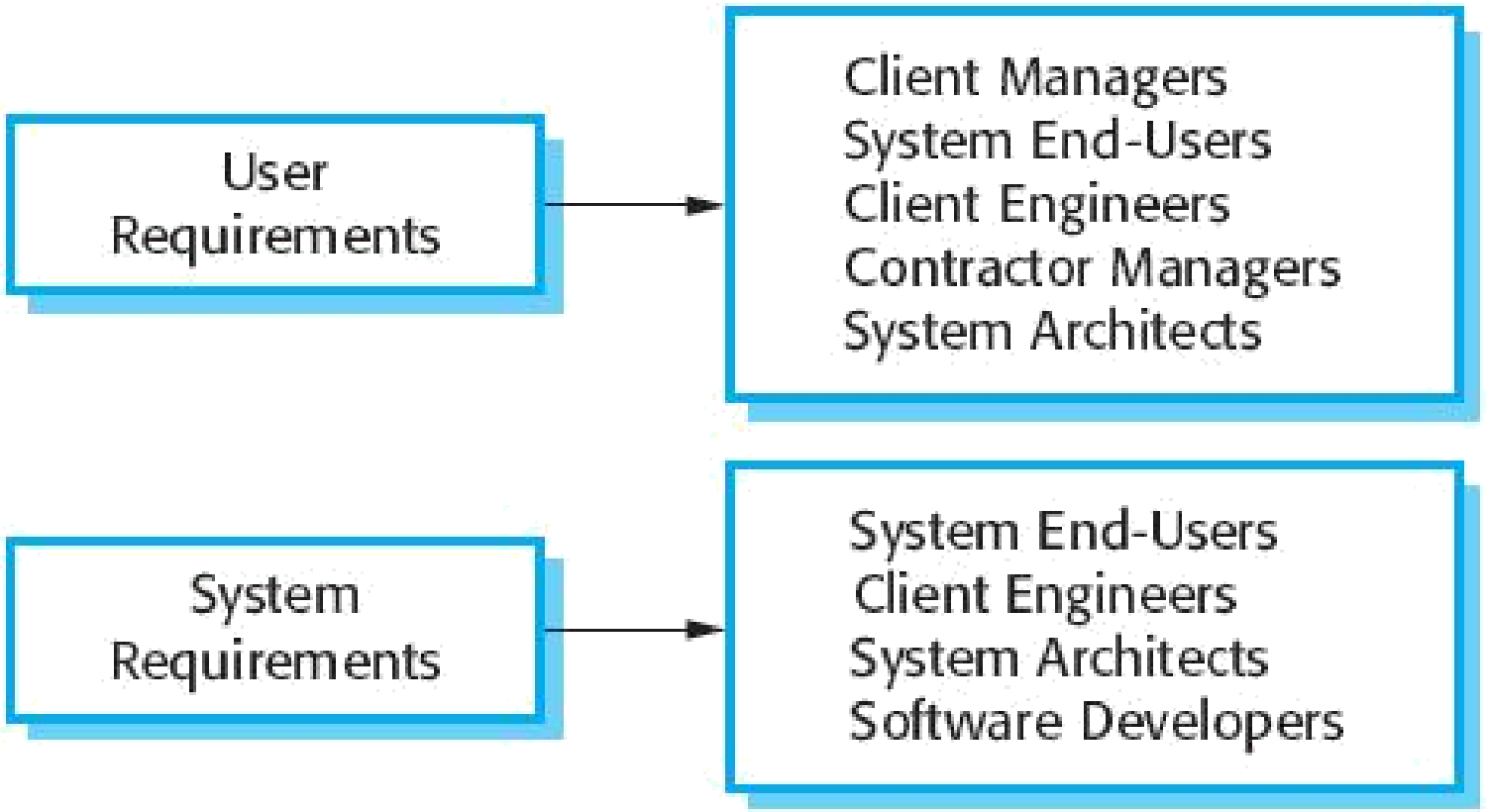

Figure 4.2 :- Readers of different types of requirements specification

Case Study with User & System Requirement

- 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.

- The system requirements provide more specific information about the services and functions of the system that is to be implemented.

Sr. No.	User Requirement Specification
1	The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

Sr. No.	System Requirement Specification
1	On the last working day of each month, a summary of the drugs prescribed, their cost, and the prescribing clinics shall be generated.
2	The system shall automatically generate the report for printing after 17.30 on the last working day of the month.
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.
4	If drugs are available in different dose units (e.g., 10 mg, 20 mg) separate reports shall be created for each dose unit.
5	Access to all cost reports shall be restricted to authorized users listed on a management access control list.

Functional and Non-Functional requirements

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

❖ **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.

❖ **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.

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.

- ✓ 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 define specific facilities to be provided by the system.
- ✓ New requirements have to be established and changes made to the system. Of course, this delays system delivery and increases costs.

- ✓ For example, the first example requirement for the MHC-PMS states that a user shall be able to search the appointments lists for all clinics.
- ✓ The rationale for this requirement is that patients with mental health problems are sometimes confused.

- They may have an appointment at one clinic but actually go to a different clinic. If they have an appointment, they will be recorded as having attended, irrespective of the clinic.

- In principle, the functional requirements specification of a system should be both **complete and consistent.**

- **Completeness** means that all services required by the user should be defined.

- **Consistency** means that requirements should not have contradictory definitions.

- In practice, for large, complex systems, it is practically impossible to achieve requirements consistency and completeness. One reason for this is that it is easy to make mistakes and omissions when writing specifications for complex systems.

- Another reason is that there are many stakeholders in a large system.

- A stakeholder is a person or role that is affected by the system in some way. Stakeholders have different— and often inconsistent— needs.

- These inconsistencies may not be obvious when the requirements are first specified, so inconsistent requirements are included in the specification.

- The problems may only emerge after deeper analysis or after the system has been delivered to the customer.

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.

- Non-functional requirements, such as performance, security, or availability, usually specify or constrain characteristics of the system as a whole.

- Non-functional requirements are often more critical than individual functional requirements.

- However, **failing to meet a non-functional requirement can mean that the whole system is unusable.**

- For example, if an aircraft system does not meet its reliability requirements, it will not be certified as safe for operation; if an embedded control system fails to meet its performance requirements, the control functions will not operate correctly.

- **Non-functional requirements may affect the overall architecture of a system rather than the individual components.**

- A single non-functional requirement, such as a security requirement, may generate a number of related functional requirements.

- **Figure 4.3 is a classification of non-functional requirements.**

- You can see from this diagram that the non-functional requirements may come from required characteristics of the software (**product requirements**), the organization developing the software (**organizational requirements**), or from **external sources**:

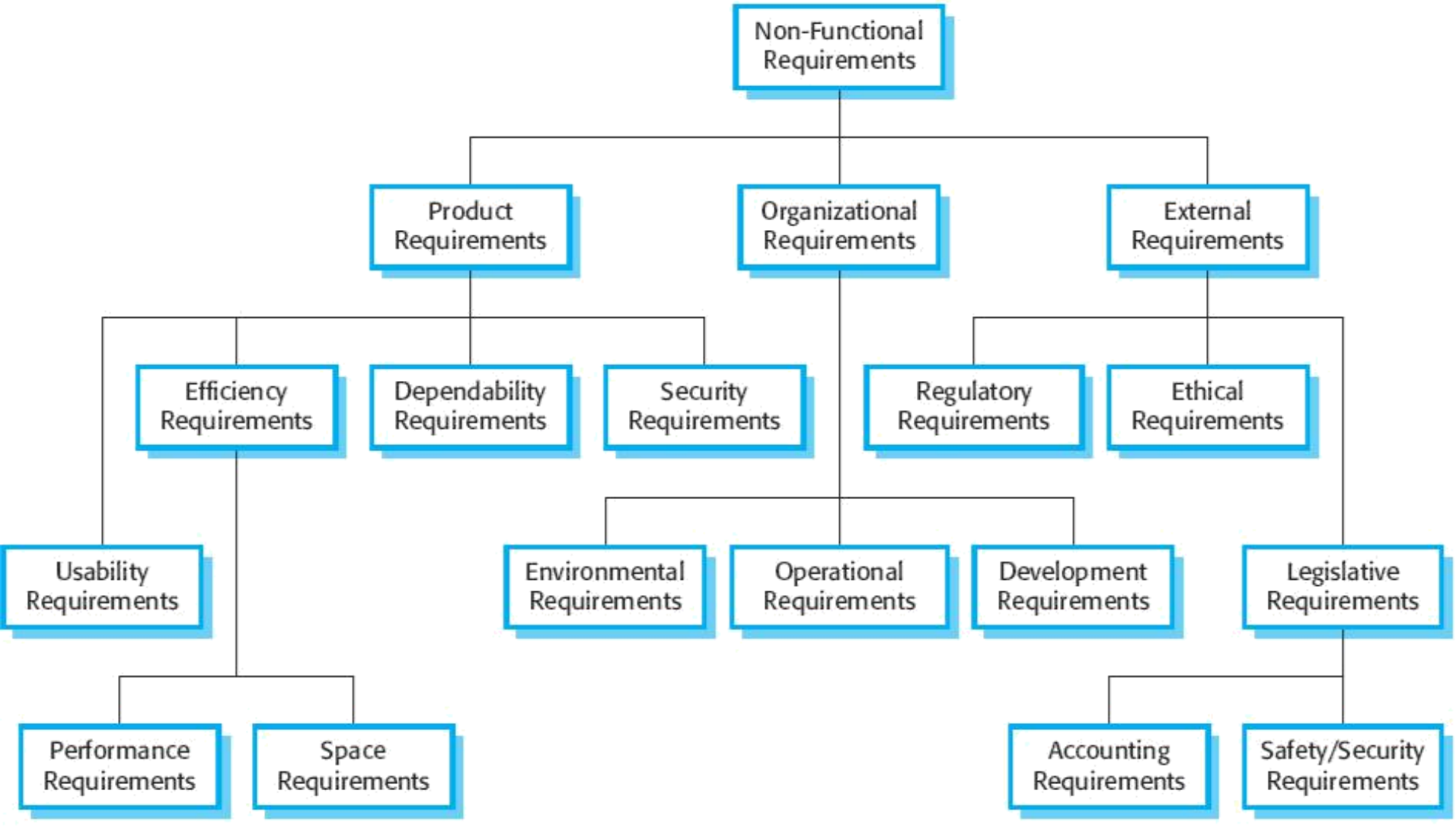

Figure 4.3 :-Types of non-functional requirement

1. **Product requirements :** These requirements specify or constrain the behavior of the software.

✓ Examples include performance requirements on how fast the system must execute and how much memory it requires, reliability requirements that set out the acceptable failure rate, security requirements, and usability requirements.

2. **Organizational requirements :** These requirements are broad system requirements derived from policies and procedures in the customer's and developer's organization.

✓ Examples include operational process requirements that define how the system will be used, development process requirements that specify the programming language, the development environment or process standards to be used, and environmental requirements that specify the operating environment of the system.

3. **External requirements :** This broad heading covers all requirements that are derived from factors external to the system and its development process.

✓ These may include regulatory requirements that set out what must be done for the system to be approved for use by a regulator, such as a central bank; legislative requirements that must be followed to ensure that the system operates within the law; and ethical requirements that ensure that the system will be acceptable to its users and the general public.

✓ **Figure 4.4 shows examples of product, organizational, and external requirements taken from the MHC-PMS.**

PRODUCT REQUIREMENT

The MHC-PMS shall be available to all clinics during normal working hours (Mon–Fri, 08.30–17.30). Downtime within normal working hours shall not exceed five seconds in any one day.

ORGANIZATIONAL REQUIREMENT

Users of the MHC-PMS system shall authenticate themselves using their health authority identity card.

EXTERNAL REQUIREMENT

The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.

Examples of non-functional requirements in the MHC-PMS

- The **Product requirement** is an availability requirement that defines when the system has to be available and the allowed down time each day.

- It says nothing about the functionality of MHC-PMS and clearly identifies a constraint that has to be considered by the system designers.

- The **Organizational requirement** specifies how users authenticate themselves to the system.

- The health authority that operates the system is moving to a standard authentication procedure for all software where, instead of users having a login name, they swipe their identity card through a reader to identify themselves.

- The **External requirement** is derived from the need for the system to conform to privacy legislation.

- Privacy is obviously a very important issue in healthcare systems and the requirement specifies that the system should be developed in accordance with a national privacy standard.

- A common problem with non-functional requirements is that users or customers often propose these requirements as general goals, such as ease of use, the ability of the system to recover from failure, or rapid user response.

- Goals set out good intentions but cause problems for system developers as they leave scope for interpretation and subsequent dispute once the system is delivered.

- For example, the **following system goal is typical of how a manager might express usability requirements:**

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

- We will 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 when the system is being tested to check whether or not the system has met its nonfunctional requirements.

Table 4.5 :- Metrics for specifying non-functional requirements

Sr. No.	Property	Measure
1	**Speed**	Processed transactions/second User/event response time Screen refresh time
2	**Size**	Mbytes Number of ROM chips
3	**Ease of use**	Training time Number of help frames
4	**Reliability**	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
5	**Robustness**	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
6	**Portability**	Percentage of target dependent statements Number of target systems

A spiral view of the requirements engineering process

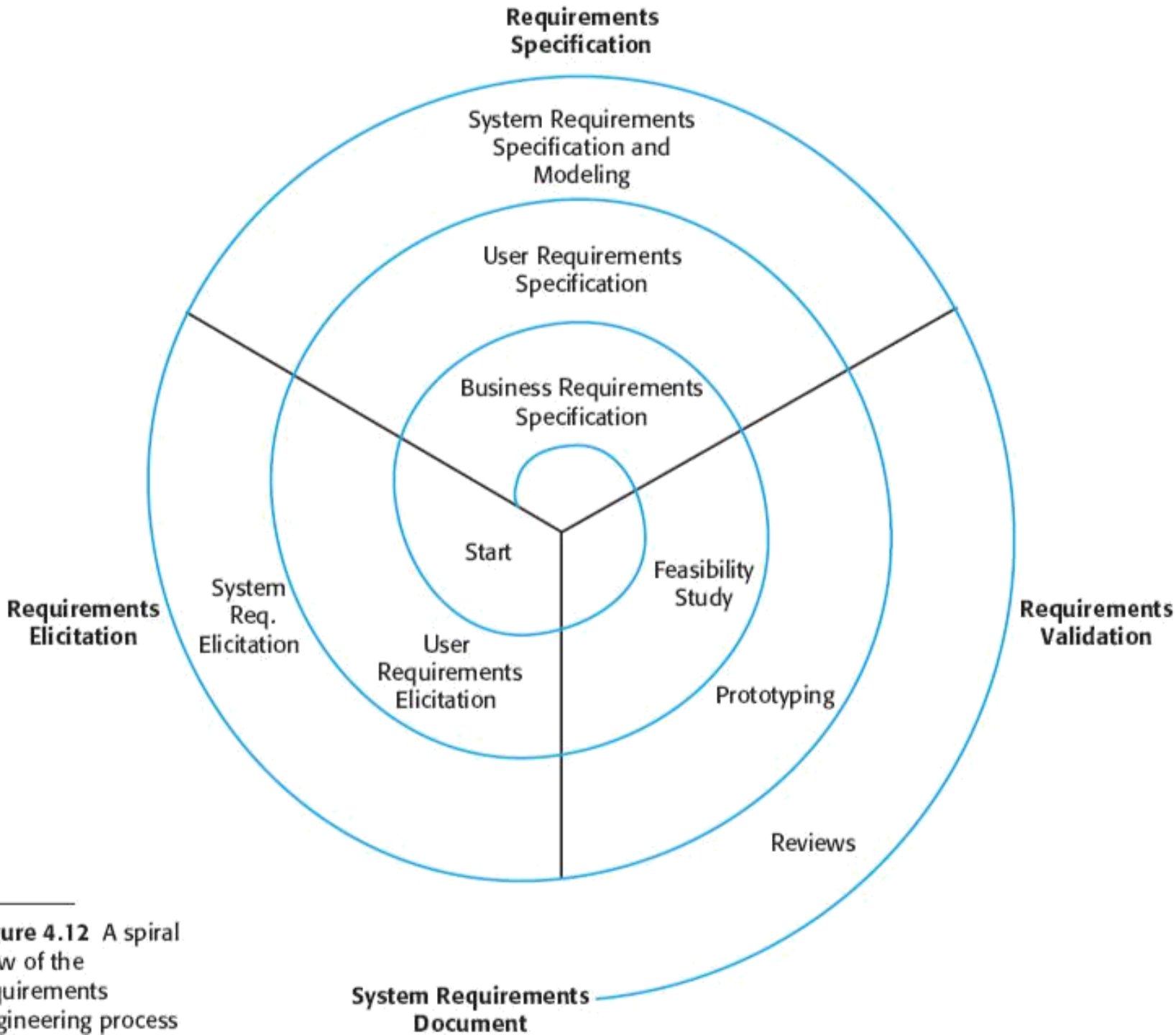

Figure 4.12 A spiral view of the requirements engineering process

- 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).

- 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.

- ***Elicitation is the practice of collecting the requirements of a system from users, customers and other stakeholders.***

- Later in the process, in the **outer rings** of the spiral, more effort will be devoted to eliciting and understanding the detailed system requirements.

- 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.

- (Kindly elaborate all the components which are shown in Spiral diagram in detail)

Software Requirement Specification (SRS) :

The software requirements Specification 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.

- Requirements documents are essential when an outside contractor is developing the software system.

- However, **agile** development methods argue that **requirements change so rapidly** that a requirements document is out of date as soon as it is written, so the effort is largely wasted.

- Rather than a formal document, approaches such as Extreme Programming (Beck, 1999) collect user requirements incrementally and write these on cards as user stories.

- The user then prioritizes requirements for implementation in the next increment of the system.

- For business systems where requirements are unstable, I think that this approach is a good one.

- ✓ It is still useful to write a **short supporting document** that **defines the business and dependability requirements for the system.**

- ✓ The **requirements document has a diverse set of users**, ranging from the senior management of the organization that is paying for the system to the engineers responsible for developing the software.

- ✓ The **level of detail** that you should include in a requirements document depends on the type of system that is being developed and the development process used.

- ✓ **Critical systems need to have detailed requirements because safety and security have to be analyzed in detail.**

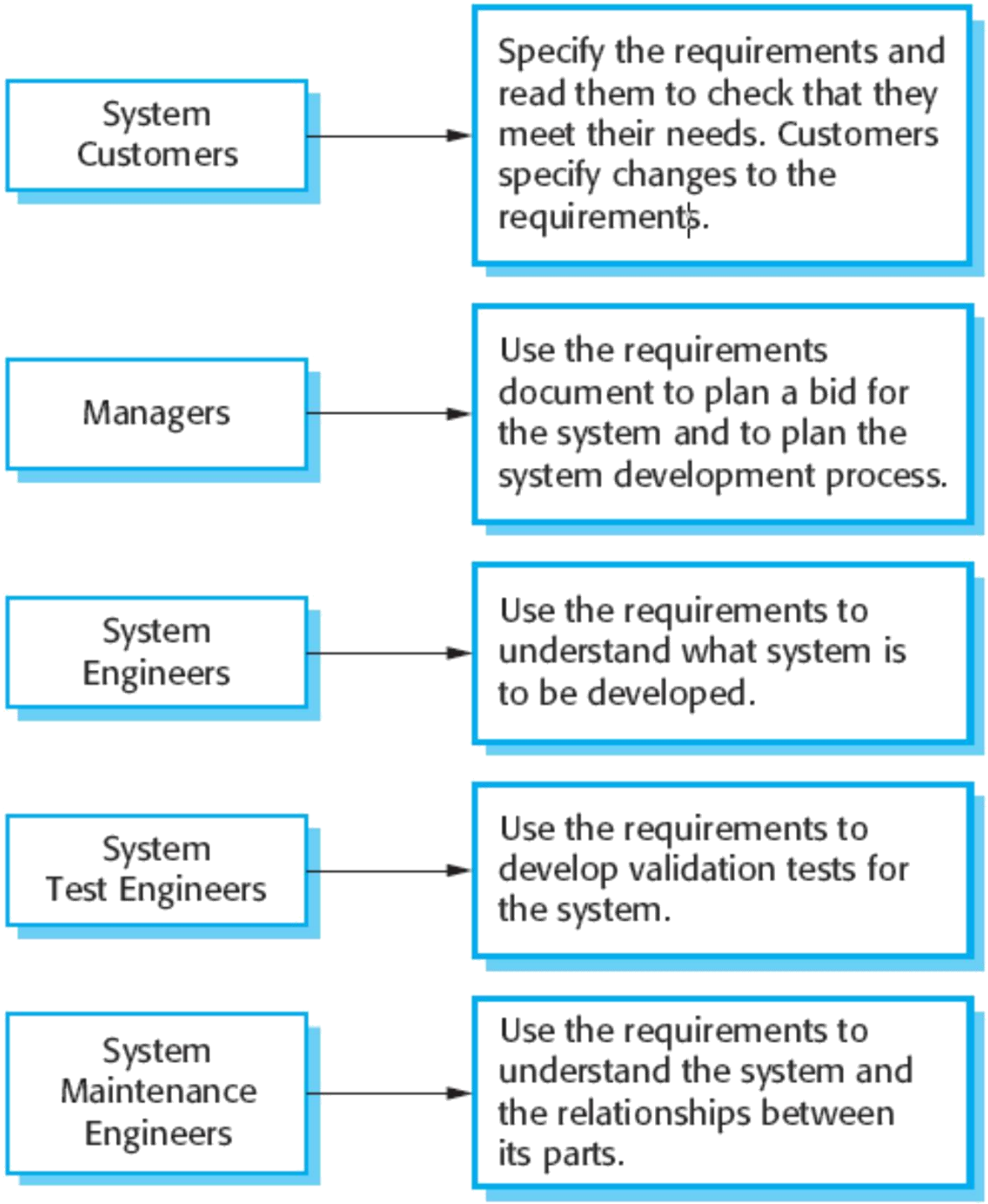

Figure 4.6 : Users of a requirements document

The Structure of SRS

- **Table 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.

- 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.

- **However, when the software is part of a large system project that includes interacting hardware and software systems, it is usually necessary to define the requirements to a fine level of detail.**

Table 4.7 :- The structure of a requirements document (SRS)

Sr. No.	Chapter	Description
1	**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.**
2	**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.
3	**Glossary**	**This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.**

Sr. No.	Chapter	Description
4	**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.
5	**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.**
6	**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.

Sr. No.	Chapter	Description
7	**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.
8	**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.**

Sr. No.	Chapter	Description
9	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.
10	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.**

Ways of writing a SRS

- Requirements specification is the process of writing down the user and system requirements in a requirements document.

- Ideally, **the user and system requirements should be clear, unambiguous, easy to understand, complete, and consistent**.

- 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 (A characteristic language of a particular group), structured notations, or formal notations.
- You should write user requirements in natural language, with simple tables, forms, and intuitive diagrams.
- **User requirements** are almost always written in **natural language** supplemented by appropriate diagrams and tables in the requirements document.
- **System requirements** may also be written in natural language **but other notations based on forms, graphical system models, or mathematical system models can also be used**.
- Table 4.8 summarizes the possible notations that could be used for writing system requirements.

Sr. No.	Notation	Description
1	**Natural language sentences**	The requirements are written using numbered **sentences** in natural language. Each sentence should express one requirement.
2	**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.
3	**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**.
4	**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.

Sr. No.	Notation	Description
5	**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.

Table 4.8 :-Ways of writing a system requirements specification

- Graphical models are most useful when you need to show how a state changes or when you need to describe a sequence of actions.

- Formal mathematical specifications are sometimes used to describe the requirements for safety- or security-critical systems, but are rarely used in other circumstances.

Natural Language Specification

✓ Natural language has been used to write requirements for software since the beginning of software engineering. It is expressive, intuitive, and universal.

✓ To minimize misunderstandings when writing natural language requirements, follow some simple guidelines:

1. Invent a standard format and ensure that all requirement definitions adhere (in accordance) to that format.

2. Use language consistently to distinguish between mandatory and desirable requirements. **Mandatory requirements** are requirements that the system must support and are usually written using **'shall'**. **Desirable requirements** are not essential and are written using **'should'**.

3. Use text highlighting (bold, italic, or color) to pick out key parts of the requirement.

4. Do not assume that readers understand technical software engineering language. It is easy for words like ‘architecture’ and ‘module’ to be misunderstood. You should, therefore, avoid the use of jargon, abbreviations, and acronyms.

5. Whenever possible, you should try to associate a rationale (an explanation of the fundamental reasons) with each user requirement. The rationale should explain why the requirement has been included.

- **The system shall measure the blood sugar & deliver insulin, if required, every 10 minutes.** (Changes in blood sugar are relatively slow so more frequent measurement is unnecessary; less frequent measurement could lead to unnecessarily high sugar levels.)

- The system shall run a self test routine every minute with the conditions to be tested & the associated actions. (A self-test routine can discover hardware & software problems & alert the user to the fact the normal operation may be impossible.)

Structured SRS for an insulin pump case study (Structured Specification)

Fig 4.10 : A structured specification of a requirement for an insulin pump

Function : Compute insulin dose: Safe sugar level.

Description : Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.

Inputs : Current sugar reading (r2), the previous two readings (r0 and r1).

Source : Current sugar reading from sensor. Other readings from memory.

Outputs : CompDose—the dose in insulin to be delivered.

Destination : Main control loop.

Action : CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing.

If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result.

If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.

Requirements : Two previous readings so that the rate of change of sugar level can be computed.

Pre-condition : The insulin reservoir contains at least the maximum allowed single dose of insulin.

Post-condition : r0 is replaced by r1 then r1 is replaced by r2.

Side effects : None.

- Structured natural language is a way of writing system requirements where the freedom of the requirements writer is limited and all requirements are written in a standard way.

- Structured language notations use templates to specify system requirements.

- The specification may use programming language constructs to show alternatives and iteration, and may highlight key elements using shading or different fonts.

- To use a structured approach to specifying system requirements, you define one or more standard templates for requirements and represent these templates as structured forms.

- The specification may be structured around the objects manipulated by the system, the functions performed by the system, or the events processed by the system.

- An example of a form-based specification, in this case, one that defines how to calculate the dose of insulin to be delivered when the blood sugar is within a safe band, is shown in Figure 4.10.

Tabular SRS for an insulin pump case study (Structured Specification)

Table 4.11 : Tabular specification of computation for an insulin pump

Sr. No.	Condition	Action
1	Sugar level falling (r2 < r1)	CompDose = 0
2	Sugar level stable (r2 = r1)	CompDose = 0
3	Sugar level increasing and rate of increase decreasing ((r2 - r1) < (r1 - r0))	CompDose = 0
4	Sugar level increasing and rate of increase stable or increasing ((r2 - r1) >= (r1 - r0))	CompDose = round ((r2 - r1)/4) If rounded result = 0 then CompDose = MinimumDose

- Using structured specifications removes some of the problems of natural language specification.
- Variability in the specification is reduced and requirements are organized more effectively.
- **However, it is still sometimes difficult to write requirements in a clear and unambiguous way, particularly when complex computations (e.g., how to calculate the insulin dose) are to be specified.**
- To address this problem, you can add extra information to natural language requirements, for example, **by using tables or graphical models of the system.**

- These can show how computations proceed, how the system state changes, how users interact with the system, and how sequences of actions are performed.

- ✓ Tables are particularly useful when there are a number of possible alternative situations and you need to describe the actions to be taken for each of these.

- ✓ The insulin pump bases its computations of the insulin requirement on the rate of change of blood sugar levels.

- ✓ The rates of change are computed using the current and previous readings.

- ✓ Figure 4.11 is a tabular description of how the rate of change of blood sugar is used to calculate the amount of insulin to be delivered.

Requirements elicitation & Analysis :-

Process

- 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.

- 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 end users 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:

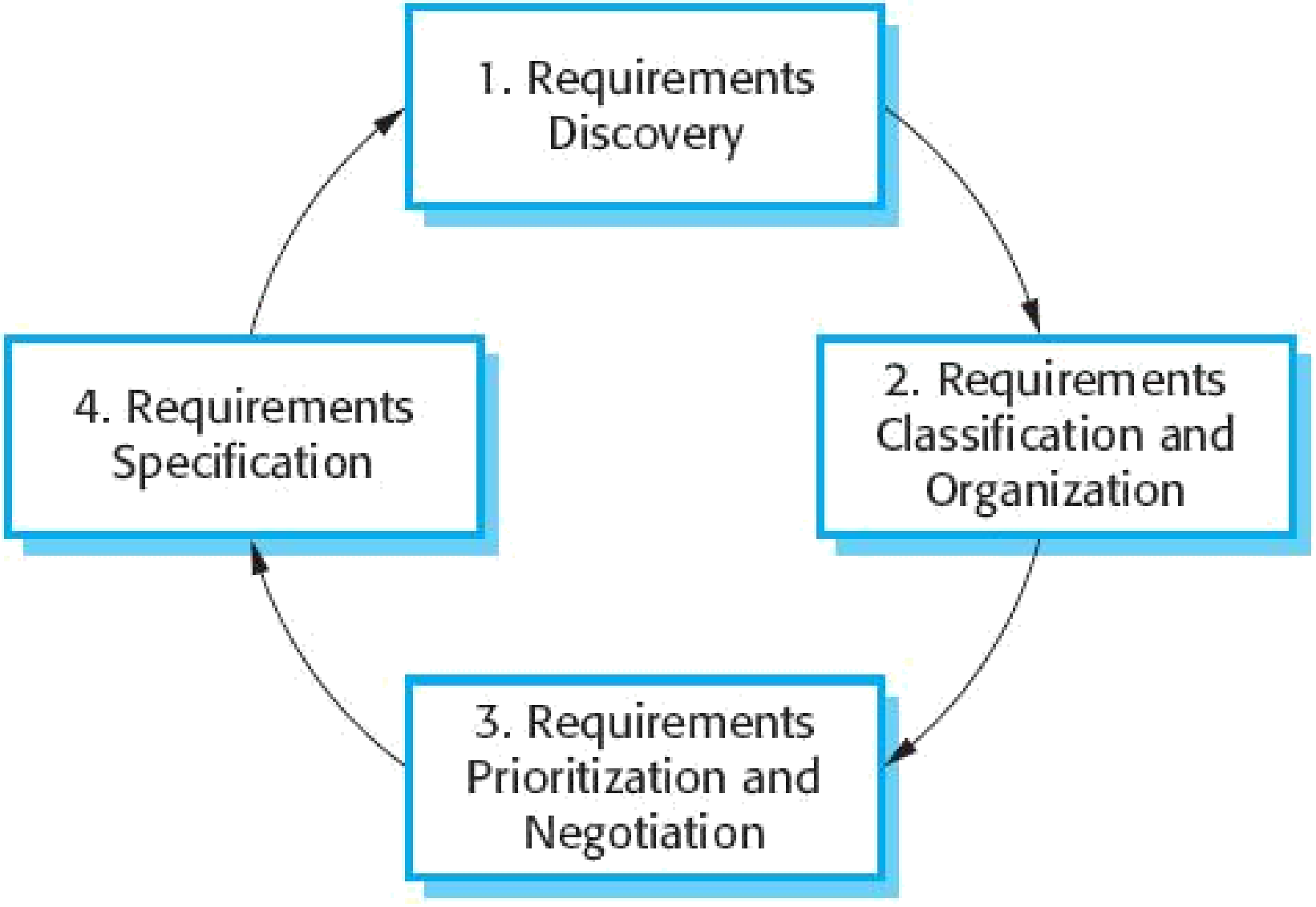

Figure 4.13 : The requirements elicitation and analysis process

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.

2. Requirements classification and organization

- ✓ Domain requirements from stakeholders and documentation are also discovered during this activity.
- ✓ 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.

- Figure 4.13 shows that requirements elicitation and analysis is an iterative 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 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.

Requirements Discovery

- Requirements discovery (sometime called requirements elicitation) is the process of gathering information about the required system and existing systems, and distilling the user and system requirements from this information.

- Sources of information during the requirements discovery phase include documentation, system stakeholders, and specifications of similar systems.

- Stakeholders range from end-users of a system through managers to external stakeholders such as regulators, who certify the acceptability of the system.

- For example, system stakeholders for the mental healthcare patient information system include:

1. **Patients** whose information is recorded in the system.
2. **Doctors** who are responsible for assessing and treating patients.
3. **Nurses** who coordinate the consultations with doctors and administer some treatments.
4. **Medical receptionists** who manage patients' appointments.
5. **IT staff** who are responsible for installing and maintaining the system.
6. A **medical ethics manager** who must ensure that the system meets current ethical guidelines for patient care.
7. **Healthcare managers** who obtain management information from the system.
8. **Medical records staff** who are responsible for ensuring that system information can be maintained and preserved, and that record keeping procedures have been properly implemented.

Interviewing

- Formal or informal interviews with system stakeholders are part of most requirements engineering process.
- In these interviews, the requirement engineering puts questions to stakeholders about the system that they currently use & the system to be developed.
- Requirements are designed from the answers to these questions.
- Interviews may be of two types :-
- **Closed Interviews**, where the stakeholders answers a pre-defined set of questions.
- **Open interviews**, in which there is no pre-defined agenda. The requirements engineering team explores a range of issues with system stakeholders & hence develop a better understanding of their needs.

- In practice, interview with stakeholders are normally a mixture of both of these.
- Completely open ended discussions rarely work well.
- You usually ask some questions to get started and keep interview focused on the system to be developed.
- Interviews are good for getting an overall understanding of wehat stakeholders do, how they might interact with the new system, & the difficulties that they face with current systems.
- People like talking about their work so are usually happy to get involved in interview.
- However, interviews are not so helpful in understanding the requirements form the application domain.

Scenarios

- ✓ People usually find it easier to relate real-life example rather than abstract descriptions. They can understand & criticize a scenario of how they might interact with a software system.
- ✓ Scenarios can be particularly useful for adding to an outline requirements description.
- ✓ They are descriptions of example interaction sessions.
- ✓ Each scenario usually covers one or a small number of possible interactions.
- ✓ Different forms of scenarios are developed & they provide different types of information at different levels of detail about the system.
- ✓ The stories used in extreme programming are a type of requirement scenario.
- ✓ A Scenario starts with an outline of the interaction.

- ✓ During the elicitation process, details are added to this to create a complete description of that interaction.
- ✓ At its most general, a scenario may include :-

1. A description of what the system & users expects when the scenario starts.
2. A description of the normal flow of events in the scenario.
3. A description of what can go wrong & how this is handled.
4. Information about other activities that might be going on at the same time.
5. A description of the system state when the scenario finishes.

- ✓ Scenario-based elicitation involves working with stakeholders to identify scenarios & to capture details to be included in these scenarios.

- Scenarios may be written as text, supplemented by diagrams, screen shots, etc.
- A more structured approach such as event scenarios or use cases may be used.
- As an example of a simple text scenario, consider how the MHC-PMS may be used to enter data for a new patient (**as below**).
- When a new patient attends a clinic, a new record is created by a medical receptionist & personal information (name, age, etc..) is added to it.
- A nurse then interview the patient & collects medical history.
- The patient then has an initial consultation with a doctor who makes a diagnosis and, if appropriate, recommends a course of treatment.
- The scenario shows what happens when medical is collected.

Scenario for collecting medical History in MHC-PMS

Initial Assumption :-

- ✓ The patient has seen a medical receptionist who has created a record in the system & collected the patients personal information (name, address, age, etc…). A nurse is logged on to the system & is collecting medical history.

Normal :-

- ✓ The nurse searches for the patient by family name. If there is more than one patient with the same surname, the given name (first name in English) & date of birth are used to identify the patient.
- ✓ The nurse chooses the menu option to add medical history.
- ✓ The nurse then follows a series of prompts from the system to enter information about consultations elsewhere on mental health problems (free text input), existing medical conditions (nurse selects

- conditions from menu) , medication currently taken (selected form menu), allergies (free text), & home life (form).

What can go wrong :-

- The patient's record does not exist or cannot be found. The nurse should create a new record & record personal information.

- Patient conditions or medication are not entered in the menu. The nurse should choose the 'other' option & enter free text describing the condition / medication.

- Patient cannot/will not provide information on medical history.

- The nurse should enter free text recording the patient's inability/unwillingness to provide information.

- The system should print standard exclusion form stating that the lack of information may mean that treatment will be limited or delayed. This should be signed and handed to the patient.

Other Activities :-

- ✓ Record may be consulted but not edited by other staff while information is being entered.

System State on Completion :-

- ✓ User is logged on.
- ✓ The patient record including medical history is entered in the database, a record is added to the system log showing the start & end time of the session & the nurse involved.

Use Cases

- ✓ Use case now become a fundamental feature of the unified modeling language.
- ✓ A use case identifies the actors involved in an interaction & names the type of interaction.
- ✓ This is then supplemented by additional information describing the interaction with the system.
- ✓ The additional information may be a textual description or more graphical models such as UML sequence or state charts.
- ✓ Use case are documented using a high-level use case diagram.
- ✓ The set of use case represents all of the possible interactions that will be described in the system requirements.
- ✓ **Actors** in the process, who may be the human or other systems, are represented as Stick Figures.

- Each class of interaction is represented as a named **eclipse**.
- **Lines** link the actors with interaction.
- Optionally, Arrowheads ay be added to lines to show how the interaction is initiated.
- Figure 15, which shows some of the use cases for the patient information system.
- Use cases identify the individual interactions between the system and its users or other systems. Each use case should be documented with a textual description.
- These can then be linked to other models in the UML that will develop the scenario in more detail.
- For example, a brief description of the Setup Consultation use case from Figure 4.15 might be:

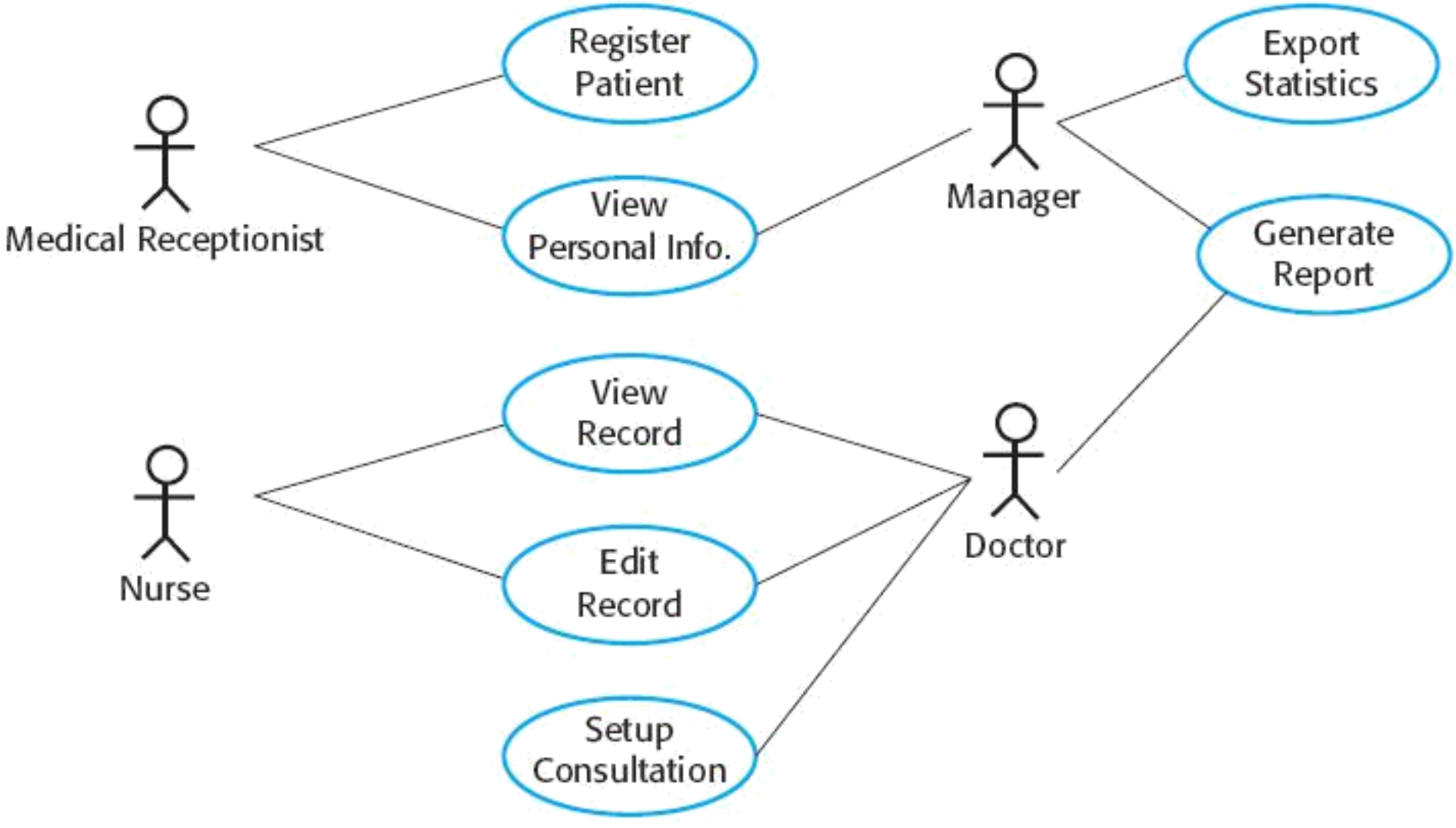

Figure 4.15 Use cases for the MHC-PMS

- *Setup consultation allows two or more doctors, working in different offices, to view the same record at the same time.*

- *One doctor initiates the consultation by choosing the people involved from a drop-down menu of doctors who are online.*

- *The patient record is then displayed on their screens but only the initiating doctor can edit the record.*

- *In addition, a text chat window is created to help coordinate actions.*

- *It is assumed that a phone conference for voice communication will be separately set up.*

- ✓ Scenarios and use cases are effective techniques for eliciting requirements from stakeholders who interact directly with the system.
- ✓ Each type of interaction can be represented as a use case.
- ✓ However, because they focus on interactions with the system, they are not as effective for eliciting constraints or high-level business and nonfunctional requirements or for discovering domain requirements.

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.

- 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.

- There are a number of **requirements validation techniques** that can be used individually or in conjunction with one another:

1. Requirements reviews

- The requirements are analyzed systematically by a team of reviewers who check for errors and inconsistencies.

2. Requirements reviews

- In this approach to validation, an executable model of the system in question is demonstrated to end-users and customers.
- They can experiment with this model to see if it meets their real needs.

3. Test-case generation

- ✓ Requirements should 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.
- ✓ Developing tests from the user requirements before any code is written is an integral part of extreme programming.

- Ultimately, it is difficult to show that a set of requirements does in fact meet a user's needs.

- It is hard even for skilled computer professionals to perform this type of abstract analysis and harder still for system users.

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.

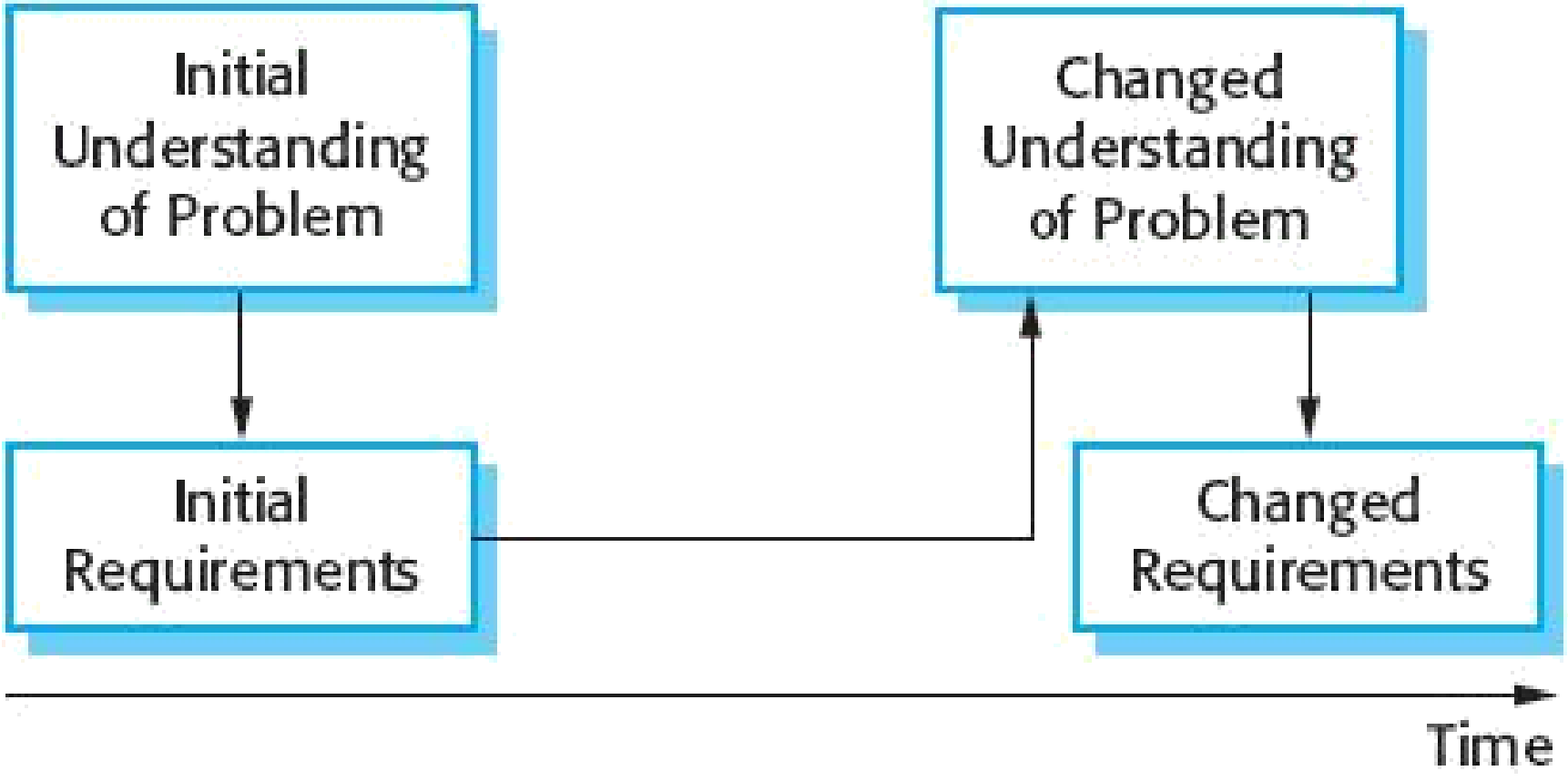

Figure 4.17 : Requirements evolution

- ✓ Once a system has been installed and is regularly used, new requirements inevitably emerge.

- ✓ It is hard for users and system customers to anticipate what effects the new system will have on their business processes and the way that work is done.

- ✓ Once end-users have experience of a system, they will discover new needs and priorities.

- ✓ **<u>There are several reasons why change is inevitable :</u>**

1. The business and technical environment of the system always changes after installation. New hardware may be introduced, it may be necessary to interface the system with other systems, business priorities may change.

2. The people who pay for a system and the users of that system are rarely the same people. New features may have to be added for user support if the system is to meet its goals.

3. Large systems usually have a diverse user community, with many users having different requirements and priorities that may be conflicting or contradictory.

- ✓ Requirements management is the process of understanding and controlling changes to system requirements.
- ✓ You need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes.
- ✓ You need to establish a formal process for making change proposals and linking these to system requirements.

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.

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.

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.

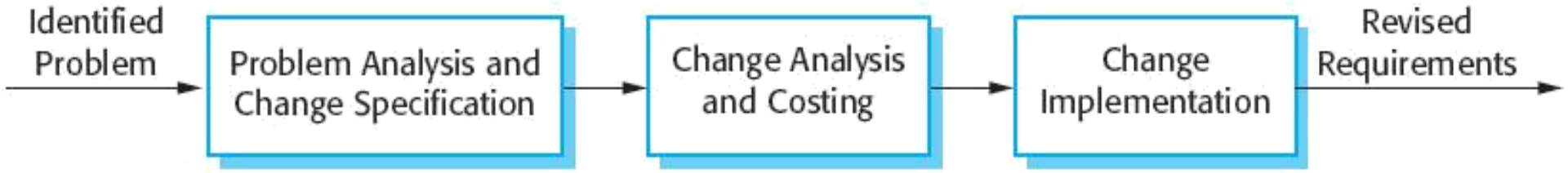

Figure 4.18 : Requirements change management

✓ 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.

- 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.

References

1. Ian Sommerville, “ Software Engineering”, Addison and Wesley, ISBN 0-13-703515-2