3rd Unit

6 Software requirements

Contents

Requirement Engineering?

The requirements for a system are the descriptions of the services provided by the system and its operational constraints.

The process of finding out, analysing, documenting and checking these services and constraints is called requirement engi*neering* (RE).

Requirement?

The requirements for a system are the descriptions of the services provided by the system and its operational constraints. It may range from a high level abstract statement of a service of a system constraint to a detailed mathematical functional specification.

Types of requirement

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

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

Figure 6.1 illustrates the distinction between user and system requirements.

From Figure 6.1 the user requirement is more abstract, and the system requirements add detail, explaining the services and functions that should be provided by the system to be developed.

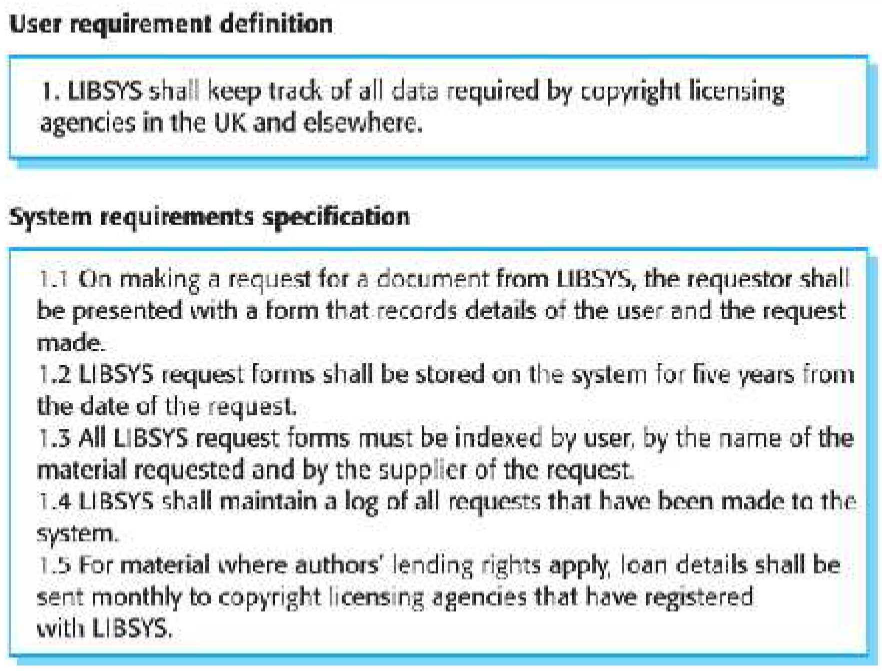


Figure 6.1 User and system requirements

User requirement definition

1. LIBSYS shall keep track of all data required by copyright licensing agencies in the UK and elsewhere.

System requirements specification

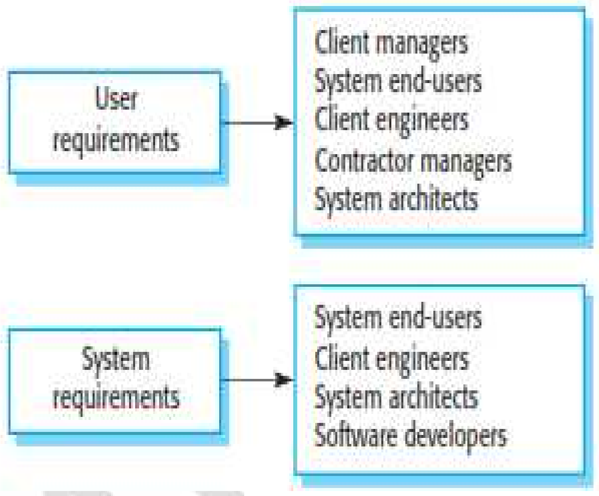
1.1 On making a request for a document from LIBSYS, the requestor shall be presented with a form that records details of the user and the request made.
1.2 LIBSYS request forms shall be stored on the system for five years from the date of the request.
1.3 All LIBSYS request forms must be indexed by user, by the name of the material requested and by the supplier of the request.
1.4 LIBSYS shall maintain a log of all requests that have been made to the system.
1.5 For material where authors' lending rights apply, loan details shall be sent monthly to copyright licensing agencies that have registered with LIBSYS.

Note: write any three system requirements

Figure 6.2 shows the types of readers for the user and system requirements

1. 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.
2. 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.

Figure 6.2 Readers of different types of specification



6.1 Functional and non-functional requirements

Software system requirements are often classified as functional requirements, nonfunctional requirements or domain 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 standards.

Non-functional requirements often apply to the system as a whole. They do not usually just apply to individual system features or services.

3. *Domain requirements* These are requirements that come from the application domain of the system and that reflect characteristics and constraints of that domain.

6.1.1 Functional requirements

1. The functional requirements for a system describe what the system should do.

2. These requirements depend on the type of software being developed, the expected users of the software and the general approach taken by the organisation when writing requirements.

3. Functional system requirements describe the system function in detail, its inputs and outputs, exceptions, and so on.

4. **Functional requirements Example**

Here are examples of functional requirements for a university library system called LIBSYS, used by students and faculty to order books and documents from other libraries.

a. The user shall be able to search either all of the initial set of databases or select a subset from it.

b. The system shall provide appropriate viewers for the user to read documents in the document store.

c. Every order shall be allocated a unique identifier (ORDER_ID), which the user shall be able to copy to the account's permanent storage area.

-The LIBSYS system is a single interface to a range of article databases. It allows users to download copies of published articles in magazines, newspapers and scientific journals.

-Consider the second example requirement for the library system that refers to 'appropriate viewers' provided by the system. The library system can deliver documents in a range of formats.

5. 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.

6.1.2 Non-functional requirements

1. These are constraints on the services or functions offered by the system. They include timing constraints, constraints on the development process and standards.

2. 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 a real-time control system fails to meet its performance requirements, the control functions will not operate correctly.

3. Non-functional requirements arise through user needs, because of budget constraints, because of organisational policies, because of the need for interoperability with other software or hardware systems, or because of external factors such as safety regulations or privacy legislation.

Figure 6.3 is a classification of non-functional requirements.

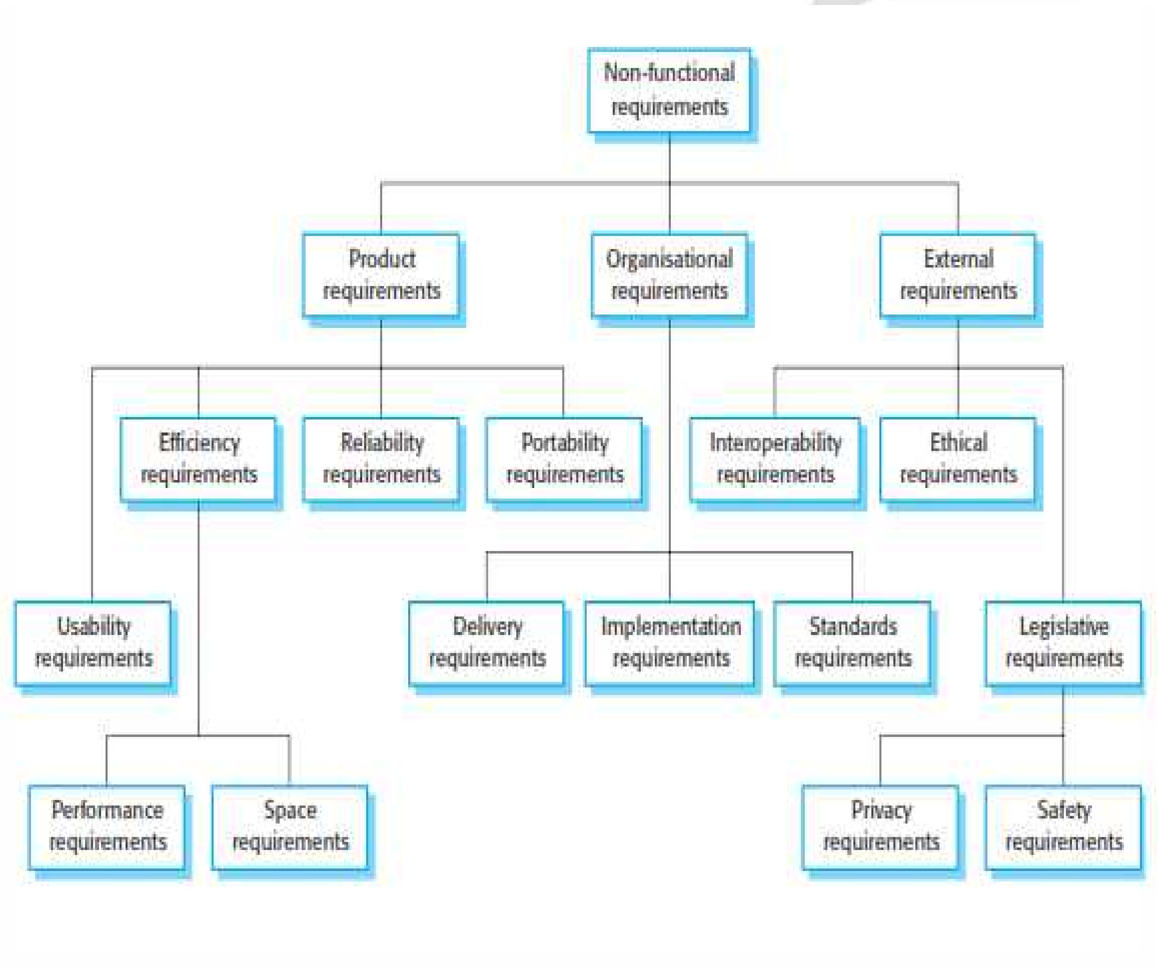


Figure 6.3 Types of non-functional requirements

4. The types of non-functional requirements are:

a. ***Product requirements*** These requirements specify product behaviour.

Examples : performance requirements on how fast the system must execute and how much memory it requires; reliability requirements that set out the acceptable failure rate; portability requirements; and usability requirements.

b. *Organisational requirements* These requirements are derived from policies and procedures in the customer's and developer's organisation.

Examples process standards that must be used; implementation requirements such as the programming language or design method used;

c. *External requirements* These requirements are derived from factors external to the system and its development process.

Examples interoperability requirements that define how the system interacts with systems in other organizations.

5-Non-functional Requirement Example

Figure 6.4 shows examples of product, organisational and external requirements taken from the library system LIBSYS

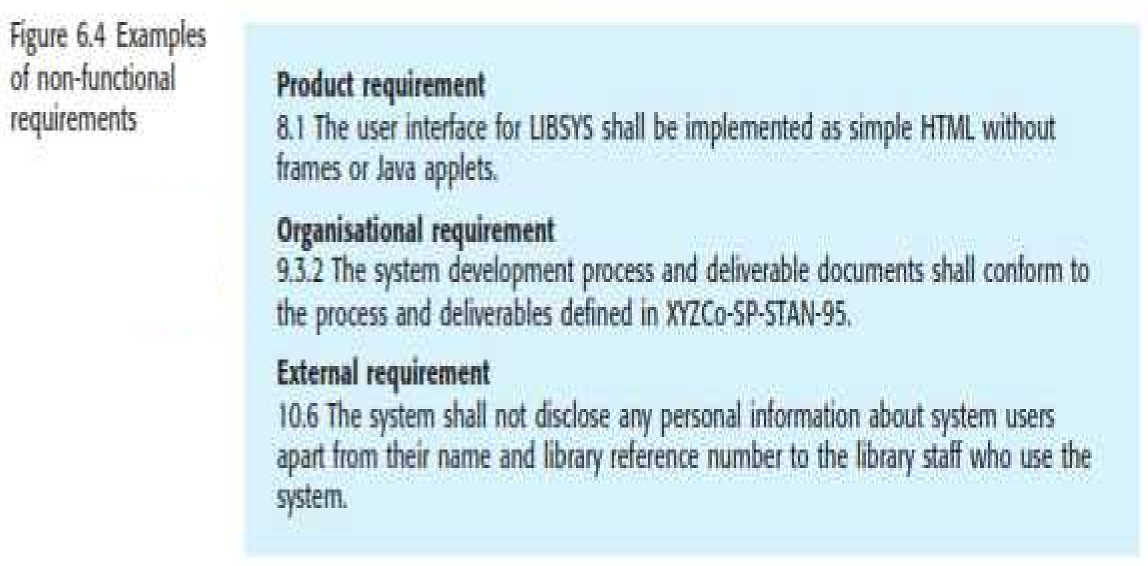


Figure 6.4 Examples of non-functional requirements

Product requirement
8.1 The user interface for LIBSYS shall be implemented as simple HTML without frames or Java applets.

Organisational requirement
9.3.2 The system development process and deliverable documents shall conform to the process and deliverables defined in XYZCo-SP-STAN-95.

External requirement
10.6 The system shall not disclose any personal information about system users apart from their name and library reference number to the library staff who use the system.

6-Goals and verifiable Requirements –

A common problem with non-functional requirements is that they can be difficult

to verify. Users or customers intention is ease of use, the ability of the system to recover from failure.

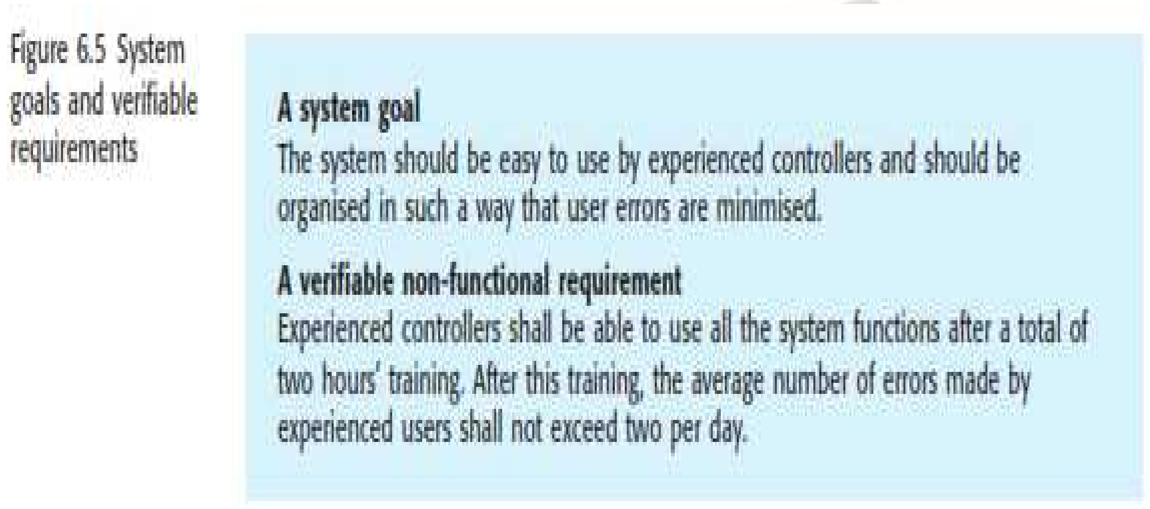


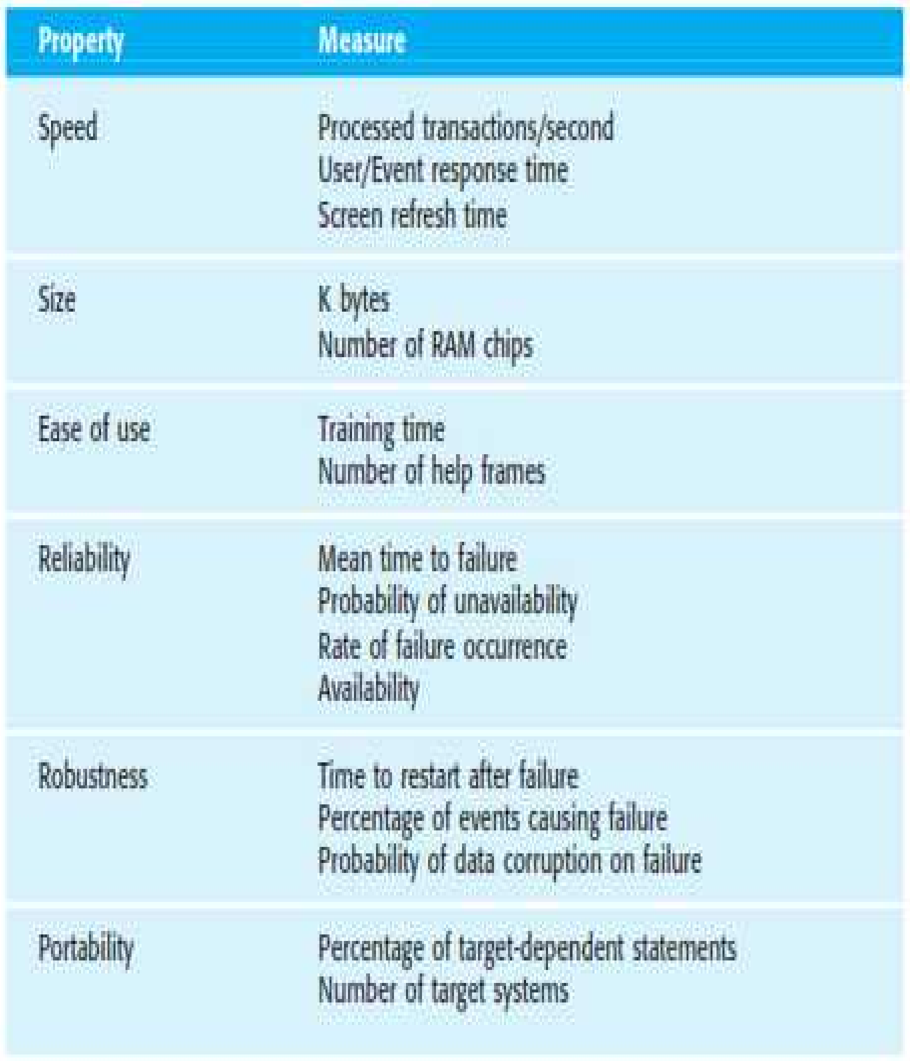
Figure 6.5 System goals and verifiable requirements

A system goal
The system should be easy to use by experienced controllers and should be organised in such a way that user errors are minimised.

A verifiable non-functional requirement
Experienced controllers shall be able to use all the system functions after a total of two hours' training. After this training, the average number of errors made by experienced users shall not exceed two per day.

-Figure 6.6 shows a number of possible metrics that can be used to specify non functional system properties, to measure whether or not the system has met its non-functional requirements.

Figure 6.6 Metrics for specifying non-functional requirements



Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	K bytes Number of RAM 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

6.1.3 Domain requirements

1. Domain requirements are derived from the application domain of the system rather than from the specific needs of system users. They usually include specialised domain terminology or reference to domain concepts.

2. Domain requirements are important because they often reflect fundamentals of the application domain. If these requirements are not satisfied, it may be impossible to make the system work satisfactorily.

3. Example:

The LIBSYS system includes a number of domain requirements:

a. There shall be a standard user interface to all databases that shall be based on the Z39.50 standard.
b. Because of copyright restrictions, some documents must be deleted immediately on arrival.

-The first requirement is a design constraint. It specifies that the user interface to the database must be implemented according to a specific library standard.

-The second requirement has been introduced because of copyright laws that apply to material used in libraries. It specifies that the system must include an automatic delete-on-print facility for some classes of document.

4. Major problems

a. They are written in the language of the application domain and it is often difficult for software engineers to understand them.

b. Domain experts may leave information out of a requirement simply because it is so obvious to them. However, it may not be obvious to the developers of the system, and they may therefore implement the requirement in the wrong way.

6.2 User requirements

1. The user requirements for a system should describe the functional and nonfunctional requirements so that they are understandable by system users without detailed technical knowledge (you should not use software jargon, structured notations or formal notations, or describe the requirement by describing the system implementation. You should write user requirements in simple language, with simple tables and forms and intuitive diagrams).

2. They should only specify the external behavior of the system and should avoid, as far as possible, system design characteristics.

However, various problems can arise when requirements are written in natural language sentences in a text document:

a. *Lack of clarity* It is sometimes difficult to use language in a precise and unambiguous way without making the document wordy and difficult to read.

b.*Requirements confusion* Functional requirements, non-functional requirements, system goals and design information may not be clearly distinguished.

c. *Requirements amalgamation* together as a single requirement.

3. As an illustration of some of these problems, consider one of the requirements for the library shown in Figure 6.8.

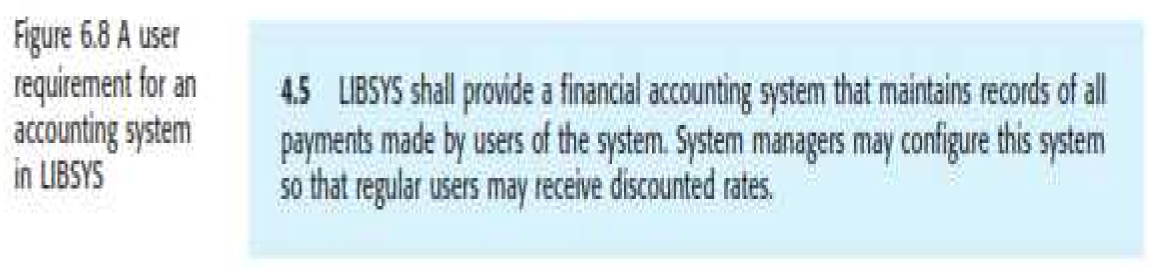


Figure 6.8 A user requirement for an accounting system in LIBSYS

4.5 LIBSYS shall provide a financial accounting system that maintains records of all payments made by users of the system. System managers may configure this system so that regular users may receive discounted rates.

A). This requirement includes both conceptual and detailed information.

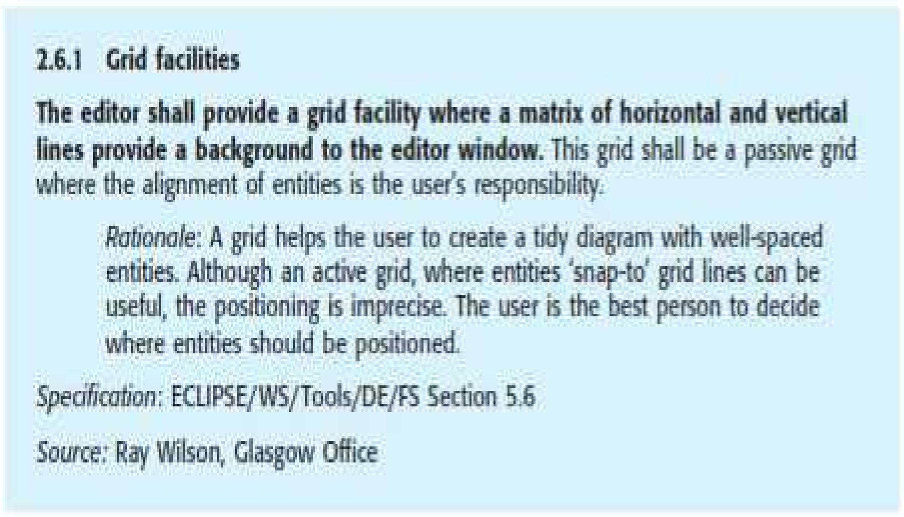
the conceptual- that there should be an accounting system as an inherent part of LIBSYS.

the detail- that the accounting system should support discounts for regular LIBSYS users.

B). It is necessary to separate user requirements from more detailed system requirements in a requirements document. Otherwise, non-technical readers of the user requirements Get confused.

C). The user requirement should simply focus on the key facilities to be provided. As in Figure 6.10 to focus only on the essential system features.

Figure 6.10
A definition of an editor grid facility



2.6.1 Grid facilities

The editor shall provide a grid facility where a matrix of horizontal and vertical lines provide a background to the editor window. This grid shall be a passive grid where the alignment of entities is the user's responsibility.

Rationale: A grid helps the user to create a tidy diagram with well-spaced entities. Although an active grid, where entities 'snap-to' grid lines can be useful, the positioning is imprecise. The user is the best person to decide where entities should be positioned.

Specification: ECLIPSE/WS/Tools/DE/FS Section 5.6

Source: Ray Wilson, Glasgow Office

4. Guidelines to minimise misunderstandings when writing user requirements,

a). Invent a standard format and ensure that all requirement definitions adhere to that format.

b) Use language consistently. 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'.

c) Use text highlighting (bold, italic or colour) to pick out key parts of the requirement. Avoid, as far as possible, the use of computer jargon.

6.3 System requirements

1. System requirements are expanded versions of the user requirements that are used by software engineers as the starting point for the system design. Explain how the user requirements should be provided by the system.

2. 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. But it is not possible in practice.

- **a)** Intial architecture of the system have to be designed to structure the requirements specification.
- **b)** systems must interoperate with other existing systems. These constrain the design, and these constraints impose requirements on the new system.
- **c)** The use of a specific architecture to satisfy non-functional requirements may be necessary.

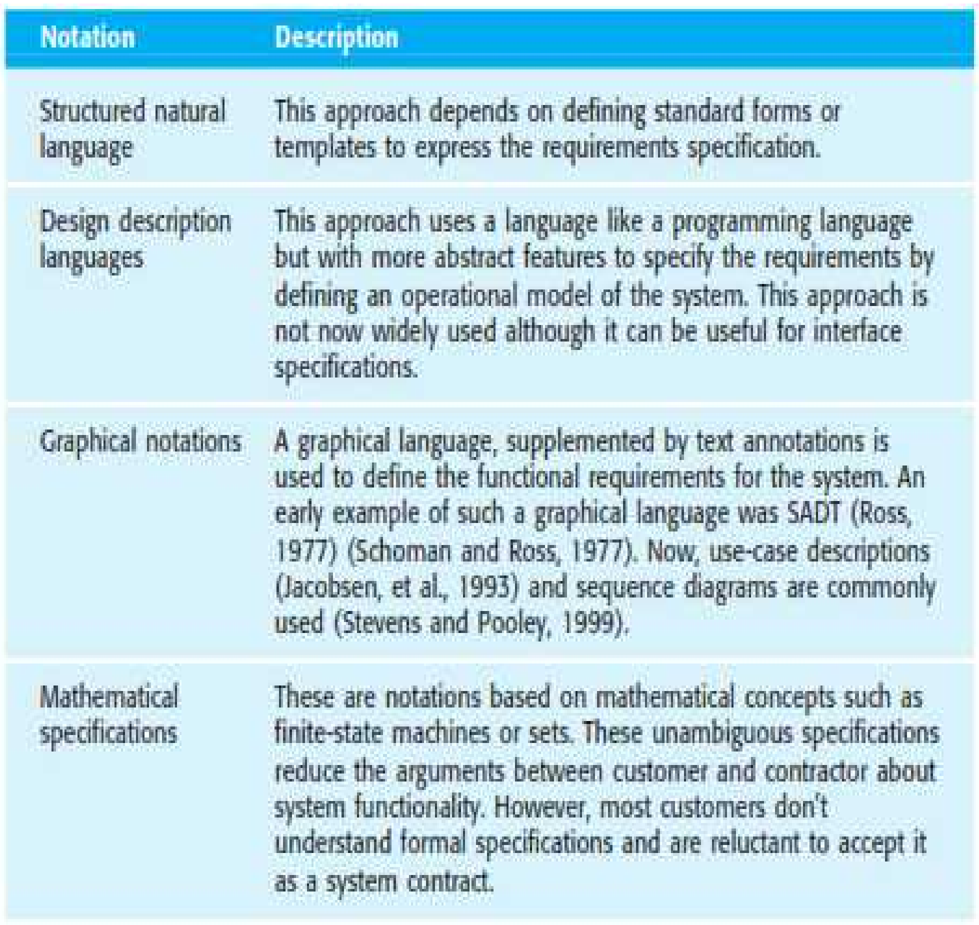
3. Natural language is often used to write system requirements specifications as well as user requirements. However , natural language specifications can be confusing and hard to understand:

- **a)** Natural language understanding relies on the specification readers and writers using the same words for the same concept. This leads to misunderstandings because of the ambiguity of natural language.
- **b)** A natural language requirements specification is overflexible. We can say the same thing in completely different ways. It is up to the reader to find out when requirements are the same and when they are distinct.
- **c)** There is no easy way to modularise natural language requirements. It may be difficult to find all related requirements.

It is essential to write user requirements in a language that non-specialists can understand.

Therefore consider system requirements in more specialised notations (Figure 6.11). These include stylised, structured natural language, graphical notations.

Figure 6.11 Notations for requirements specification



Notation	Description
Structured natural language	This approach depends on defining standard forms or templates to express the requirements specification.
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 not now widely used although it can be useful for interface specifications.
Graphical notations	A graphical language, supplemented by text annotations is used to define the functional requirements for the system. An early example of such a graphical language was SADT (Ross, 1977) (Schoman and Ross, 1977). Now, use-case descriptions (Jacobsen, et al., 1993) and sequence diagrams are commonly used (Stevens and Pooley, 1999).
Mathematical specifications	These are notations based on mathematical concepts such as finite-state machines or sets. These unambiguous specifications reduce the arguments between customer and contractor about system functionality. However, most customers don't understand formal specifications and are reluctant to accept it as a system contract.

6.3.1 Structured language specifications

1. 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.

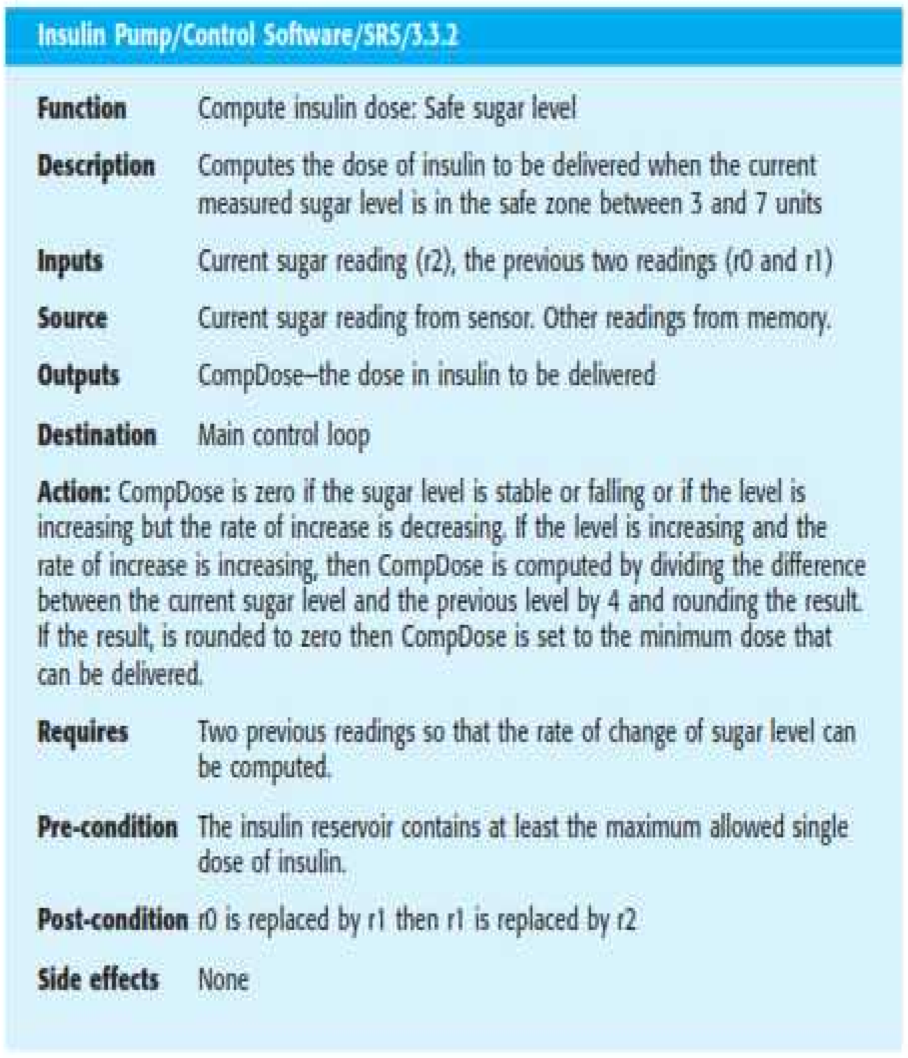
2. The advantage of this approach is that it maintains most of the expressiveness and understandability of natural language but ensures that some degree of uniformity is imposed on the specification.

Form Based specification

1. Special-purpose forms were designed to describe the input, output and functions of an software system. The system requirements were specified using these forms.
2. To use a form-based approach to specify system requirements, define one or more standard forms or templates to express the requirements.

An example of such a form-based specification is shown in Figure 6.12. The insulin pump bases its computations of the user's insulin requirement on the rate of change of blood sugar levels.

Figure 6.12 System requirements specification using a standard form



Insulin Pump/Control Software/SRS/3.3.2

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.

Requires	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

When a standard form is used for specifying functional requirements, the following information should be included:

1. Description of the function or entity being specified

2. Description of its inputs and where these come from

3. Description of its outputs and where these go to

4. Indication of what other entities are used (the *requires* part)

5. Description of the action to be taken

6. If a functional approach is used, a pre-condition setting out what must be true before the function is called and a post-condition specifying what is true after the function is called

7. Description of the side effects (if any) of the operation.

3. Advatages

1.Using formatted specifications removes some of the problems of natural language specification.

2.Variability in the specification is reduced and requirements are organised more effectively.

4. Disadvantages

1. It is difficult to write requirements in an unambiguous way, particularly when complex computations are required.
2. Things cannot be specified clearly.

Tabular specification

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

2. Graphical models are most useful when you need to show how state changes or where you need to describe a sequence of actions.
3. Figure 6.13 is a revised description of the computation of the insulin dose.
4. Figure 6.14 illustrates the sequence of actions when a user wishes to withdraw cash from an automated teller machine (ATM).

In Figure 6.14, there are three basic sub-sequences:

a) ***Validate card*** The user's card is validated by checking the card number and user's PIN.

b)Handle request The user's request is handled by the system. For a withdrawal, the database must be queried to check the user's balance and to debit the amount withdrawn. Notice the exception here if the requestor does not have enough money in their account.

c). ***Complete transaction*** The user's card is returned and, when it is removed, the cash and receipt are delivered.

Figure 6.13 Tabular specification of computation

Condition	Action
Sugar level falling ($r2 < r1$)	CompDose = 0
Sugar level stable ($r2 = r1$)	CompDose = 0
Sugar level increasing and rate of increase decreasing ($(r2 - r1) < (r1 - r0)$)	CompDose = 0
Sugar level increasing and rate of increase stable or increasing ($(r2 - r1) \geq (r1 - r0)$)	CompDose = round ($(r2 - r1)/4$) If rounded result = 0 then CompDose = MinimumDose

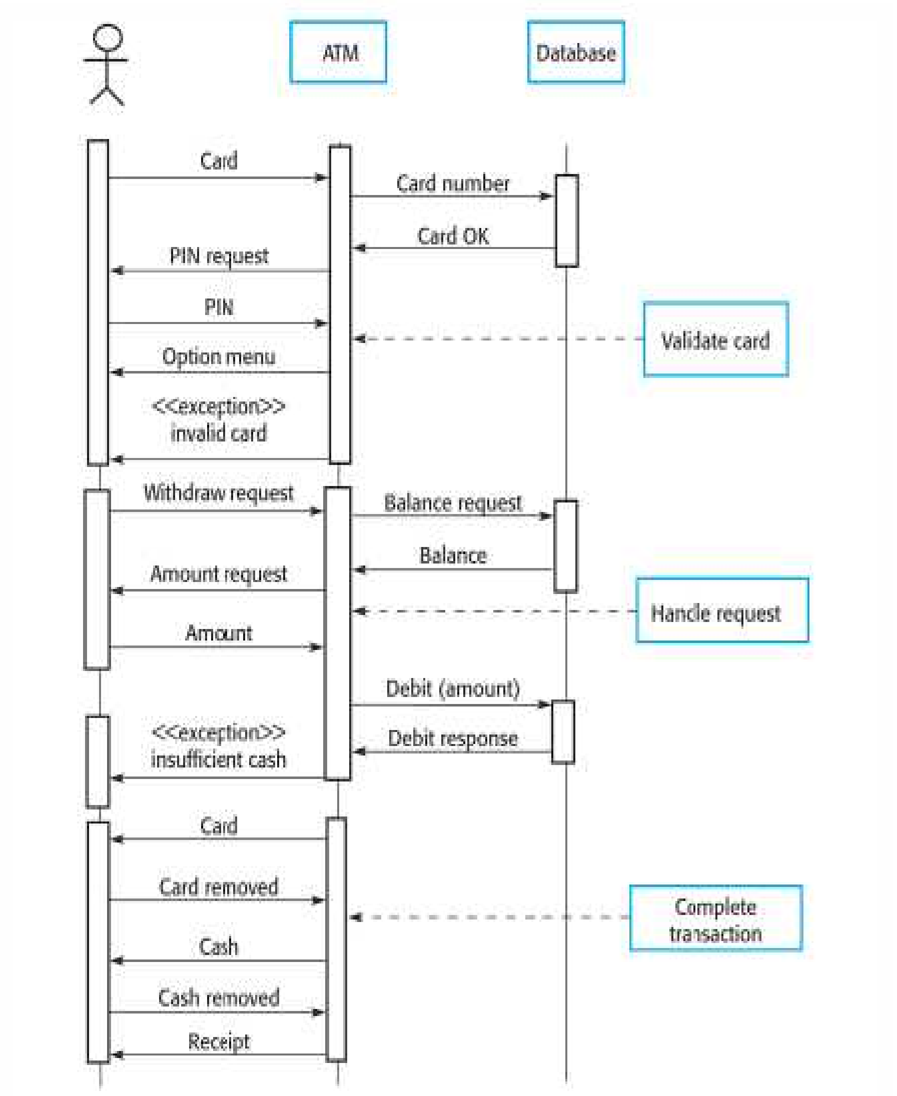


Figure 6.14 Sequence diagram of ATM withdrawal

6.4 Interface specification

Almost all software systems must operate with existing systems that have already been implemented and installed in an environment. If the new system and the existing systems must work together, the interfaces of existing systems have to be precisely specified.

There are three types of interface that may have to be defined:

1. ***Procedural interfaces*** where existing programs or sub-systems offer a range of services that are accessed by calling interface procedures. These interfaces are sometimes called Application Programming Interfaces (APIs).

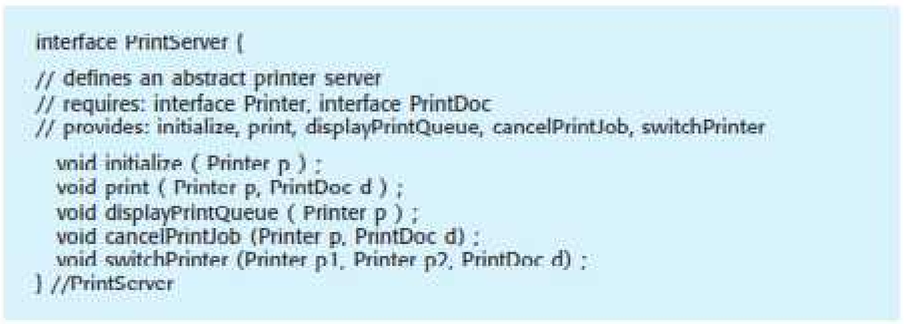
2. ***Data structures*** that are passed from one sub-system to another. Graphical data models are the best notations for this type of description.

Example: program descriptions in Java or C++ can be generated automatically from these descriptions.

3.***Representations of data*** (such as the ordering of bits) that have been established for an existing sub-system. These interfaces are most common in embedded, real-time system.

Figure 6.15 is an example of a procedural interface definition defined in Java. In this case, the interface is the procedural interface offered by a print server. This manages a queue of requests to print files on different printers. Users may examine the queue associated with a printer and may remove their print jobs from that queue. They may also switch jobs from one printer to another.

Figure 6.15 The Java PDL description of a print server interface



```
interface PrintServer {

// defines an abstract printer server
// requires: interface Printer, interface PrintDoc
// provides: initialize, print, displayPrintQueue, cancelPrintJob, switchPrinter

   void initialize ( Printer p ) ;
   void print ( Printer p, PrintDoc d ) ;
   void displayPrintQueue ( Printer p ) ;
   void cancelPrintJob (Printer p, PrintDoc d) ;
   void switchPrinter (Printer p1, Printer p2, PrintDoc d) ;
} //PrintServer
```

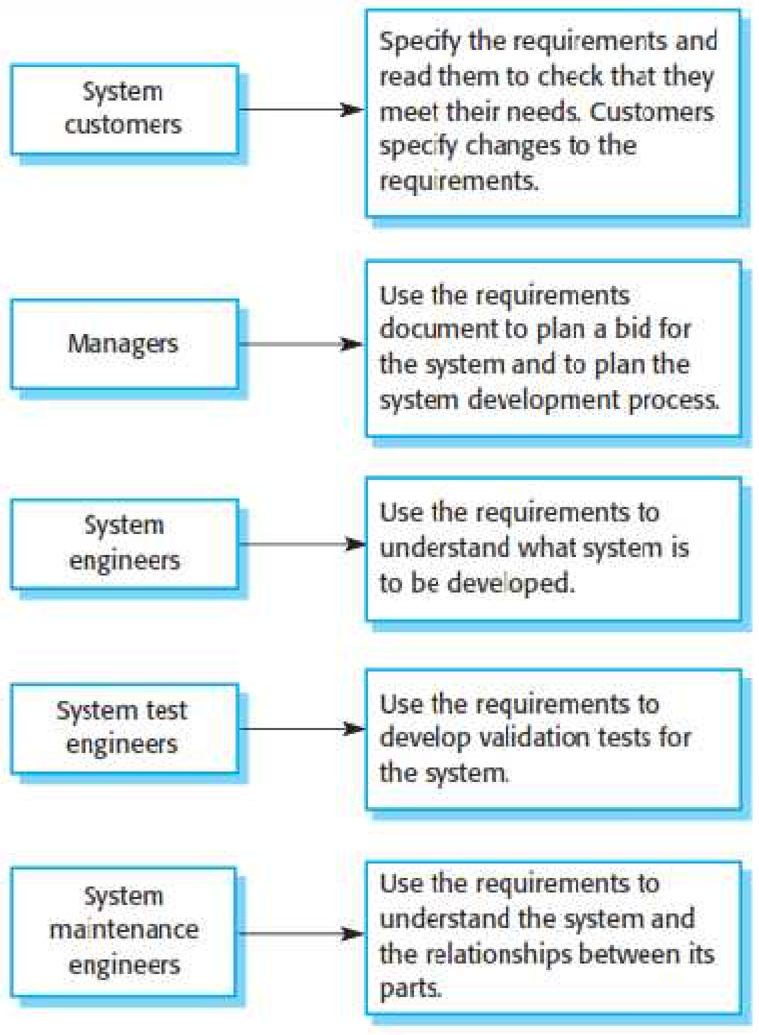
6.5 The software requirements document

1. The software requirements document (sometimes called the software requirements specification or SRS) is the 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.

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

Figure 6.16, illustrates possible users of the document and how they use it.

Figure 6.16 Users of a requirements document


System customers
Specify the requirements and read them to check that they meet their needs. Customers specify changes to the requirements.
Managers
Use the requirements document to plan a bid for the system and to plan the system development process.
System engineers
Use the requirements to understand what system is to be developed.
System test engineers
Use the requirements to develop validation tests for the system.
System maintenance engineers
Use the requirements to understand the system and the relationships between its parts.

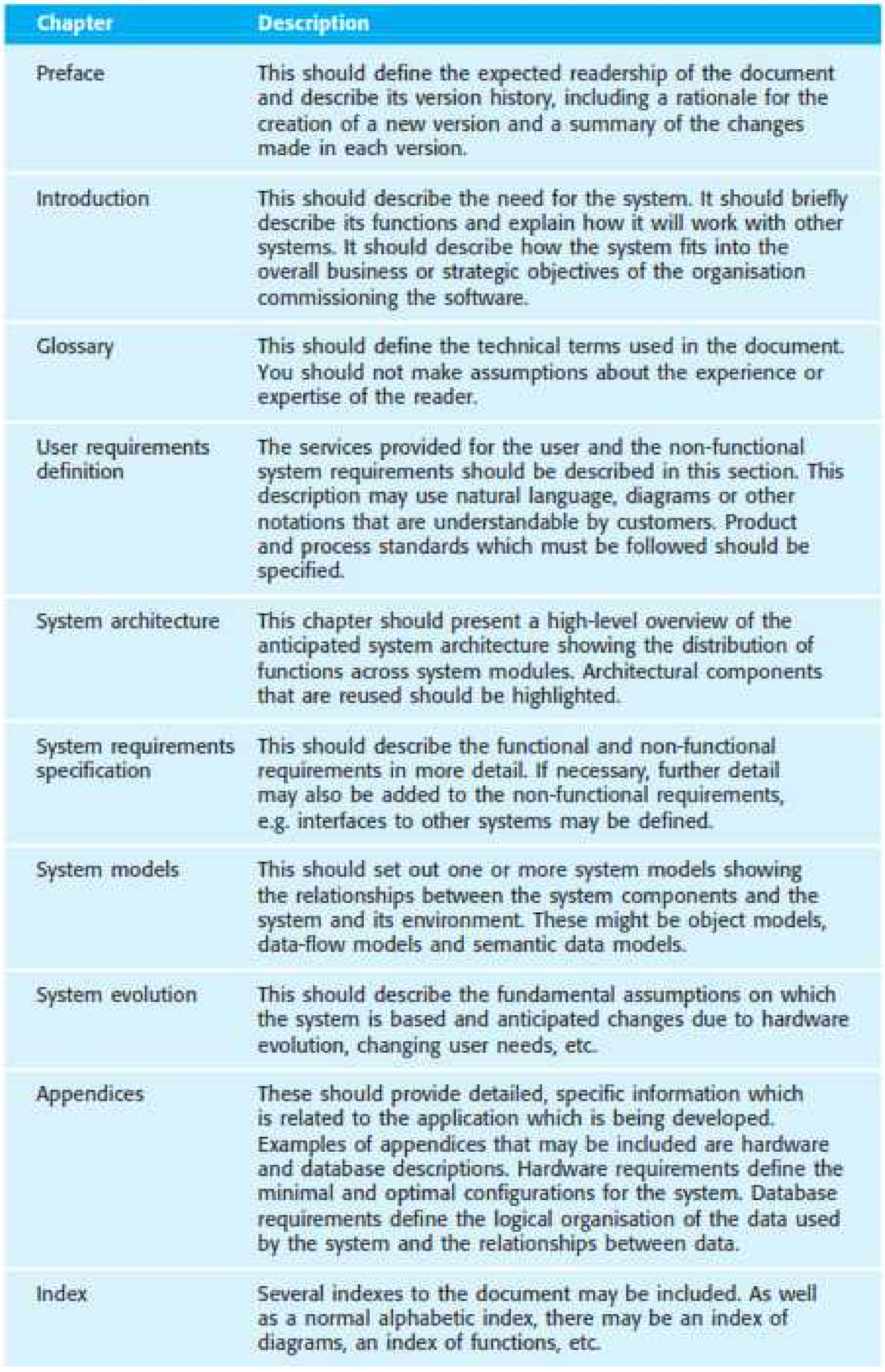
3. The level of detail that you should include in a requirements document depends on the type of system that is being developed and the development process used.

4. IEEE suggests standard for requirements documents:

1. **Introduction**
    - 1.1 Purpose of the requirements document
    - 1.2 Scope of the product
    - 1.3 Definitions, acronyms and abbreviations
    - 1.4 References
    - 1.5 Overview of the remainder of the document
2. **General description**
    - 2.1 Product perspective
    - 2.2 Product functions
    - 2.3 User characteristics
    - 2.4 General constraints
    - 2.5 Assumptions and dependencies
3. **Specific requirements** cover functional, non-functional and interface requirements. This is obviously the most substantial part of the document but because of the wide variability in organisational practice, it is not appropriate to define a standard structure for this section. The requirements may document external interfaces, describe system functionality and performance, specify logical database requirements, design constraints, emergent system properties and quality characteristics.
4. **Appendices**
5. **Index**

By contrast, when the software is part of a large system engineering project that includes interacting hardware and software systems, it is often essential to define the requirements to a fine level of detail. This means that the requirements documents are likely to be very long and should include most if not all of the chapters shown in Figure 6.17.

Figure 6.17 The structure of a requirements document



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 its functions and explain how it will work with other systems. It should describe how the system fits into the overall business or strategic objectives of the organisation 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	The services provided for the user and the non-functional system requirements should be described in this section. This description may use natural language, diagrams or other notations that are understandable by customers. Product and process standards which 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, e.g. interfaces to other systems may be defined.
System models	This should set out one or more system models showing the relationships between the system components and the system and its environment. These might be object models, data-flow models and semantic data models.
System evolution	This should describe the fundamental assumptions on which the system is based and anticipated changes due to hardware evolution, changing user needs, etc.
Appendices	These should provide detailed, specific information which is related to the application which is being developed. Examples of appendices that may be included are hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organisation 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, etc.

2nd chapter
7.Requirements engineering processes

.

Contents

1. The goal of the requirements engineering process is to create and maintain a system requirements document.

2. The overall process includes four high-level requirements engineering sub-processes.

- **a.** These are concerned with assessing whether the system is useful to the business (feasibility study);
- **b.** discovering requirements (elicitation and analysis);
- **c.** converting these requirements into some standard form (specification);
- **d.** checking that the requirements actually define the system that the customer wants (validation).

3. Figure 7.1 illustrates the relationship between these activities. It also shows the documents produced at each stage of the requirements engineering process

4. Figure 7.2 presents the process as a three-stage activity where the activities are organised as an iterative process around a spiral.

a) The amount of time and effort devoted to each activity in an iteration depends on the stage of the overall process and the type of system being developed.

b) Early in the process, most effort will be spent on understanding high-level business and non-functional requirements and the user requirements.

c) Later in the process, in the outer rings of the spiral, more effort will be devoted to system requirements engineering and system modelling.

Figure 7.1 The requirements engineering process

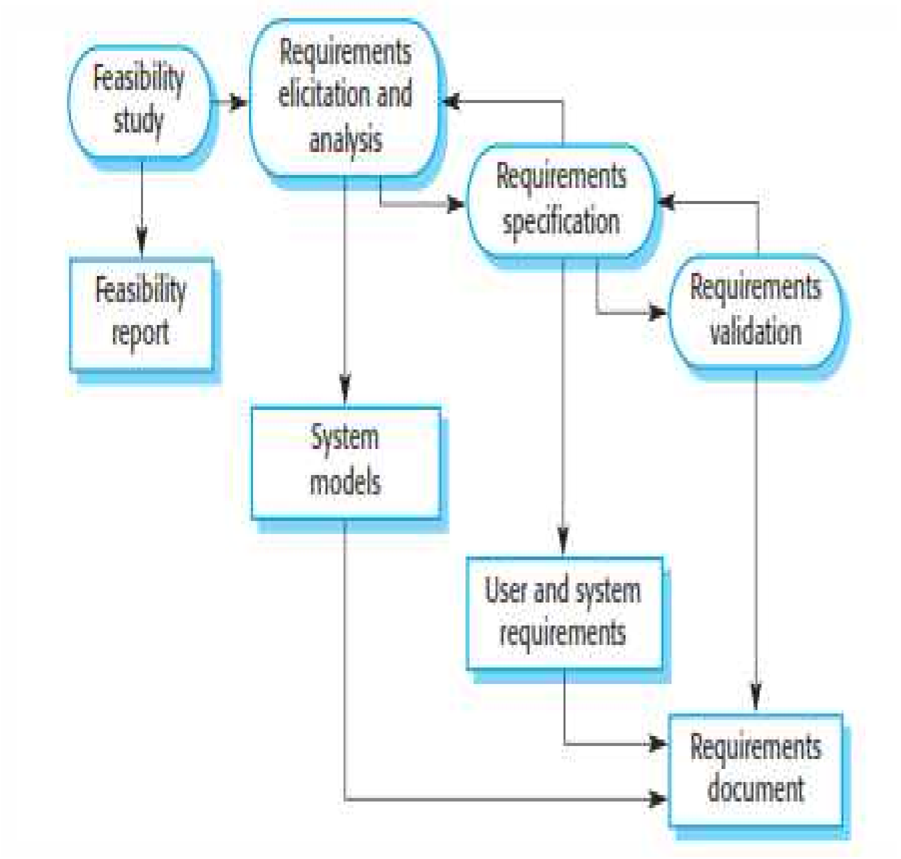
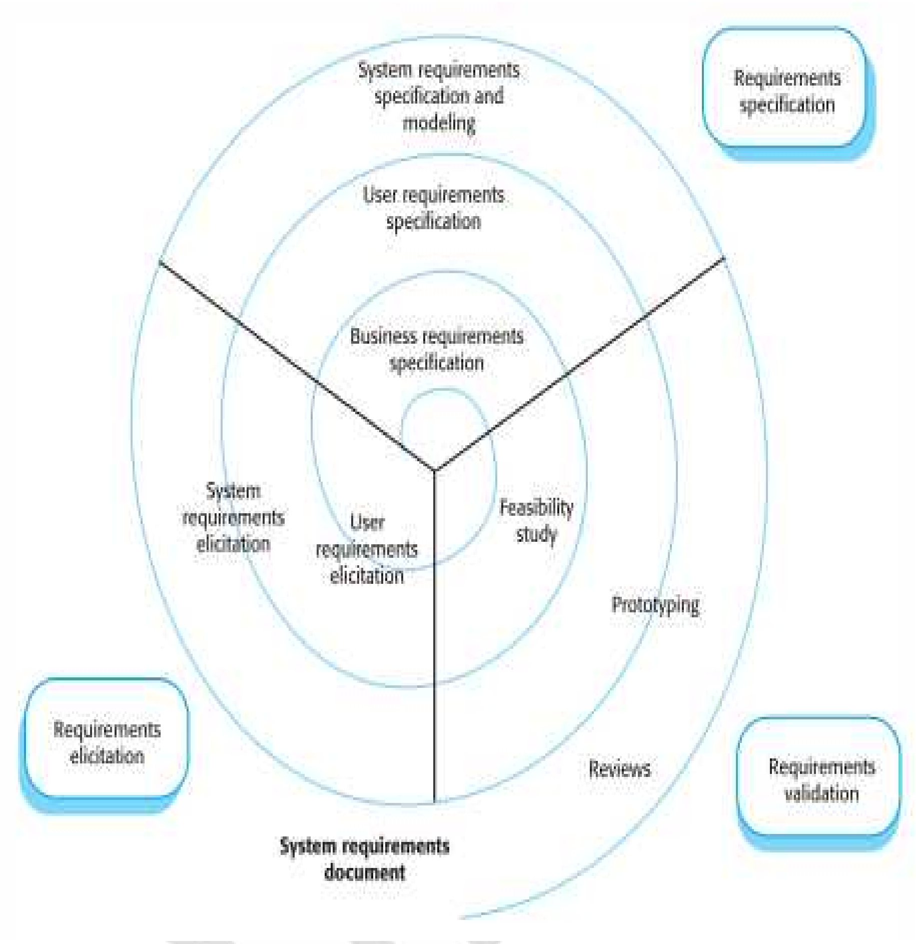


Figure 7.2 Spiral model of requirements engineering processes


System requirements specification and modeling
Requirements specification
User requirements specification
Business requirements specification
System requirements elicitation
User requirements elicitation
Feasibility study
Prototyping
Requirements elicitation
Reviews
Requirements validation
System requirements document

7.1 Feasibility studies

For all new systems, the requirements engineering process should start with a feasibility study.

1. The input to the feasibility study is a set of preliminary business requirements, an outline description of the system and how the system is intended to support business processes.
2. The results of the feasibility study should be a report that recommends whether or not it is worth carrying on with the requirements engineering and system development process.

3. A feasibility study is a short, focused study that aims to answer a number of questions:

a. Does the system contribute to the overall objectives of the organisation?

b. Can the system be implemented using current technology and within given cost and schedule constraints?

c. Can the system be integrated with other systems which are already in place?

4. Carrying out a feasibility study involves information assessment, information collection and report writing.

-The information assessment phase identifies the information that is required to answer the three questions set out above.

-Once the information has been identified, talk with information sources to discover the answers to these questions

Some examples of possible questions that may be put are:

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

5. Once the information is ready, write the feasibility study report. In feasibility report make a recommendation about whether or not the system development should continue,may propose changes to the scope, budget and schedule of the system and suggest further high-level requirements for the system.

7.2 Requirements elicitation and analysis

1. 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 people in an organisation.

2. The term *stakeholder* is used to refer to any person or group who will be affected by the system, directly or indirectly.

3. Stakeholders include end-users who interact with the system and everyone else in an organisation that may be affected by its installation.

Eliciting and understanding stakeholder requirements is difficult for several reasons:

a). Stakeholders often don't know what they want from the computer system except in the most general terms. They may find it difficult to articulate what they want the system to do or make unrealistic demands because they are unaware of the cost of their requests.

b). Stakeholders naturally express requirements in their own terms and with implicit knowledge of their own work. Requirements engineers, without experience in the customer's domain, must understand these requirements.

c) Different stakeholders have different requirements, which they may express in different ways. Requirements engineers have to consider all potential sources of requirements and discover commonalities and conflict.

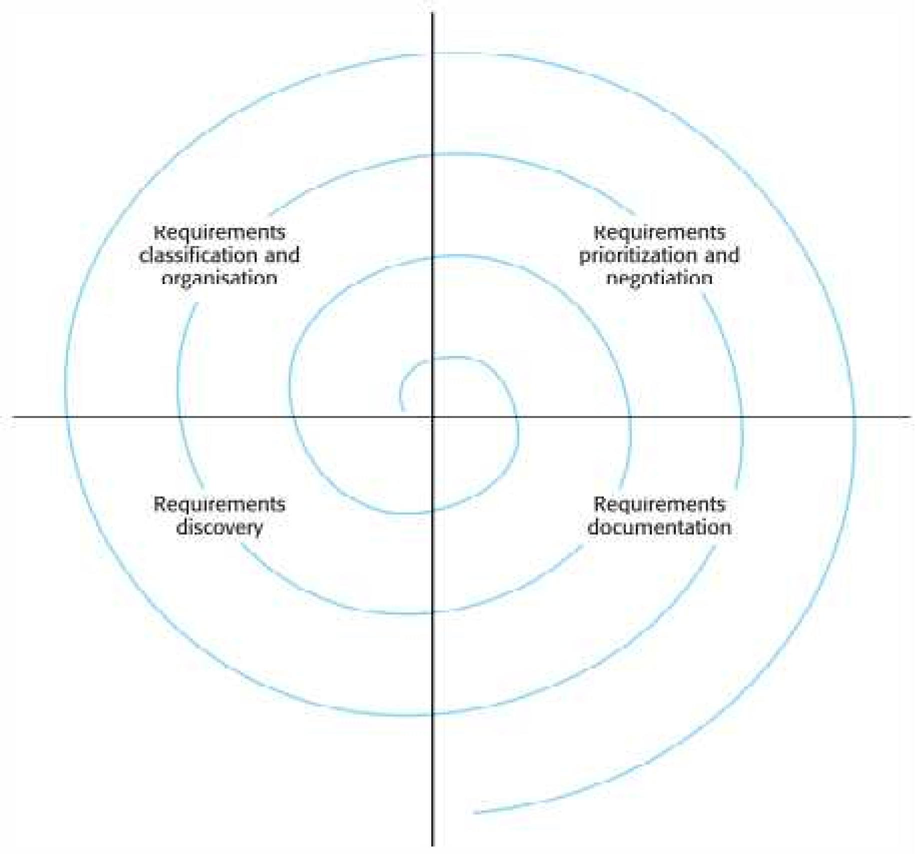
d). Political factors may influence the requirements of the system. For example, managers may demand specific system requirements that will increase their influence in the organisation.

e). The economic and business environment in which the analysis takes place is dynamic. It inevitably changes during the analysis process. Hence the importance of particular requirements may change. New requirements may emerge

From new stakeholders who were not originally consulted.

4. A very general process model of the elicitation and analysis process is shown in Figure 7.3.

Figure 7.3 The requirements elicitation and analysis process


Requirements classification and organisation
Requirements prioritization and negotiation
Requirements discovery
Requirements documentation

5. The process activities are:

A). *Requirements discovery* This is the process of interacting with stakeholders in the system to collect their requirements. Domain requirements from stakeholders and documentation are also discovered during this activity.

B) *Requirements classification and organisation* This activity takes the unstructured collection of requirements, groups related requirements and organises them into coherent clusters.

C) *Requirements prioritisation and negotiation* Inevitably, where multiple stakeholders are involved, requirements will conflict. This activity is concerned with prioritising requirements, and finding and resolving requirements conflicts through negotiation.

D). *Requirements documentation* The requirements are documented and input into the next round of the spiral. Formal or informal requirements documents may be produced.

Figure 7.3 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 requirements documentation. The analyst's understanding of the requirements improves with each round of the cycle.

7.2.1 Requirements discovery

1. Requirements discovery is the process of gathering information about the proposed and existing systems and distilling the user and system requirements from this information.

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

3. For example,

system stakeholders for a bank ATM include:

1. *Current bank customers*
2. *Representatives from other banks* who have reciprocal agreements that allow each other's ATMs to be used
3. *Managers of bank branches* who obtain management information from the system
4. *Counter staff at bank branches* who are involved in the day-to-day running of the system
5. *Database administrators* who are responsible for integrating the system with the bank's customer database
6. *Bank security managers* who must ensure that the system will not pose a security hazard
7. *The bank's marketing department* who are likely be interested in using the sys-tem as a means of marketing the bank
8. *Hardware and software maintenance engineers* who are responsible for maintaining and upgrading the hardware and software
9. *National banking regulators* who are responsible for ensuring that the system conforms to banking regulations

Viewpoints

1. Viewpoint-oriented approaches to requirements engineering organise both the elicitation process and the requirements using viewpoints.
2. A key strength of viewpoint-oriented analysis is that it recognises multiple perspectives and provides a framework for discovering conflicts in the requirements proposed by different stakeholders.

3. Viewpoints can be used as a way of classifying stakeholders and other sources of requirements. There are three generic types of viewpoint:

a) ***Interactor viewpoints*** represent people or other systems that interact directly with the system.
In the bank ATM system, examples of interactor viewpoints are the bank's customers and the bank's account database.

b) ***Indirect viewpoints*** represent stakeholders who do not use the system themselves but who influence the requirements in some way.
In the bank ATM system, examples of indirect viewpoints are the management of the bank and the bank security staff.

c) ***Domain viewpoints*** represent domain characteristics and constraints that influence the system requirements.
In the bank ATM system, an example of a domain viewpoint would be the standards that have been developed for interbank communications.

4. The initial identification of viewpoints that are relevant to a system can sometimes be difficult. To help with this process, you should try to identify more specific viewpoint types:

a) Providers of services to the system and receivers of system services
b) Systems that should interface directly with the system being specified
c) Regulations and standards that apply to the system
d) The sources of system business and non-functional requirements
e) Engineering viewpoints reflecting the requirements of people who have to develop manage and maintain the system

f) Marketing and other viewpoints that generate requirements on the product features expected by customers and how the system should reflect the external image of the organization

5. Advantages: Engineering viewpoints important for two reasons

Firstly, the engineers developing the system may have experience with similar systems and may be able to suggest requirements from that experience.

Secondly, technical staff who have to manage and maintain the system may have requirements that will help simplify system support.

6. As an illustration, consider the viewpoint hierarchy shown in Figure 7.4. This is the diagram of the viewpoints for the LIBSYS system.

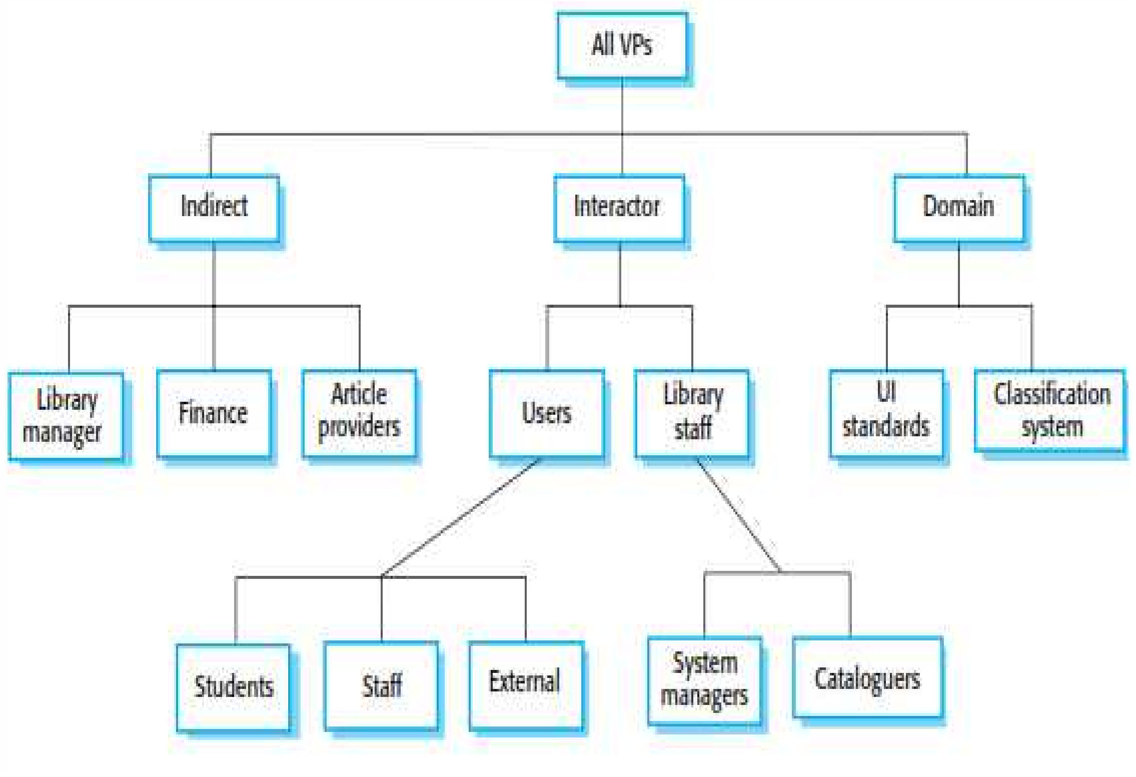


Figure 7.4
Viewpoints in LIBSYS

Interviewing

1. In these interviews, the requirements engineering team puts questions to stakeholders about the system that they use and the system to be developed. Requirements are derived from the answers to these questions.

2. Interviews may be of two types:

a). Closed interviews where the stakeholder answers a predefined set of questions.

b). Open interviews where there is no predefined agenda. The requirements engineering team explores a range of issues with system stakeholders and hence develops a better understanding of their needs.

3. **Advantages**: Interviews are good for getting an overall understanding of what stakeholders do, how they might interact with the system and the difficulties that they face with current systems.

4. Dis-advantages :

a). However, interviews are not so good for understanding the requirements from the application domain.

It is hard to elicit domain knowledge during interviews for two reasons:

A. All application specialists use terminology and jargon that is specific to a domain. It is impossible for them to discuss domain requirements without using this terminology.
B. Some domain knowledge is so familiar to stakeholders that they either find it difficult to explain or they think it is so fundamental that it isn't worth mentioning.

For example, for a librarian, it goes without saying that all acquisitions
Are catalogued before they are added to the library. However, this may not be

Obvious to the interviewer so it isn't taken into account in the requirements.

b) Interviews are not an effective technique for eliciting knowledge about organizational requirements and constraints because there are subtle power and influence relationships between the stakeholders in the organisation.

5. Effective interviewers have two characteristics:

- **a.** They are open-minded, avoid preconceived ideas about the requirements and are willing to listen to stakeholders. If the stakeholder comes up with surprising requirements, they are willing to change their mind about the system.
- **b.** They prompt the interviewee to start discussions with a question, a requirements proposal or by suggesting working together on a prototype system.

Scenarios

1. Scenarios can be particularly useful for adding detail to an outline requirements description. They are descriptions of example interaction sessions.

2. Each scenario covers one or more possible interactions.

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

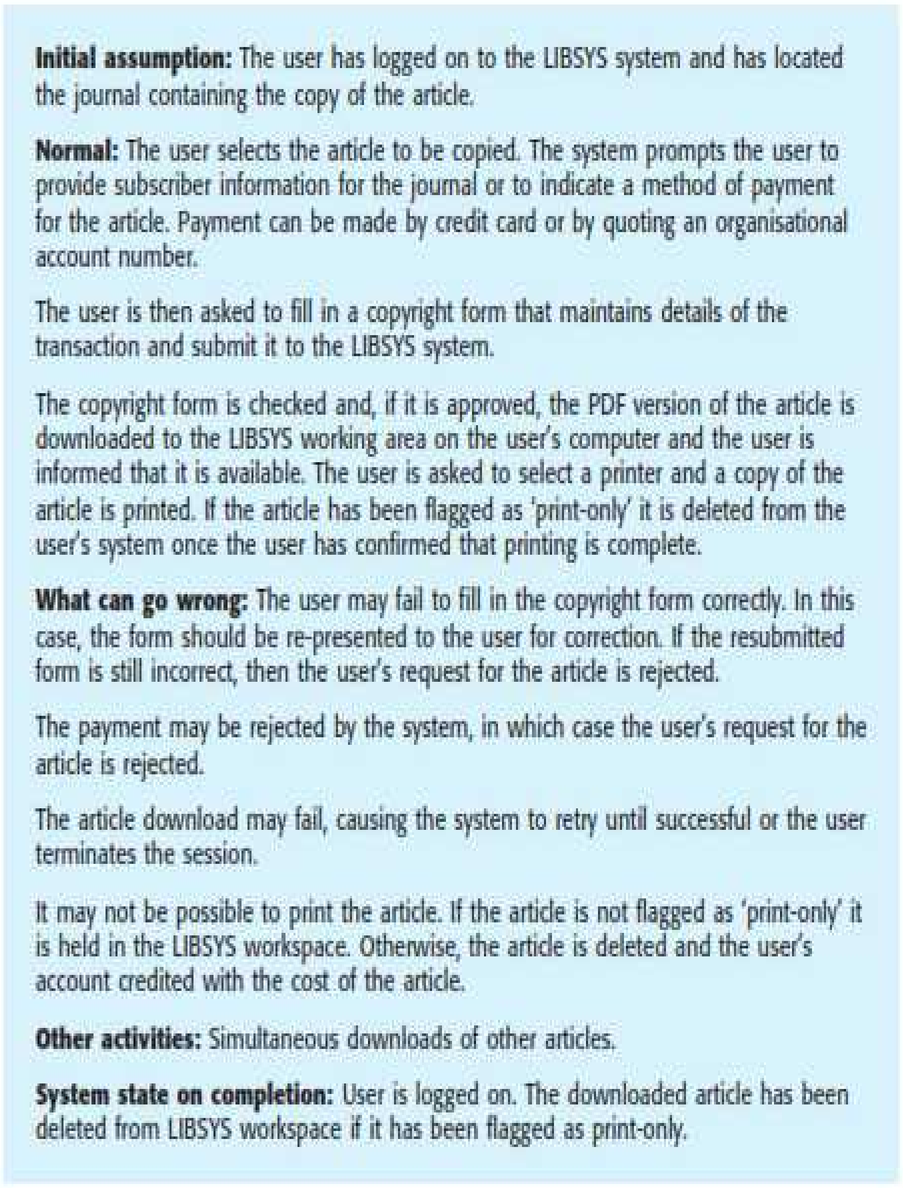
- **a.** A description of what the system and users expect when the scenario starts
- **b)** A description of the normal flow of events in the scenario
- **c)** A description of what can go wrong and how this is handled
- **d)** Information about other activities that might be going on at the same time
- **e)** A description of the system state when the scenario finishes.

4. Scenario-based elicitation can be carried out informally, where the requirements engineer works with stakeholders to identify scenarios and to capture details of these scenarios.

5.Scenarios may be written as text, supplemented by diagrams, screen shots, and so on.

6. As an example of a simple text scenario, consider how a user of the LIBSYS library system may use the system. This scenario is shown in Figure 7.5.

Figure 7.5 Scenario for article downloading in LIBSYS



Initial assumption: The user has logged on to the LIBSYS system and has located the journal containing the copy of the article.

Normal: The user selects the article to be copied. The system prompts the user to provide subscriber information for the journal or to indicate a method of payment for the article. Payment can be made by credit card or by quoting an organisational account number.

The user is then asked to fill in a copyright form that maintains details of the transaction and submit it to the LIBSYS system.

The copyright form is checked and, if it is approved, the PDF version of the article is downloaded to the LIBSYS working area on the user's computer and the user is informed that it is available. The user is asked to select a printer and a copy of the article is printed. If the article has been flagged as 'print-only' it is deleted from the user's system once the user has confirmed that printing is complete.

What can go wrong: The user may fail to fill in the copyright form correctly. In this case, the form should be re-presented to the user for correction. If the resubmitted form is still incorrect, then the user's request for the article is rejected.

The payment may be rejected by the system, in which case the user's request for the article is rejected.

The article download may fail, causing the system to retry until successful or the user terminates the session.

It may not be possible to print the article. If the article is not flagged as 'print-only' it is held in the LIBSYS workspace. Otherwise, the article is deleted and the user's account credited with the cost of the article.

Other activities: Simultaneous downloads of other articles.

System state on completion: User is logged on. The downloaded article has been deleted from LIBSYS workspace if it has been flagged as print-only.

Use-cases

1. Use-cases are a scenario-based technique for requirements elicitation.

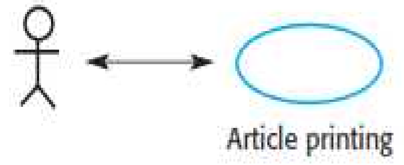
2. In their simplest form, a use-case identifies the type of interaction and the actors involved .

3. Figure 7.6 illustrates the essentials of the use-case notation.

Actors in the process are represented as stick figures,

Each class of interaction is represented as a named ellipse.

Figure 7.6 A simple use-case for article printing



4. Figure 7.7 develops the LIBSYS example and shows other use-cases in that environment.

5. use-case encapsulates a set of scenarios, and each scenario is a single thread through the use-case.

6. Use-cases identify the individual interactions with the system. They can be documented with text or linked to UML models that develop the scenario in more detail.

7. Sequence diagrams are often used to add information to a use-case. These sequence diagrams show the actors involved in the interaction, the objects they interact with and the operations associated with these objects.

8. Figure 7.8 shows the interactions involved in using LIBSYS for downloading and printing an article. In Figure 7.8, there are four objects of **classes—Article, Form, Workspace and Printer**—involved in this interaction.

The labels on the arrows between the actors and objects indicate the names of operations.

Essentially, a user request for an article triggers a request for a copyright form. Once the user has completed the form, the article is downloaded and sent to the printer. Once printing is complete, the article is deleted from the LIBSYS workspace.

9. Advantages:

a).Scenarios and use-cases are effective techniques for eliciting requirements for interactor viewpoints, where each type of interaction can be represented as a usecase.

b).They can also be used in conjunction with some indirect viewpoints where These viewpoints receive some results (such as a management report) from the system.

10. Disadvantages:

a). Because they focus on interactions, they are not as effective for eliciting constraints or high-level business and non-functional requirements from indirect viewpoints or for discovering domain requirements.

Figure 7.7 Use cases for the library system

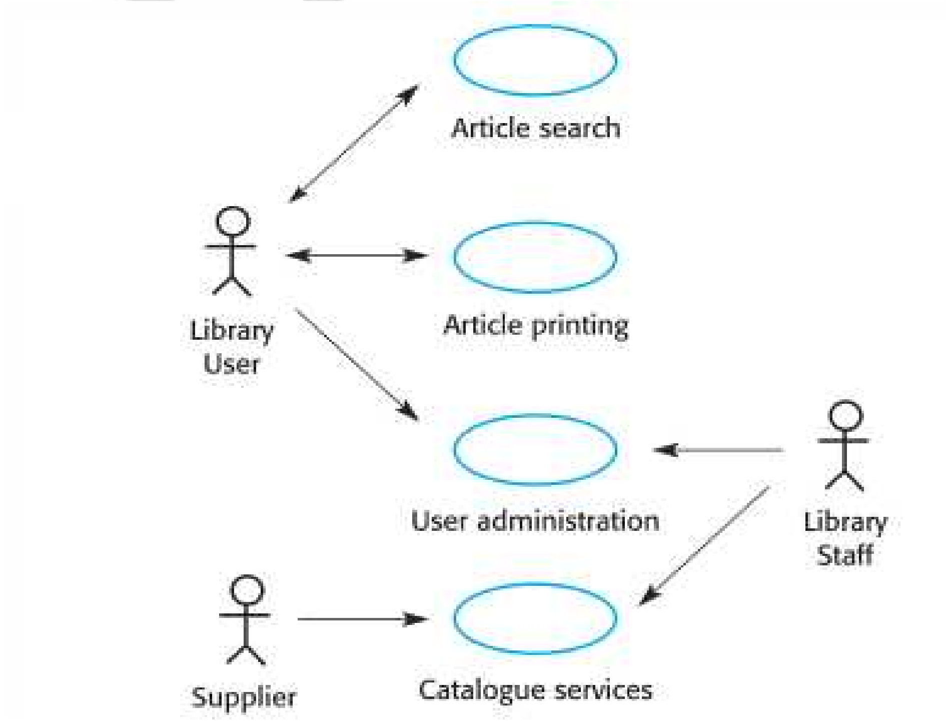
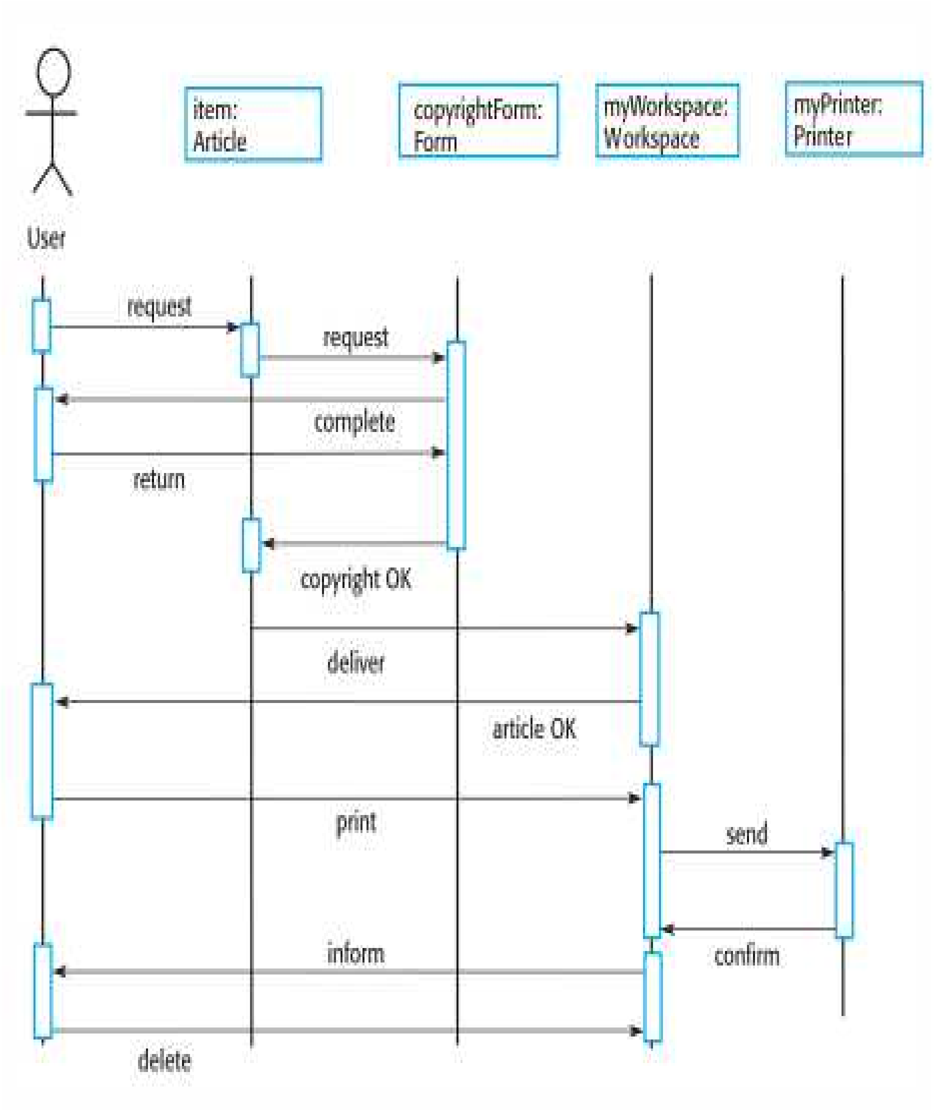


Figure 7.8 System interactions for article printing



7.2.2 Ethnography

1. Software systems do not exist in isolation—they are used in a social and organizational context. Satisfying these social and organisational requirements is often critical for the success of the system.

2. *Ethnography* is an observational technique that can be used to understand social and organisational requirements.

3. Ethnography is particularly effective at discovering two types of requirements:

a). *Requirements that are derived from the way in which people actually work* rather than the way in which process definitions say they ought to work.

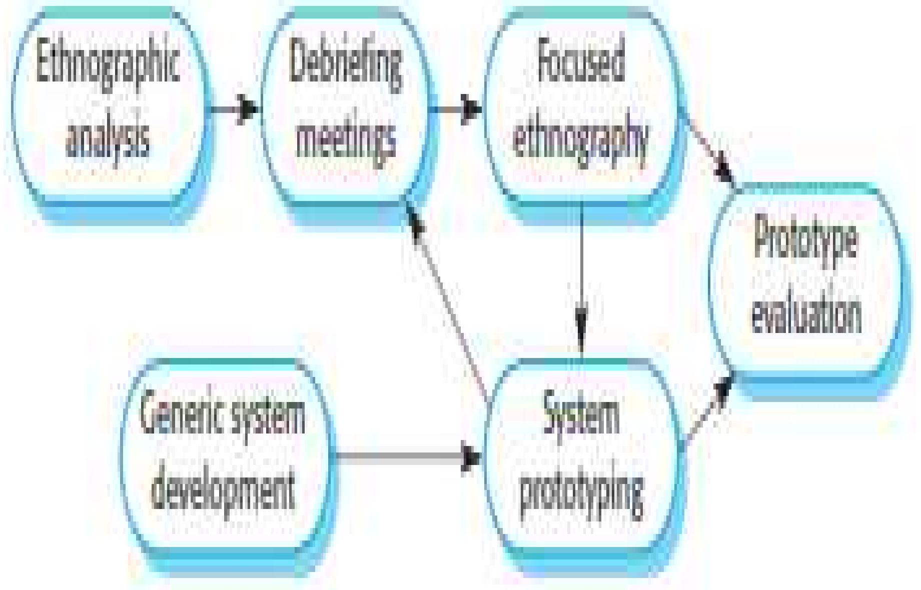
For example, air traffic controllers may switch off an aircraft conflict alert system that detects aircraft with intersecting flight paths even though normal control procedures specify that it should be used. Because air traffic controllers distracts from their work.

b) *Requirements that are derived from cooperation and awareness of other people's activities*.

For example, air traffic controllers may use an awareness of other controllers' work to predict the number of aircraft that will be entering their control sector.

4. Ethnography may be combined with prototyping (Figure 7.9). The prototyping focuses the ethnography by identifying problems and questions that can then be discussed with the ethnographer.

Figure 7.9 Ethnography and prototyping for requirements analysis



5. Advantages:

1. Ethnographic studies can reveal critical process details that are often missed by other requirements elicitation techniques.

6. Disadvantages:

1. Because of its focus on the enduser, this approach is not appropriate for discovering organisational or domain requirements.

2.Ethnographic studies cannot always identify new features that should be added to a system.

3. Ethnography is not, a complete approach to elicitation on its own, and it should be used to complement other approaches, such as use-case analysis.

7.3 Requirements validation

1. Requirements validation is concerned with showing that the requirements actually define the system that the customer wants.

2. During the requirements validation process, checks should be carried out on the requirements in the requirements document.

These checks include:

a). *Validity checks:* checks whether the system meets user needs?

b).*Consistency checks* Requirements in the document should not conflict. there should be no contradictory constraints or descriptions of the same system function.

c).*Completeness checks* The requirements document should include requirements, which define all functions, and constraints intended by the system user.

d).*Realism checks* using knowledge of existing technology, the requirements should be checked to ensure that they could actually be implemented.

e). *Verifiability* system requirements should always be written so that they are verifiable.

3. A number of requirements validation techniques can be used in conjunction or individually:

a). *Requirements reviews* The requirements are analysed systematically by a team of reviewers.

b). *Prototyping* an executable model of the system is demonstrated to end users and customers.

c). *Test-case generation* Requirements should be testable, by using test cases.

7.3.1 Requirements reviews

1. A requirements review is a manual process that involves people from both client and contractor organisations. They check the requirements document for anomalies and omissions.

2. Requirements reviews can be informal or formal.

Informal reviews -involve contractors discussing requirements with as many system stakeholders as possible. Many problems can be detected about the system to stakeholders before making a commitment to a formal review.

formal requirements review- the development team should 'walk' the client through the system requirements, explaining the implications of each requirement.

3. The review team should check each requirement for consistency as well as check the requirements as a whole for completeness. Reviewers may also check for:

a) *Verifiability* Is the requirement as stated realistically testable?

b) ***Comprehensibility*** Do the procurers or end-users of the system properly understand the requirement?

c) ***Traceability*** Is the origin of the requirement clearly stated?

d) ***Adaptability*** Is the requirement adaptable? That is, can the requirement be changed without large-scale effects on other system requirements?

7.4 Requirements management

1. The requirements for large software systems are always changing. Requirement management is the process of managing changing requirements during the requirement engineering process.

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

a. Large systems usually have a diverse user community where users have different requirements and priorities. These may be conflicting or contradictory.

b) System customers impose requirements because of organisational and budgetary constraints. These may conflict with end-user requirements and, after delivery, new features may have to be added for user support if the system is to meet its goals.

c) The business and technical environment of the system changes after installation, and these changes must be reflected in the system. New hardware may be introduced, it may be necessary to interface the system with other systems etc

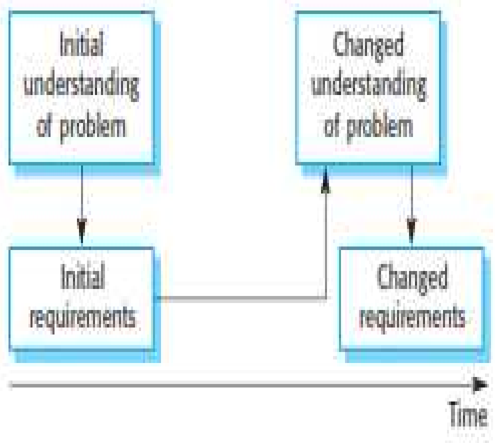
3. Requirements management is the process of understanding and controlling changes to system requirements.

7.4.1 Enduring and volatile requirements

1. Requirements evolution during the RE process and after a system has gone into service is inevitable.

2. As the requirements definition is developed, leads to a better understanding of users' needs. This feeds information back to the user, who may then propose a change to the requirements (Figure 7.10).

Figure 7.10 Requirements evolution


Initial understanding of problem
Changed understanding of problem
Initial requirements
Changed requirements
Time

From an evolution perspective, requirements fall into two classes:

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

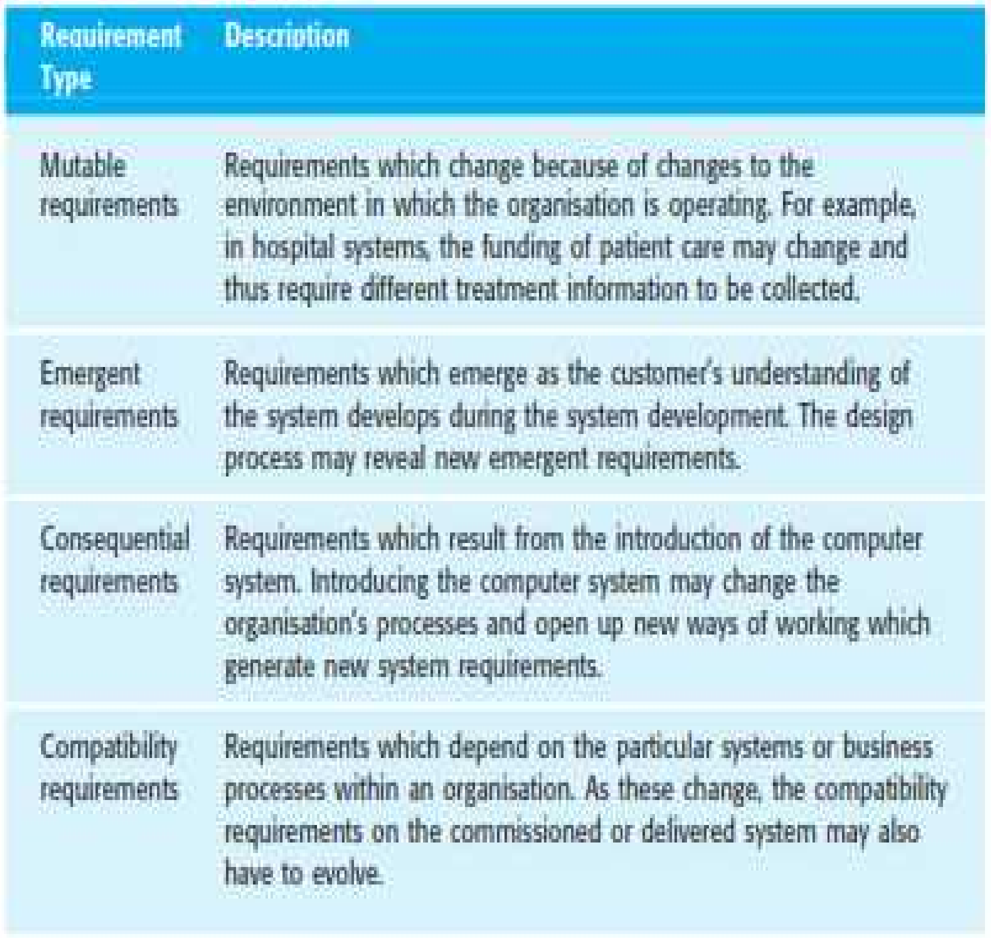
example, in a hospital, there will always be requirements concerned with patients, doctors, nurses and treatments.

b). *Volatile requirements* These are requirements that are likely to change during the system development process or after the system has been become operational.

example would be requirements resulting from government healthcare policies.

volatile requirements fall into five classes. As shown in Figure 7.11.

Figure 7.11 Classification of volatile requirements



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

7.4.2 Requirements management planning

1. Planning is an essential first stage in the requirements management process. Requirements management is very expensive. For each project, the planning stage establishes the level of requirements management detail that is required. That decide on:

a. ***Requirements identification*** Each requirement must be uniquely identified so that it can be cross-referenced by other requirements and so that it may be used in traceability assessments.

b. ***A change management process*** This is the set of activities that assess the impact and cost of changes. section.

c. *Traceability policies* These policies define the relationships between requirements, and between the requirements and the system design that should be recorded and how these records should be maintained.

d. *CASE tool support* provides automated support for system development. Tools that may be used range from specialist requirements management systems to spreadsheets and simple database systems.

Traceability

Traceability is the property of a requirements specification that reflects the ease of finding related requirements.

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

a) ***Source traceability*** information links the requirements to the stakeholders who proposed the requirements and to the rationale for these requirements. When a change is proposed, this information is used to find and consult the stakeholders about the change.

b). ***Requirements traceability*** information links dependent requirements within the requirements document. This information is used to assess how many requirements are likely to be affected by a proposed change and the extent of consequential requirements changes that may be necessary.

c). ***Design traceability*** information links the requirements to the design modules where these requirements are implemented. This information is used to assess the impact of proposed requirements changes on the system design and implementation.

Traceability Matrices

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

2. In a requirements traceability matrix, each requirement is entered in a row and in a column in the matrix. Where dependencies between different requirements exist, these are recorded in the cell at the row/column intersection.

3. Figure 7.12 shows a simple traceability matrix that records the dependencies between requirements.

-A 'D' in the row/column intersection illustrates that the requirement in the row depends on the requirement named in the column;

-an 'R' means that there is some other, weaker relationship between the requirements.

example:they may both define the requirements for parts of the same subsystem.

4.Advantages:

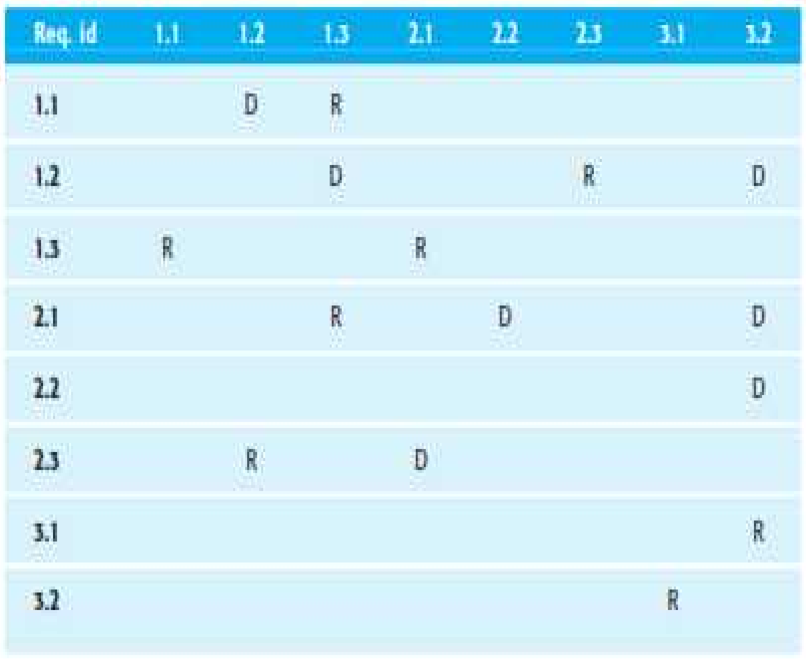
-Traceability matrices may be used when a small number of requirements have to be managed

- Traceability matrices can be generated automatically from the database.

5.Disavantages:

- Unwieldy and expensive to maintain for large systems with many requirements.

Figure 7.12
A traceability matrix



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

CASE Tools

Requirements management needs automated support; the CASE tools for this should be chosen during the planning phase, 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 7.13) is simplified if active tool support is available.

3. *Traceability management* tool support for traceability allows related requirements to be discovered. Some tools use natural language processing techniques to discover possible relationships between the requirements.

For small systems, it may not be necessary to use specialised requirements management tools. However, for larger systems, more specialised tool support is required.

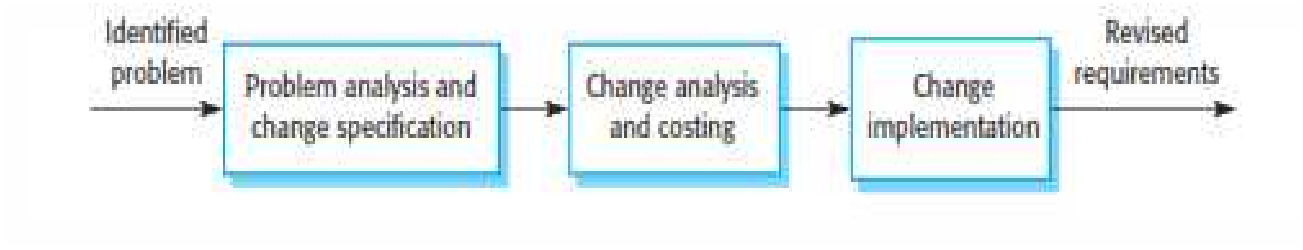

Identified problem
Problem analysis and change specification
Change analysis and costing
Change implementation
Revised requirements

Figure 7.13
Requirements change management

7.4.3 Requirements change management

1. Requirements change management (Figure 7.13) should be applied to all proposed changes to the requirements.

2. The advantage of using a formal process for change management is that all change proposals are treated consistently and that changes to the requirements document are made in a controlled way.

There are three principal stages to a change management process:

a) ***Problem analysis and change specification*** The process starts with an identified requirements problem or, sometimes, with a specific change proposal.

-During this stage, the problem or the change proposal is analysed to check that it is valid.

-The results of the analysis are fed back to the change requestor, and sometimes a more specific requirements change proposal is then made.

b) ***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 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 to proceed with the requirements change.

c)***Change implementation*** The requirements document and, where necessary, the system design and implementation are modified.

-Then organise the requirements document so that changes can be made to it without extensive rewriting or reorganisation.

-As with programs, changeability in documents is achieved by minimising 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.