Module 2

Software Requirements

Objectives:

- To introduce the concepts of user and system requirements
- To describe functional and non-functional requirements
- To explain two techniques for describing system requirements
- To explain how software requirements may be organised in a requirements document

User requirements and system requirements may be defined as follows:

- **User requirements**
  - ➢ Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.
- **System requirements**
  - ➢ A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

Definitions and specifications

- **Requirements definition:**

The software must provide a means of representing and accessing external files created by other tool

Requirements specification

- The user should be provided with facilities to define the type of external files.
- Each external file type may have an associated tool which may be applied to the file.
- Each external file type may be represented as a specific icon on the user's display.
- Facilities should be provided for the icon representing an external file type to be defined by the user.
- When a user selects an icon representing an external file, the effect of that election is to apply the tool associated with the type of the external file to the file represented by the selected icon.
- You need to write requirements at different levels of detail because different types of readers use them in different ways.
- Figure shows the types of readers for the user and system requirements:

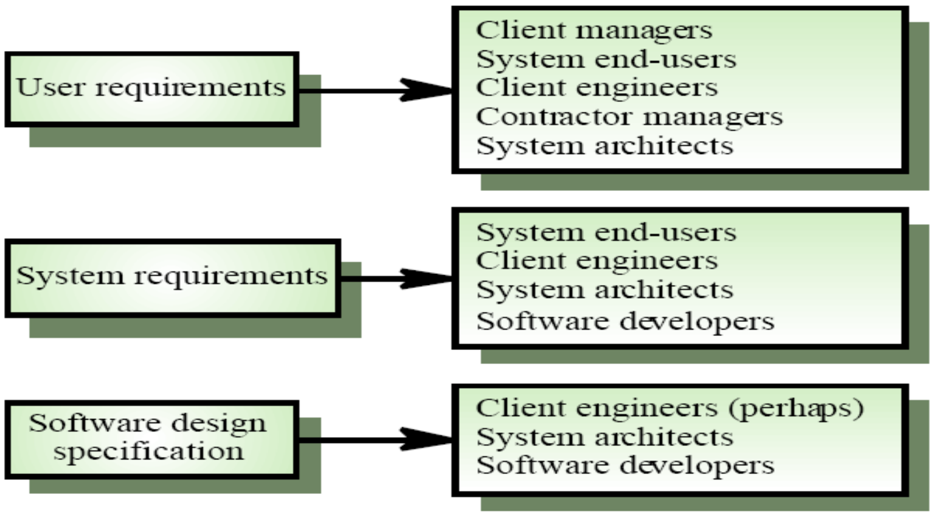


Figure: Readers of different types of specification

<u>Functional and non-functional requirements</u>

- **Functional requirements**
  - ➢ Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
- **Non-functional requirements**
  - ➢ Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
- **Domain requirements**
  - ➢ Requirements that come from the application domain of the system and that reflect characteristics of that domain.

<u>Functional requirements</u>

- Describe functionality or system services.
- Depend on the type of software, expected users and the type of system where the software is used.
- Functional user requirements may be high-level statements of what the system should do but functional system requirements should describe the system services in detail.
- Functional requirements for a software system may be expressed in a number of ways. For 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.

- ➢ The user shall be able to search either all of the initial set of databases or select a subset from it.
- ➢ The system shall provide appropriate viewers for the user to read documents in the document store.
- ➢ 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.

<u>Imprecision in the requirements specification is the cause of many software engineering problems such as:</u>

- Problems arise when requirements are not precisely stated.
- Ambiguous requirements may be interpreted in different ways by developers and users.
- Consider the term 'appropriate viewers'
  - ➢ User intention - special purpose viewer for each different document type;
  - ➢ Developer interpretation - Provide a text viewer that shows the contents of the document.

<u>Requirements completeness and consistency:</u>

- In principle, requirements should be both complete and consistent.
- **Complete**
  - ➢ They should include descriptions of all facilities required.
- **Consistent**
  - ➢ There should be no conflicts or contradictions in the descriptions of the system facilities.
- In practice, it is impossible to produce a complete and consistent requirements document.

<u>Non-functional requirements</u>

- These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.

- Process requirements may also be specified mandating a particular CASE system, programming language or development method.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless.
- Figure shows the classification of non-functional requirements.

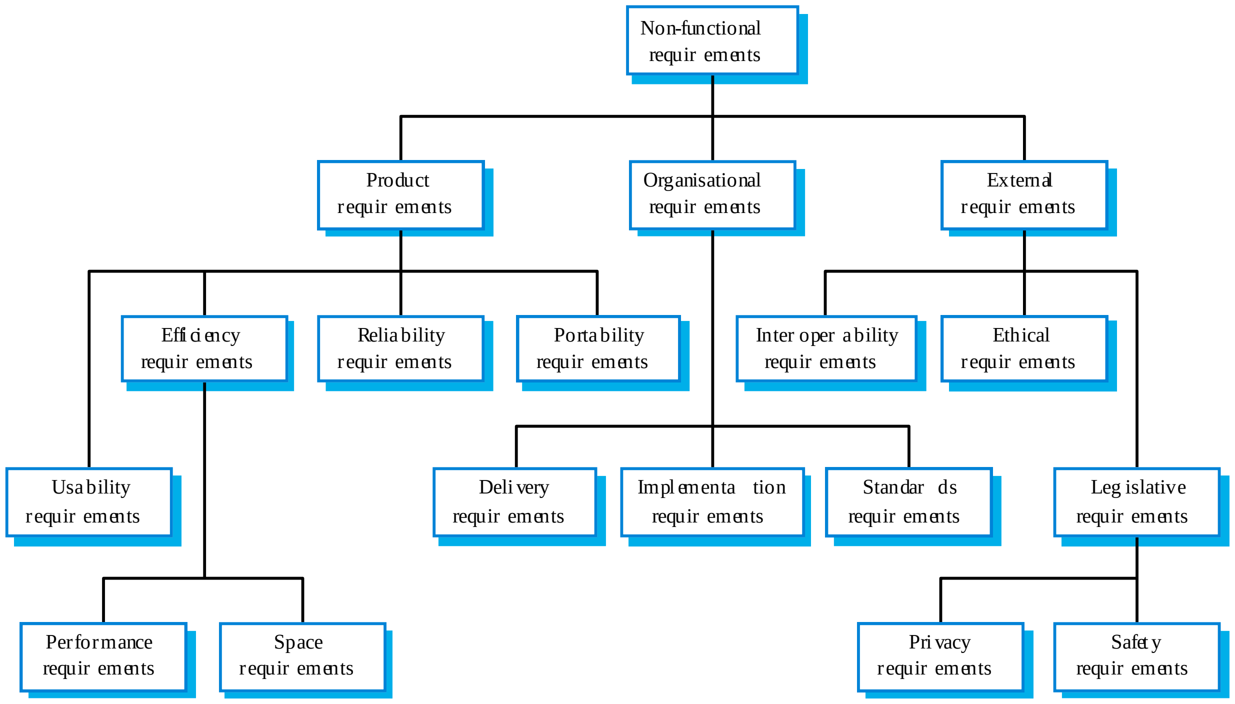


Figure: Types of non-functional requirements

- Non-functional requirements examples are:
  - ➢ **Product requirement**

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

  - ➢ **Organisational requirement**

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

  - ➢ **External requirement**

The system shall not disclose any personal information about customers apart from their name and reference number to the operators of the system.

<u>The types of non-functional requirements are:</u>

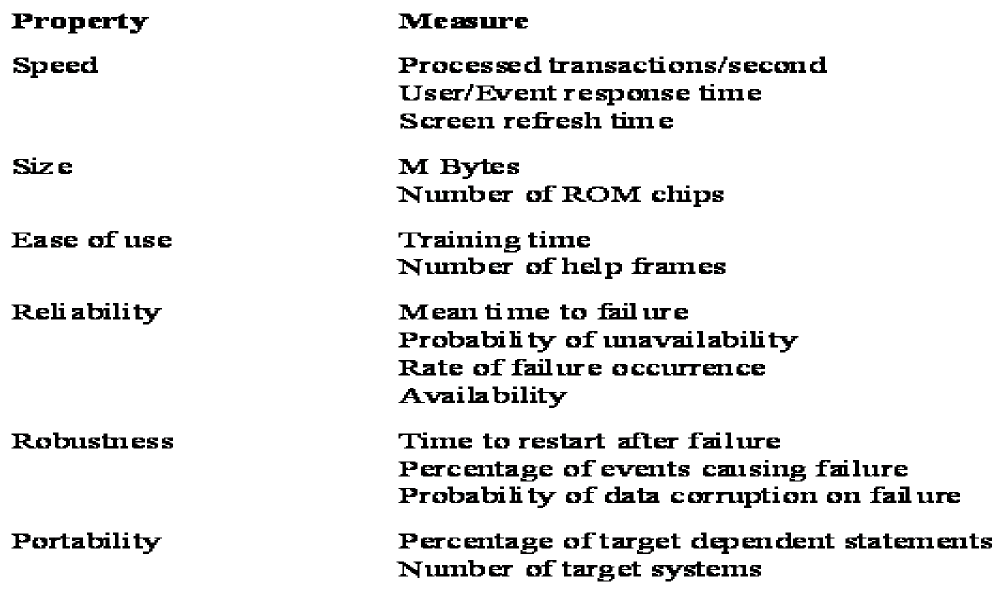
- **Product requirements**
  - ➢ Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- **Organisational requirements**
  - ➢ Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.
- **External requirements**
  - ➢ Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

Requirements documents often include statements of goals mixed with requirements.

- Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to verify.
- Goal
  - ➢ A general intention of the user such as ease of use.
- Verifiable non-functional requirement
  - ➢ A statement using some measure that can be objectively tested.
- Goals are helpful to developers as they convey the intentions of the system users.
- Figure 6.5. This shows a system goal relating to the usability of a traffic control system and is typical of how a user might express usability 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 below shows a number of possible metrics that you can use to specify non-functional system properties.



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

Figure: Metrics for specifying non-functional requirements

<u>Requirements interaction</u>

- Conflicts between different non-functional requirements are common in complex systems.
- Spacecraft system
  - ➢ To minimise weight, the number of separate chips in the system should be minimised.
  - ➢ To minimise power consumption, lower power chips should be used.
  - ➢ However, using low power chips may mean that more chips have to be used. Which is the most critical requirement?

<u>Domain requirements</u>

- Derived from the application domain and describe system characteristics and features that reflect the domain.
- Domain requirements be new functional requirements, constraints on existing requirements or define specific computations.
- If domain requirements are not satisfied, the system may be unworkable.

<u>Example: The LIBSYS system includes a number of domain requirements:</u>

- There shall be a standard user interface to all databases which shall be based on the Z39.50 standard.

- Because of copyright restrictions, some documents must be deleted immediately on arrival. Depending on the user's requirements, these documents will either be printed locally on the system server for manually forwarding to the user or routed to a network printer.

Domain requirements problems:

- **Understandability**
  - ➢ Requirements are expressed in the language of the application domain.
  - ➢ This is often not understood by software engineers developing the system.
- **Implicitness**
  - ➢ Domain specialists understand the area so well that they do not think of making the domain requirements explicit.

User requirements

- Should describe functional and non-functional requirements in such a way that they are understandable by system users who don't have detailed technical knowledge.
- User requirements are defined using natural language, tables and diagrams as these can be understood by all users.

Various problems can arise when requirements are written in natural languages sentences in a text document:

- **Lack of clarity**
  - ➢ Precision is difficult without making the document difficult to read.
- **Requirements confusion**
  - ➢ Functional and non-functional requirements tend to be mixed-up.
- **Requirements amalgamation**
  - ➢ Several different requirements may be expressed together.

Requirement problems

- Database requirements includes both conceptual and detailed information
  - ➢ Describes the concept of a financial accounting system that is to be included in LIBSYS.
  - ➢ However, it also includes the detail that managers can configure this system - this is unnecessary at this level.

To minimise misunderstandings when writing user requirements, it is necessary to follow some simple guidelines:

- Invent a standard format and use it for all requirements.
- Use language in a consistent way. Use shall for mandatory requirements, should for desirable requirements.
- Use text highlighting to identify key parts of the requirement.
- Avoid the use of computer jargon.

System requirements

- Ideally, the system requirements should simply describe the external behaviour of the system and its operational constraints.
- They should not be concerned with how the system should be designed or implemented.
- However, at the level of detail required to completely specify a complex software system, it is impossible, in practice, to exclude all design information. There are several reasons for this:
  - You may have to design an initial architecture of the system to help structure the requirements specification. The system requirements are organised according to the different sub-systems that make up the system.
  - In most cases, systems must interoperate with other existing systems. These constrain the design, and these constraints impose requirements on the new system.
  - The use of a specific architecture to satisfy non-functional requirements as N-version programming to achieve reliability may be necessary.

Requirements and design:

- In principle, requirements should state what the system should do and the design should describe how it does this.
- In practice, requirements and design are inseparable
  - A system architecture may be designed to structure the requirements

- The system may inter-operate with other systems that generate design requirements

- The use of a specific design may be a domain requirement.

- Natural language is often used to write system requirements specifications as well as user requirements.
- However, because system requirements are more detailed than user requirements, natural language specifications can be confusing and hard to understand.

Problems with NL specification:

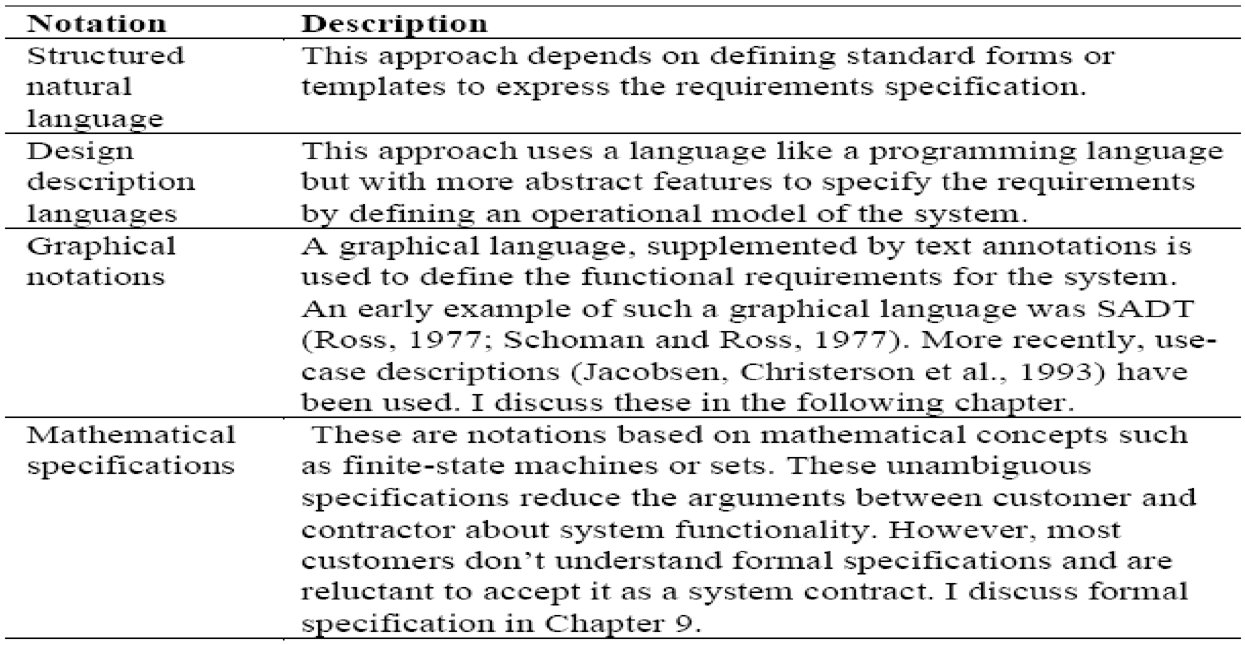
- **Ambiguity**
  - The readers and writers of the requirement must interpret the same words in the same way. NL is naturally ambiguous so this is very difficult.

- **Over-flexibility**
  - The same thing may be said in a number of different ways in the specification.

- **Lack of modularisation**
  - NL structures are inadequate to structure system requirements.

  Alternatives to NL specification.

- It is essential to write user requirements in a language that non-specialists can understand. However, you can write system requirements in more specialized notations shown in figure



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.
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). More recently, use-case descriptions (Jacobsen, Christerson et al., 1993) have been used. I discuss these in the following chapter.
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. I discuss formal specification in Chapter 9.

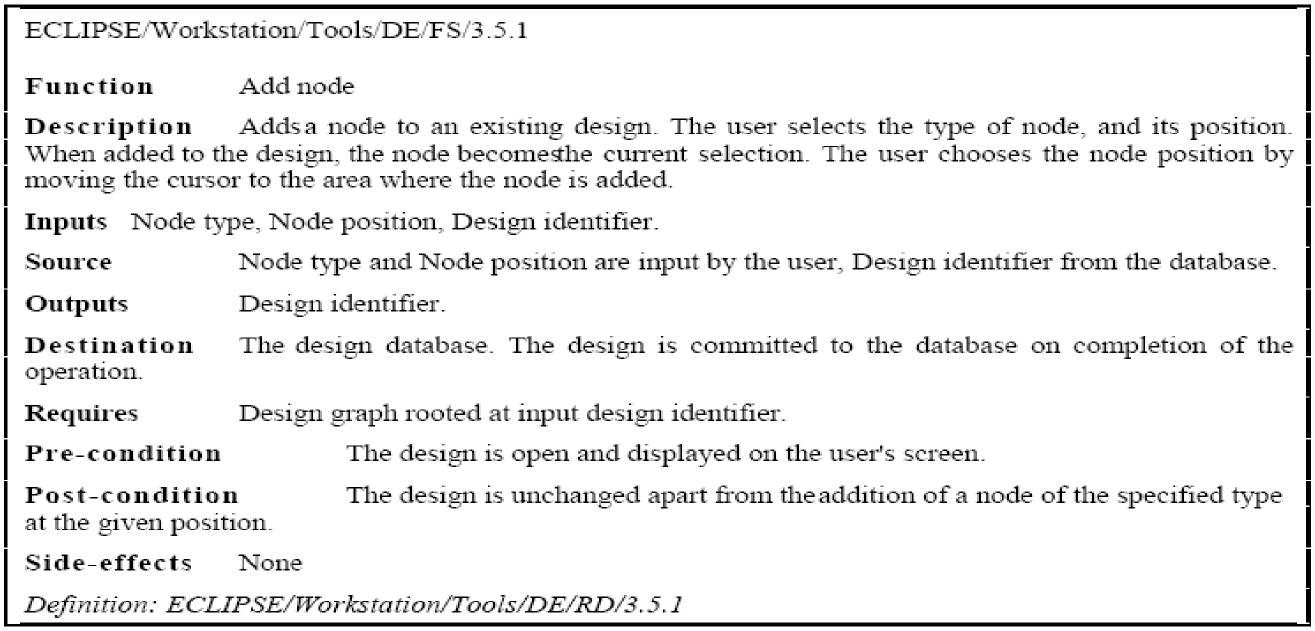
Figure: Notations for requirements specification

Structured language specifications:

- The freedom of the requirements writer is limited by a predefined template for requirements.
- All requirements are written in a standard way.
- The terminology used in the description may be limited.
- The advantage is that the most of the expressiveness of natural language is maintained but a degree of uniformity is imposed on the specification.
- Form-based specifications
- Definition of the function or entity.
- Description of inputs and where they come from.
- Description of outputs and where they go to.
- Indication of other entities required.
- Pre and post conditions (if appropriate).
- The side effects (if any) of the function.

Form-based node specification

- To use a form-based approach to specifying system requirements, we must define one or more standard form or templates to express the requirements.
- An example of such a form -based specification is shown in Figure 6.7


ECLIPSE/Workstation/Tools/DE/FS/3.5.1

Function Add node

Description Addsa node to an existing design. The user selects the type of node, and its position. When added to the design, the node becomesthe current selection. The user chooses the node position by moving the cursor to the area where the node is added.

Inputs Node type, Node position, Design identifier.

Source Node type and Node position are input by the user, Design identifier from the database.

Outputs Design identifier.

Destination The design database. The design is committed to the database on completion of the operation.

Requires Design graph rooted at input design identifier.

Pre-condition The design is open and displayed on the user's screen.

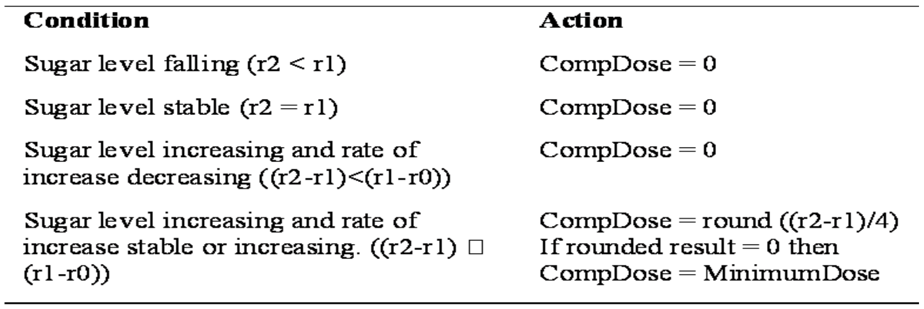
Post-condition The design is unchanged apart from theaddition of a node of the specified type at the given position.

Side-effects None

Definition: ECLIPSE/Workstation/Tools/DE/RD/3.5.1

Figure: Examples of form Based Specification

- When a standard form is used for specifying functional requirements, the following information should be included:
  - Description of the function or entity being specified
  - Description of its inputs and where these come from
  - Description of its outputs and where these go to
  - Indication of what other entities are used (the requires part)
  - Description of the action to be taken
  - 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
  - Description of the side effects (if any) of the operation.
- Figure shows the Tabular specification of computation of insulin does.



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) □ (r1-r0))	CompDose = round ((r2-r1)/4) If rounded result = 0 then CompDose = MinimumDose

Figure Tabular specification of computation of insulin

- Graphical models are most useful when you need to show how state changes or where you need to describe a sequence of actions.
- Figure 6.14 illustrates the sequence of actions when a user wishes to withdraw cash from an automated teller machine (ATM).

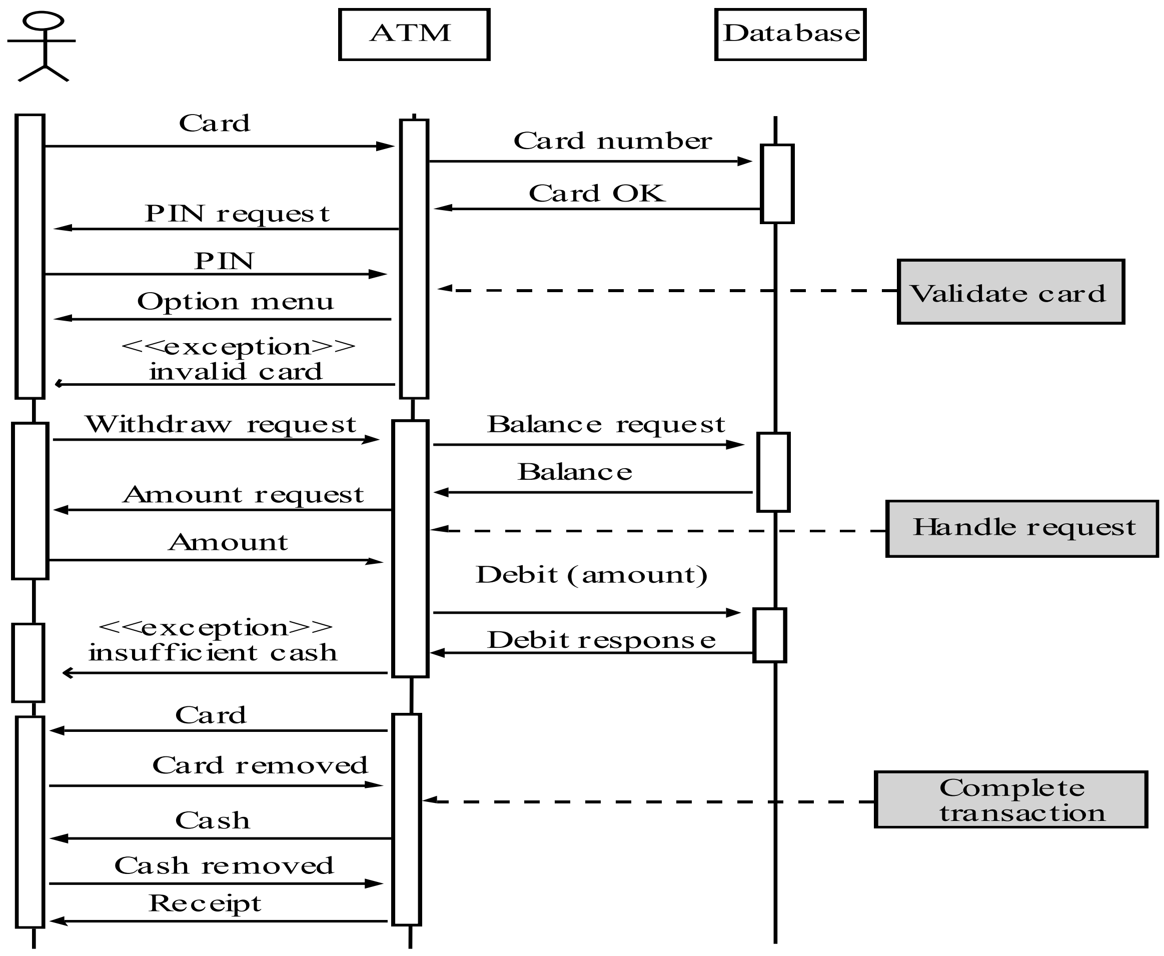


Figure: Sequence diagram of ATM withdrawal

- These show the sequence of events that take place during some user interaction with a system.
- You read them from top to bottom to see the order of the actions that take place.
- Cash withdrawal from an ATM

  - Validate card

  - Handle request

  - Complete transaction

Interface specification

- Most systems must operate with other systems and the operating interfaces must be specified as part of the requirements.
- Three types of interface may have to be defined:

- 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 (APls).

- Data structures that are passed from one sub-system to another. If necessary, program descriptions in Java or C++ can be generated automatically from these descriptions.

- 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. Some programming languages such as Ada (although not Java) support this level of specification.

The software requirements document

- The requirements document is the official statement of what is required of the system developers.
- Should include both a definition of user requirements and a specification of the system requirements.
- 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.
- Users of a requirements document:

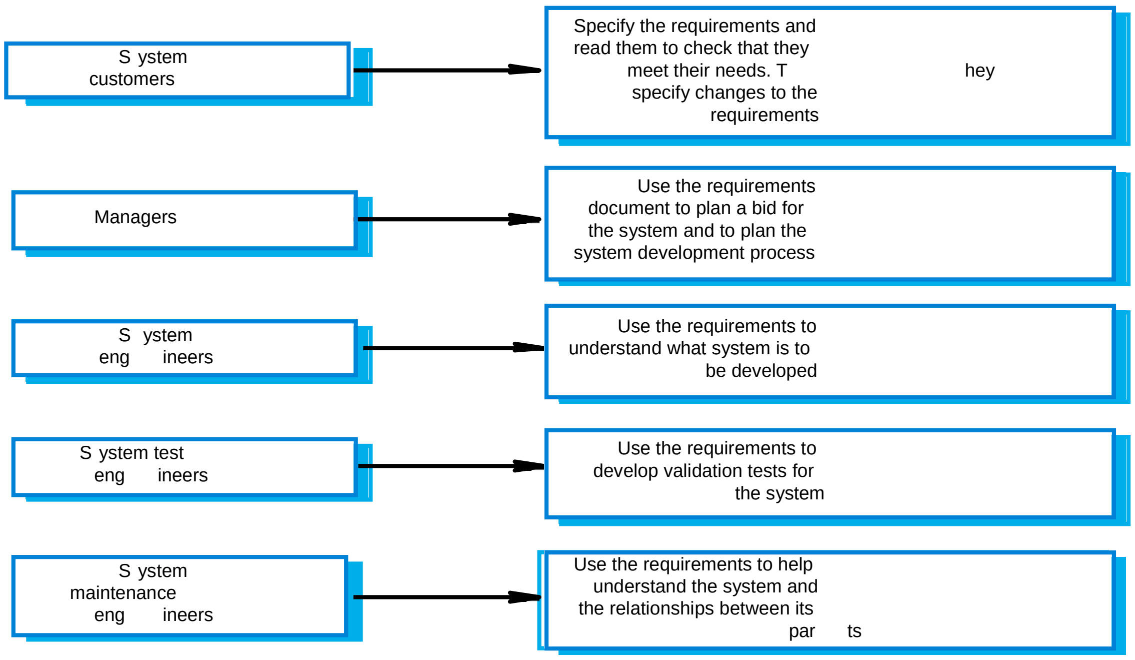

S ystem customers
Specify the requirements and read them to check that they meet their needs. T hey specify changes to the requirements
Managers
Use the requirements document to plan a bid for the system and to plan the system development process
S ystem eng ineers
Use the requirements to understand what system is to be developed
S ystem test eng ineers
Use the requirements to develop validation tests for the system
S ystem maintenance eng ineers
Use the requirements to help understand the system and the relationships between its par ts

Figure: Users of a requirements document

IEEE requirements standard:

- Defines a generic structure for a requirements document that must be instantiated for each specific system.
  - **Introduction**
    - Purpose of the requirements document
    - Scope of the product
    - Definitions, acronyms and abbreviations
    - References
    - Overview of the remainder of the document
  - **General description.**
    - Product perspective
    - Product functions
    - User characteristics
    - General constraints
    - Assumptions and dependencies
  - **Specific requirements:** cover functional, no-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.

- **Appendices.**
- **Index**.

Requirements document structure

- **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 commission1ing 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 service 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 function, etc.

Requirements Engineering Processes

Objectives:

- To describe the principal requirements engineering activities and their relationships
- To introduce techniques for requirements elicitation and analysis
- To describe requirements validation and the role of requirements reviews
- To discuss the role of requirements management in support of other requirements engineering processes.

Requirements engineering processes:

- The processes used for RE vary widely depending on the application domain, the people involved and the organisation developing the requirements.
- However, there are a number of generic activities common to all processes:
  - ➢ Requirements elicitation
  - ➢ Requirements analysis
  - ➢ Requirements validation
  - ➢ Requirements management
- The activities shown in Figure are concerned with the discovery, documentation and checking of requirements:

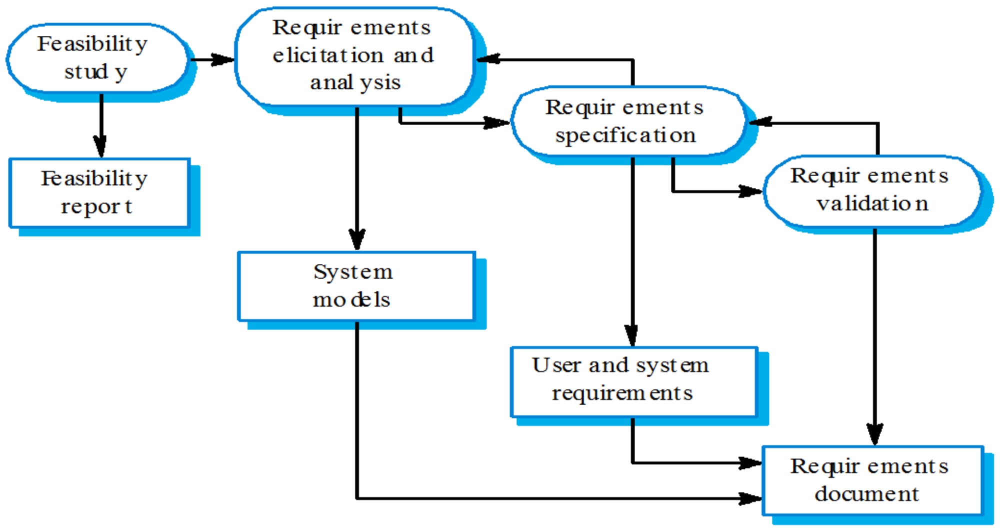


Figure: The requirements engineering process

- The process of managing these changing requirements is called **requirements management.**
- Figure present an alternative perspective on the requirements engineering process. This presents the process as a three-stage activity where the activities are organised as an iterative process around a spiral.

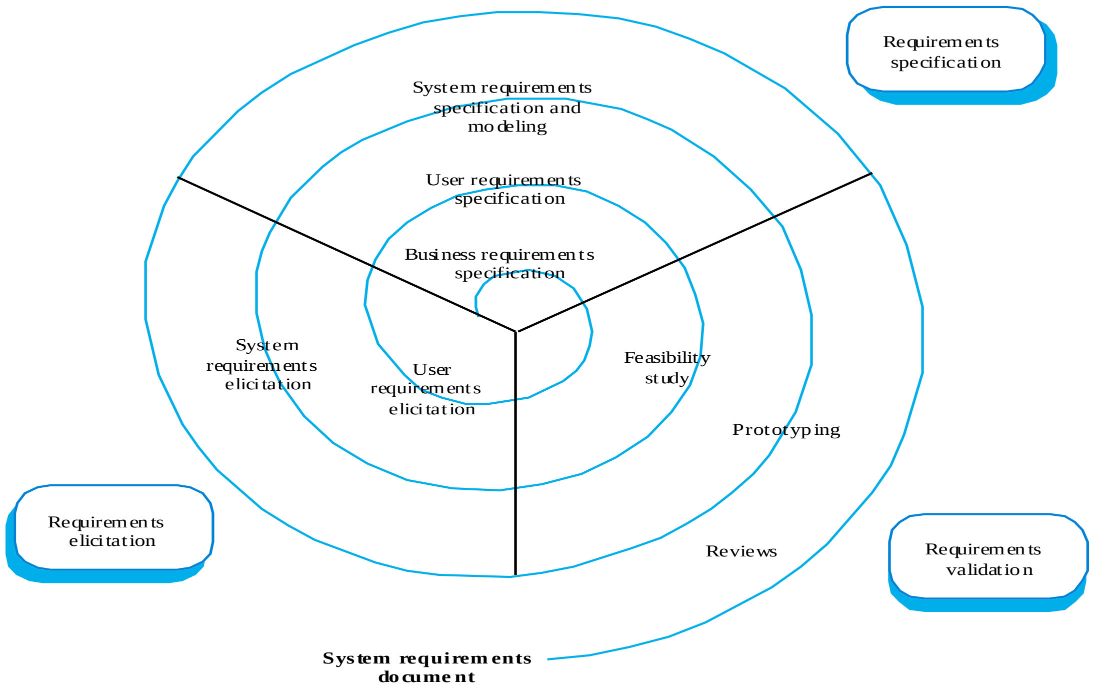

Requirements specification
System requirements specification and modeling
User requirements specification
Business requirements specification
System requirements elicitation
User requirements elicitation
Feasibility study
Prototyping
Reviews
Requirements elicitation
Requirements validation
System requirements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

Figure: Alternative approaches for requirements engineering process

Feasibility studies

- A feasibility study decides whether or not the proposed system is worthwhile.
- A feasibility study is a short, focused study that aims to answer a number of questions:
  - Does the system contribute to the overall objectives of the organisation?
  - Can the system be implemented using current technology and within given cost and schedule constraints?
  - Can the system be integrated with other systems which are already in place?

Feasibility study implementation:

- Based on information assessment (what is required), information collection and report writing.
- Questions for people in the organisation
  - What if the system wasn't implemented?
  - What are current process problems?
  - How will the proposed system help?
  - What will be the integration problems?
  - Is new technology needed? What skills?
  - What facilities must be supported by the proposed system?

Requirements elicitation and analysis

- Sometimes called requirements elicitation or requirements discovery.
- Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints.
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called stakeholders.
- Eliciting and understanding stakeholder requirements is difficult for several reasons:
  - Stakeholders don't know what they really want.

- ➢ Stakeholders express requirements in their own terms.
- ➢ Different stakeholders may have conflicting requirements.
- ➢ Organisational and political factors may influence the system requirements.
- ➢ The requirements change during the analysis process. New stakeholders may emerge and the business environment change.

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

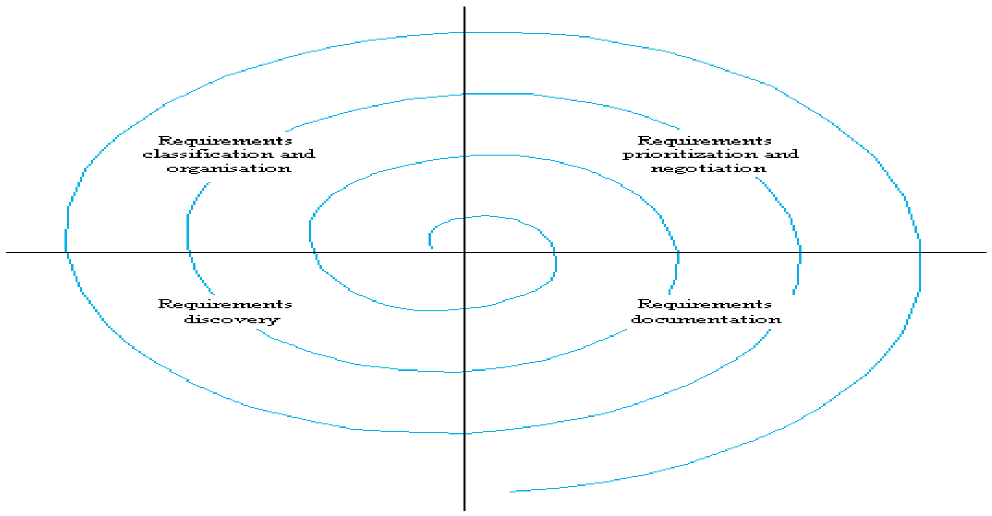

Requirements classification and organisation
Requirements prioritization and negotiation
Requirements discovery
Requirements documentation

Figure: The requirements elicitation and analysis process

<u>Process activities involve in above figure are:</u>

- **Requirements discovery**
  - ➢ Interacting with stakeholders to discover their requirements. Domain requirements are also discovered at this stage.

- **Requirements classification and organisation**
  - ➢ Groups related requirements and organises them into coherent clusters.

- **Prioritisation and negotiation**
  - Prioritising requirements and resolving requirements conflicts.
- **Requirements documentation**
  - Requirements are documented and input into the next round of the spiral.
- Figure 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.

Requirements discovery

- The process of gathering information about the proposed and existing systems and distilling the user and system requirements from this information.
- Sources of information include documentation, system stakeholders and the specifications of similar systems.
- Stakeholders range from system end-users through managers and external stakeholders such as regulators who certify the acceptability of the system. For example, system stakeholders for a bank ATM include:
  - **Bank customers:** Current bank customers who receive services from the system.
  - **Representatives of other banks:** Representatives from other banks who have reciprocal agreements that allow each other's ATMs to be used
  - **Bank managers:** Managers of bank branches who obtain management information from the system.
  - **Counter staff:** Counter staff at bank branches who are involved in the day-to-day running of the system
  - **Database administrators:** Database administrators who are responsible for integrating the system with the bank’s customer database

- **Security managers:** Bank security managers who must ensure that the system will not pose a security hazard

- **Marketing department:** The bank's marketing department who are likely be interested in using the system as a means of marketing the bank

- **Hardware and software maintenance engineers:** Hardware and software maintenance engineers who are responsible for maintaining and upgrading the hardware and software

- **Banking regulators:** National banking regulators who are responsible for ensuring that the system conforms to banking regulations.

<u>Viewpoints:</u>

- Viewpoints are a way of structuring the requirements to represent the perspectives of different stakeholders. Stakeholders may be classified under different viewpoints.
- This multi-perspective analysis is important as there is no single correct way to analyse system requirements.

- Viewpoints can be used as a way of classifying stakeholders and other sources of requirements. There are three generic types of viewpoint:
- Interactor viewpoints
  - People or other systems that interact directly with the system. In an ATM, the customer's and the account database are interactor VPs.

- Indirect viewpoints
  - Stakeholders who do not use the system themselves but who influence the requirements. In an ATM, management and security staff are indirect viewpoints.

- Domain viewpoints
  - Domain characteristics and constraints that influence the requirements. In an ATM, an example would be standards for inter-bank communications.

Viewpoint identification:

- Identify viewpoints using following:
    - Providers and receivers of system services
    - Systems that interact directly with the system being specified
    - Regulations and standards
    - Sources of business and non-functional requirements
    - Engineers who have to develop and maintain the system
    - Marketing and other business viewpoints.

LIBSYS viewpoint hierarchy:

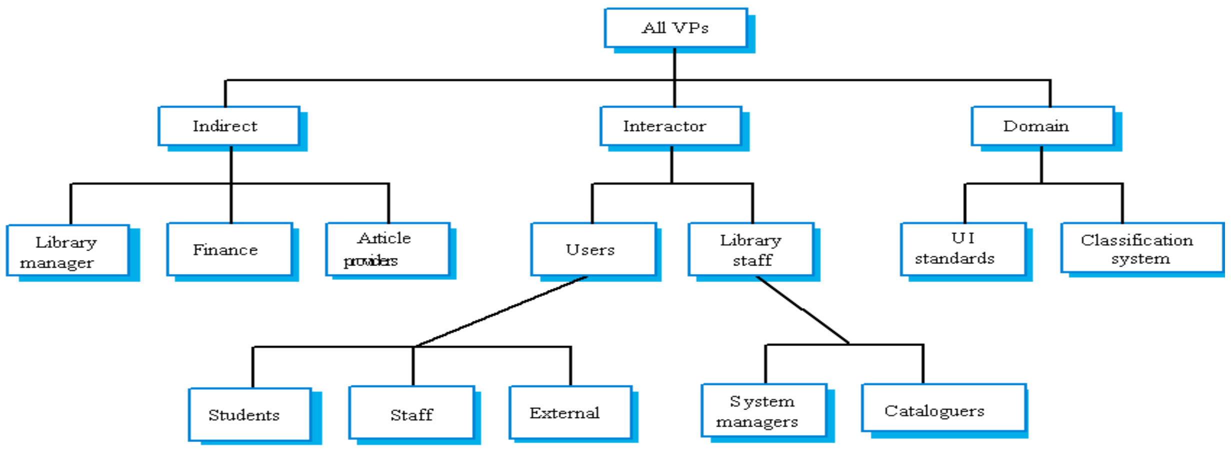


Figure: Viewpoints in L1BSYS

Interviewing:

- In formal or informal interviewing, the RE team puts questions to stakeholders about the system that they use and the system to be developed.
- There are two types of interview:
    - **Closed** interviews where a pre-defined set of questions are answered.
    - **Open** interviews where there is no pre-defined agenda and a range of issues are explored with stakeholders.

- Interviews in practice, normally a mix of closed and open-ended interviewing.
- Interviews are good for getting an overall understanding of what stakeholders do and how they might interact with the system.
- It is hard to elicit domain knowledge during interviews for two reasons:
  - Requirements engineers cannot understand specific domain terminology
  - Some domain knowledge is so familiar that people find it hard to articulate or think that it isn't worth articulating.
- Effective interviewers have two characteristics:
  - Interviewers should be open-minded, willing to listen to stakeholders and should not have pre-conceived ideas about the requirements.
  - They should prompt the interviewee with a question or a proposal and should not simply expect them to respond to a question such as 'what do you want'.

Requirements validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Requirements error costs are high so validation is very important
  - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.
- During the requirements validation process, checks should be carried out on the requirements in the requirements document. These checks include:
  - **Validity checks:** Does the system provide the functions which best support the customer's needs?
  - **Consistency checks:** Are there any requirements conflicts?
  - **Completeness checks:** Are all functions required by the customer included?

- **Realism checks:** Can the requirements be implemented given available budget and technology
- **Verifiability checks:** Can the requirements be checked?

- A number of requirements validation techniques can be used in conjunction or individually:
    - **Requirements reviews**: Systematic manual analysis of the requirements.
    - **Prototyping:** Using an executable model of the system to check requirements.
    - **Test-case generation**: Developing tests for requirements to check testability.

Requirements reviews:

- Regular reviews should be held while the requirements definition is being formulated.
- Both client and contractor staff should be involved in reviews.
- Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.
- 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:
    - **Verifiability**: Is the requirement realistically testable?
    - **Comprehensibility**: Is the requirement properly understood?
    - **Traceability:** Is the origin of the requirement clearly stated?
    - **Adaptability:** Can the requirement be changed without a large impact on other requirements?

Requirements management

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development.

- Requirements are inevitably incomplete and inconsistent
  - ➢ New requirements emerge during the process as business needs change and a better understanding of the system is developed;
  - ➢ Different viewpoints have different requirements and these are often contradictory.
- Requirements change
- The priority of requirements from different viewpoints changes during the development process.
- System customers may specify requirements from a business perspective that conflict with end-user requirements.
- The business and technical environment of the system changes during its development.

Enduring and volatile requirements

- Requirements evolution during the RE process and after a system has gone into service is inevitable.
- Figure shows the Requirements evolution of a process.

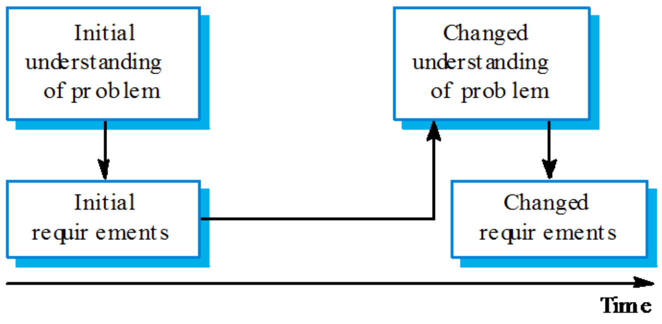
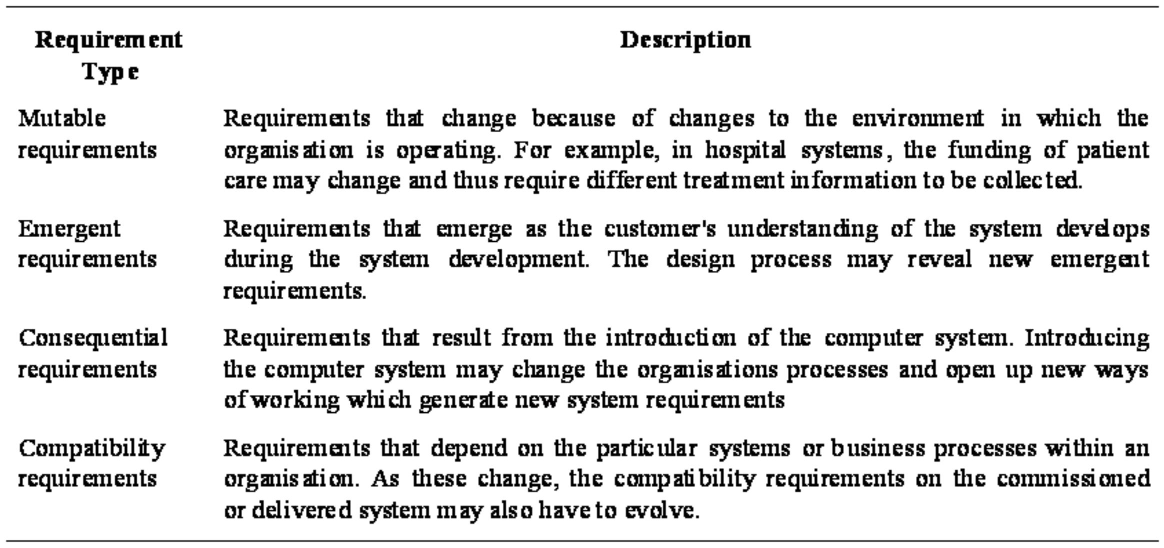

Initial understanding of problem
Changed understanding of problem
Initial requirements
Changed requirements
Time

Figure: Requirements evolution

- From an evolution perspective, requirements fall into two classes:
  - ➢ **Enduring requirements:** Stable requirements derived from the core activity of the customer organisation. E.g. a hospital will always have doctors, nurses, etc. May be derived from domain models
  - ➢ **Volatile requirements:** Requirements which change during development or when the system is in use. In a hospital, requirements derived from health-care policy

Requirements classification

- Harker and others (Harker, et al., 1993) have suggested that volatile requirements fall into five classes. Using these as a base, I have developed the classification shown in Figure



Requirement Type	Description
Mutable requirements	Requirements that 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 that 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 that result from the introduction of the computer system. Introducing the computer system may change the organisations processes and open up new ways of working which generate new system requirements
Compatibility requirements	Requirements that 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.

Figure: classifications of volatile requirements

Requirements management planning

- Planning is an essential first stage in the requirements management process. Requirements management is very expensive.
- During the requirements engineering process, you have to plan:
  - ➢ **Requirements identification:** How requirements are individually identified;
  - ➢ **A change management process:** The process followed when analysing a requirements change;
  - ➢ **Traceability policies: The** amount of information about requirements relationships that is maintained.
  - ➢ **CASE tool support:** The tool support required to help manage requirements change.

Requirements change management

- Should apply to all proposed changes to the requirements.
- There are three principal stages to a change management process:
  - **Problem analysis and change specification:** Discuss requirements problem and propose change.
  - **Change analysis and costing**: Assess effects of change on other requirements.
  - **Change implementation**: Modify requirements document and other documents to reflect change.

Change management

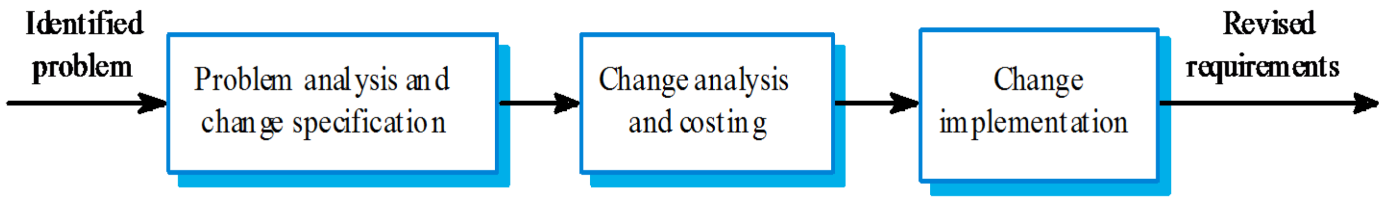

Identified problem
Problem analysis and change specification
Change analysis and costing
Change implementation
Revised requirements

Figure: Requirements change management