Unit-III

Unit lll : Requirement Engineering

Requirements Elicitation: Concept of Software Requirement, Categories and types of Requirements, Elicitation Techniques- real life application case study.

Requirements Analysis and Documentation: Textual and Graphical Documentation ,UML models: Use Case Diagram and class diagram, data modelling, data and control flow model, behavioural modelling using state diagrams - real life application case study, Software Requirement Specifications (SRS).

Software Requirement Engineering

- **It is**
  - **understanding the requirements of the customers and**
  - **documenting the requirements (Developer can refer it or customer can also validate the requirement)**

- Requirement:
  - A requirement is a
    - feature of the system or a description of something the system is capable of doing in order to fulfil the system's purpose.
    - **What to do and it does not tell us how to do?**

- Work Product:
  - Requirement engineering produces a **large document written in natural language** which contains the **description of what system will do without specifying how it will do it**.
  - Called as **Software Requirement Specification**.
- Input to requirement Engineering: Problem statement prepared by customer
  - Overview of the existing system + New or additional functionality to be added

Importance of Requirement Engineering

- Without well written requirements:
  - Developers will not know what to do.
  - the customers may not be consistent about their requirement.
  - It becomes difficult for the customer to validate the software/or to accept the produced software

Four basic steps of RE

1. Requirement Elicitation
2. Requirement Analysis
3. Requirement Documentation
4. Requirement review or requirement validation

Four basic steps of RE

- Requirement Elicitation: Requirements are identified with the help of customer (existing system might be used)

- Requirement Analysis: requirements are analysed to:
  - Find inconsistencies or contradictions (Ex: Software requirement changes)
  - Omissions (not collected proper requirement, **requirements are not collected for some of the cases**)

  Some times prioritization of the requirements is done in this step.

- Requirement Documentation: End product of first two steps
  - it leads to the preparation of SRS and
  - this SRS becomes the foundation of design of software.

- Requirement review or requirement validation:
  - To improve the quality of SRS
  - Once SRS is finalized, is should be shown to the customer for validation.

Software Requirement Elicitation (Requirement Gathering)

- Requirement elicitation is an activity that helps us to understand **what problem has to be solved and what customer expects from the software.**

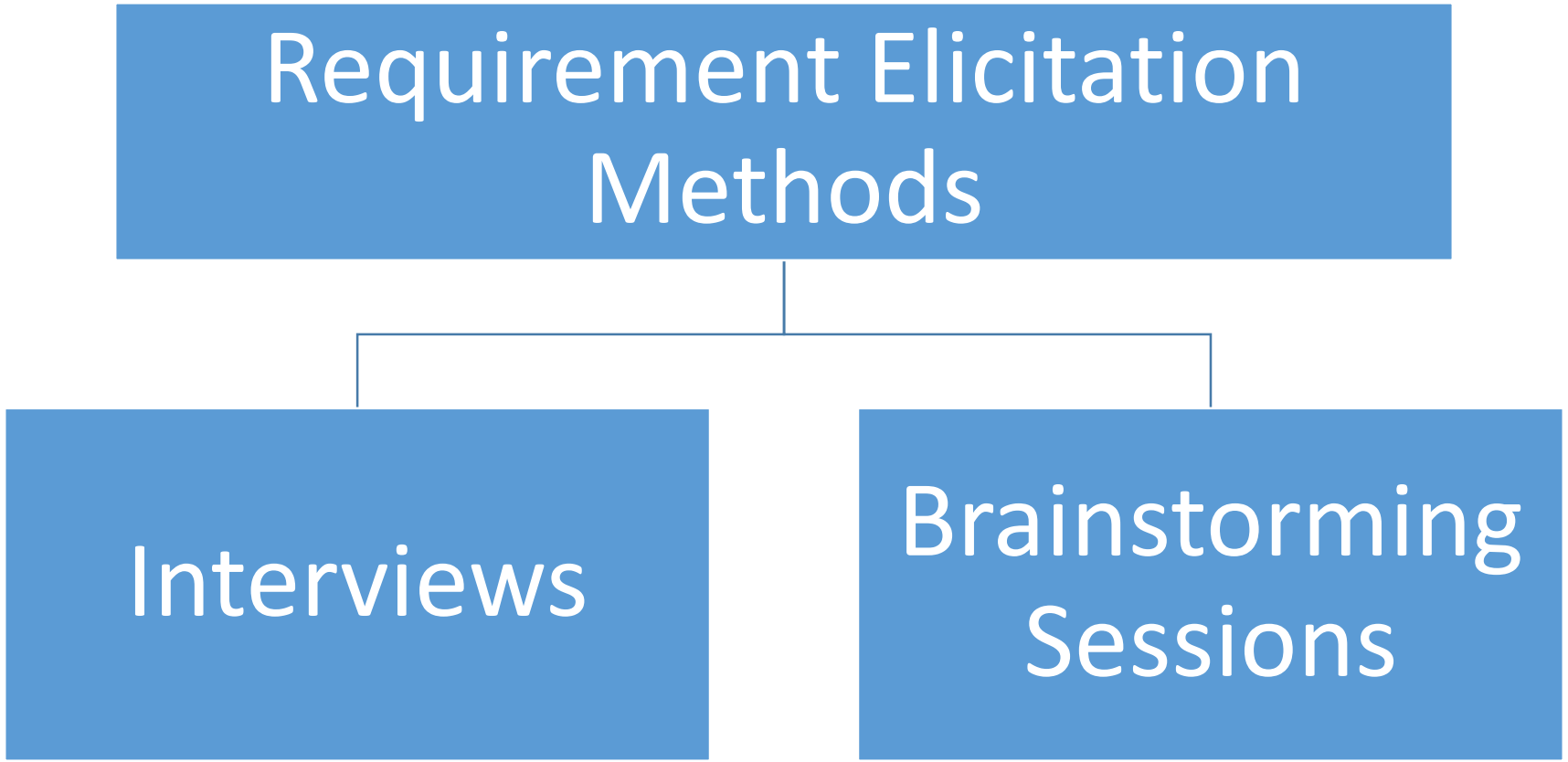
- **Foundation:**
  - Effective communication between customer and the developer

- **Method:**
  - Developer questions the customer, the customer responds to developer, cross questioning on response (formal or informal method)

- Hurdles in Requirement elicitation:
  - Misunderstanding/conflicts (between customer and developer),
  - communication gap may arise (omission of requirements)

- Reason for conflicts between concerned people:
  - Developer is efficient in the knowledge of his own development/domain while the customer is efficient only in his domain (which is different from developer)

  - Lack of proper communication skills between customer and developer. (developer talking in technical terms but customer is not getting it)


Requirement Elicitation Methods
Interviews
Brainstorming Sessions

Requirement Elicitation Methods

- Interviews:
  - Purpose: Understanding the expectations from the software : Requirement engineer act as mediator between customer and development team.
  - Correct approach: Both the parties should be open minded, cooperative, patient enough, flexible enough etc.
  - Categories of interview:
    - Open ended:
      - There is no pre-set agenda (no –predefined list of questions prepared)
    - Structured:
      - There is pre-set agenda

- Type of stakeholders (What type of people should be interviewed)
  - Entry level personnel:
    - People who do not have lot of domain knowledge, but they should be interviewed. They have creative ideas or approaches.
  - Mid-level Stakeholders:
    - Experienced people with good domain knowledge
    - They know criticality of the project
  - Managers and Higher Management
    - Give useful insights about the project
    - What organization expects from software
  - Users of Software:
    - Most important- they will be using the software maximum number of times.
    - Directly interact with the software.

Brainstorming Sessions:

- Group discussion techniques
- Promotes creative thinking and new ideas
- Platform to express and share your views, your expectations and difficulties in implementation (customers and developers both participation)
- Facilitator
  - To avoid ego clashes or conflicts
  - To encourage people to participate as much as possible

- Work Product:
  - Document: All ideas are documented to be visible to each participant (using white board or projectors)
  - Detailed report, containing each idea in simple language, is prepared and reviewed by facilitators. (No technical terms are used)
  - At end, the document is prepared with list of requirements and its priorities

Desirable Characteristics of SRS

- Consistent:
  - No conflicts between the requirement
  - Every requirement must be specified in using standard terminology.
  - Example: in SRS, there is a difference between the input of the modules, in first part it is written two and in other part it is written three.

- Correct:
  - What is stated is exactly what is desired.
  - Expected functionality matches the requirement present in SRS.

- Unambiguous:
  - Every stated requirement has only one unique meaning. (if is read by any developer or customer)
  - Words with multiple meaning (these should be specified with their intended meaning)
  - SRS language can be used;
    - Adv: Many Language processors exists, which tells different kinds of errors, which reduces ambiguity
    - Disadvantage: Developer/customer should be capable enough to understand the SRS language

- Complete:
  - Includes all functional and non-functional requirements along with specified constraints.
  - It also specifies expected output from all kinds (valid or invalid) inputs that can be provided by the user.

- Traceable:
  - Origin of each requirement should be clear. (Why this requirement? Whom to contact?)
  - Important because future referencing may be required for development or for maintenance.

- Verifiable:
  - SRS is verifiable iff each requirement is verifiable.
  - Ambiguous requirements can never be verified.
  - Words like “Fast”, “Good”, should not be used, as it can not be verified.

- Modifiable:
  - We should be able to change the requirement, without affecting the structure of it.
  - Correctly modify (keep all cross references, mention all cross references in SRS)

- Ranked for stability/Importance:
  - Each requirement should have a ranking for its stability and importance

Types of Requirements

- Functional and non-functional
- User and system requirement
- Interface specification

Stakeholders

- It refers to anyone with direct or indirect influence on system requirement.

- Two types
  - End user: who will interact with the system.

  - Anyone except the user: who will be affected by the software that is developed.

User Requirements

- Contains both functional and non-functional
- Written for users who are not experts of software field.
- Highlight the overview of the system without design description.
- It Specifies:
  - Functional and non-functional requirements
  - Constraints
  - Quality
  - External behaviour

System Requirement

- It specify both functional and non-functional.
- Expanded form of user requirement.
- It is used as a input to designers, so that can prepare the design documents.
- It is a more detailed descriptions of software system’s functions, services and operational constraints.
- This documents should define exactly what is to be implemented.

User and System Requirement Example:

- Mental Healthcare Patient Management System
  - How user requirement may be expanded into several system requirement.

User Requirement Definition

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

System Requirements Specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost, and the prescribing clinics shall be generated.

1.2 The system shall automatically generate the report for printing after 17.30 on the last working day of the month.

1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed, and the total cost of the prescribed drugs.

1.4 If drugs are available in different dose units (e.g., 10 mg, 20 mg) separate reports shall be created for each dose unit.

1.5 Access to all cost reports shall be restricted to authorized users listed on a management access control list.

Functional Requirement

- List of actual services which a user will provide.
- It specify product features.
- What are the expectations from the software.
- Example:
  - Features of the software which the client demands.
  - Business rules of the particular organization for which developing software
- It also specify what the software should not do.

Example:
Functional requirements for the MHC-PMS system

- Used to maintain information about patients receiving treatment for mental health problems:

1. A user shall be able to **search the appointments lists for all clinics**.

2. The system shall **generate each day, for each clinic, a list of patients who are expected to attend appointments that day.**

3. Each **staff member** using the system shall be **uniquely identified** by his or her **eight-digit employee number**

Non-Functional Requirements

- How the system should behave while performing the operation
- These are the constraints on the services which system is offering.
  Ex: timing of operation, way to response in particular condition.
- Known as quality attributes.
- Two different perspective:
  - User: High performance, high reliability, good interface etc.
  - Developers: Maintainability, testability, Portability etc.

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

- **failing to meet a non-functional requirement** can mean that the **whole system is unusable**.
  - For example, if an aircraft system does not meet its reliability requirements, it will not be certified as safe for operation

  - if an embedded control system fails to meet its performance requirements, the control functions will not operate correctly.

Relating components to non-functional requirements

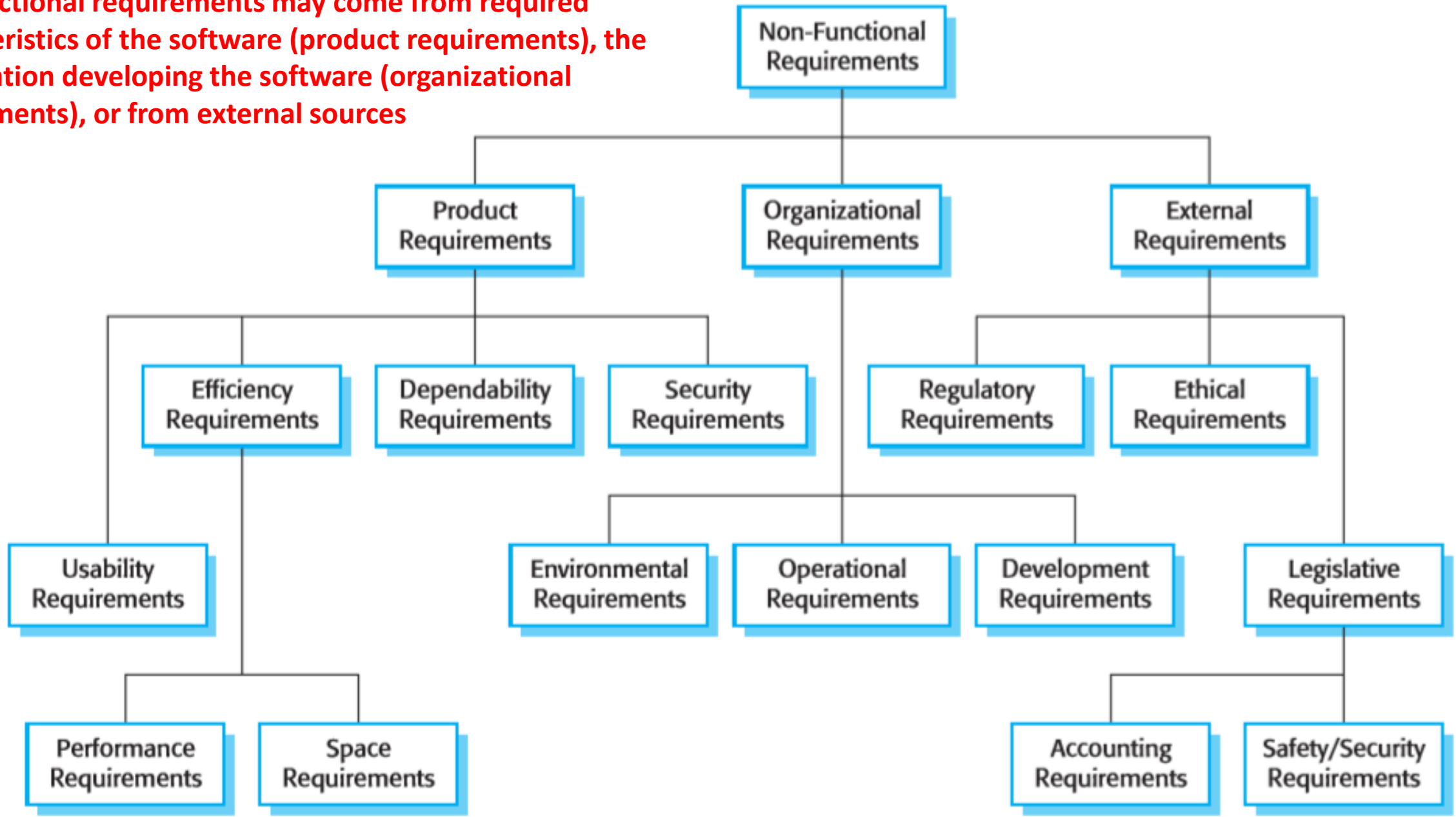
- It is often more difficult to relate components to non-functional requirements.

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

  - For example, **to ensure that performance requirements are met**, **you may have to organize the system to minimize communications between components.**

  - security requirement, may generate a number of related functional requirements

non-functional requirements may come from required characteristics of the software (product requirements), the organization developing the software (organizational requirements), or from external sources



Product requirements

- These requirements specify or constrain the behaviour of the software.

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

Organizational requirements

- These requirements are broad system requirements.
- It is derived from policies and procedures in the customer's and developer's organization.
- Examples includes:
  - **operational process requirements** that define how the system will be used,
  - **development process requirements** that specify the programming language, the development environment or process standards to be used, and
  - **environmental requirements** that specify the operating environment of the system.

External requirements

- This broad heading covers all requirements that are derived from factors external to the system and its development process.
- These may include regulatory requirements that set out what must be done for the system to be approved for use by a regulator, such as a central bank;
- legislative requirements that must be followed to ensure that the system operates within the law; and
- Ethical requirements that ensure that the system will be acceptable to its users and the general public.

PRODUCT REQUIREMENT

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

ORGANIZATIONAL REQUIREMENT

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

EXTERNAL REQUIREMENT

The system shall implement patient privacy provisions

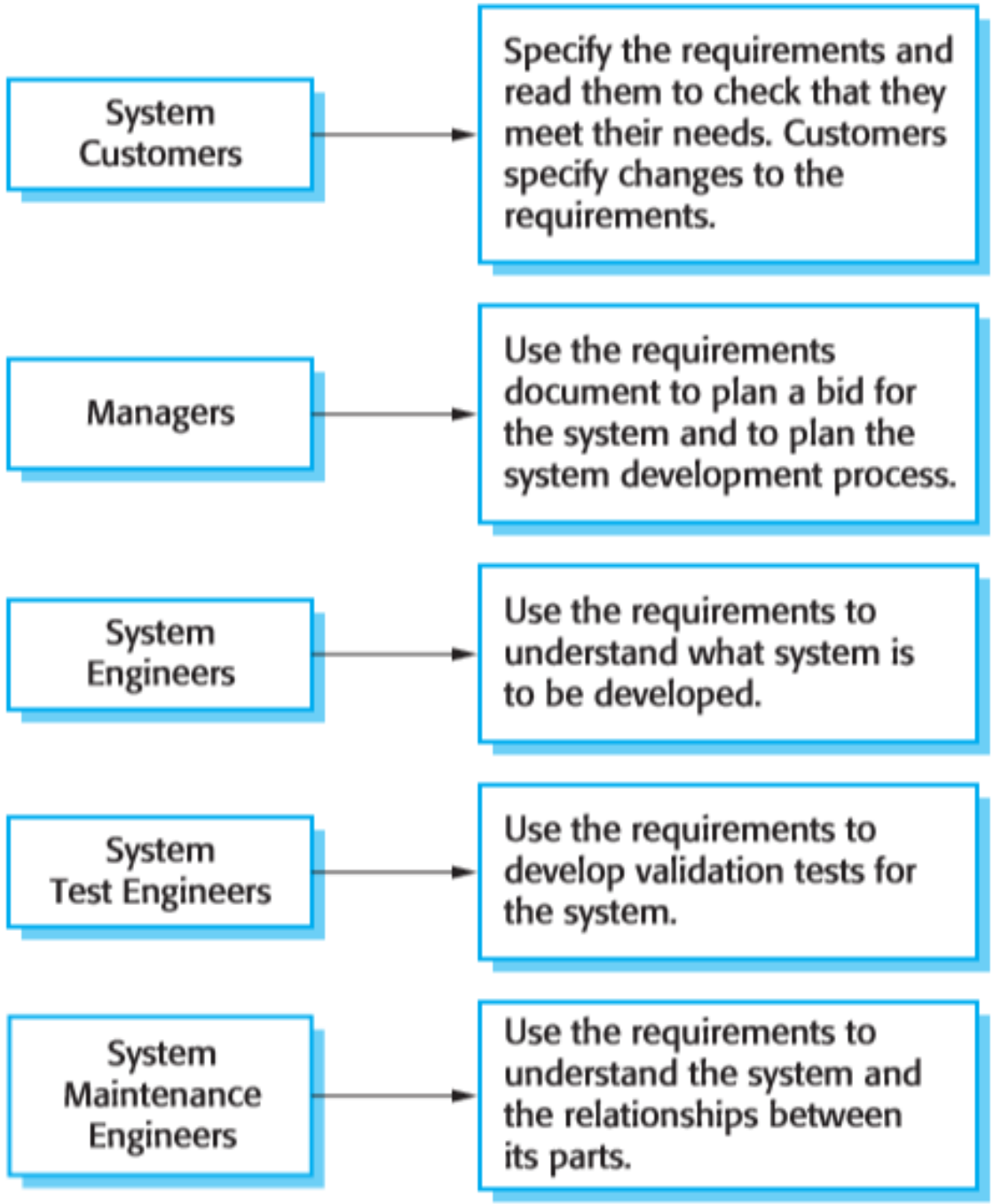
Non-functional requirement Metrics

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

Software Requirement Documents

- Also called as SRS is an official statement of what the system developers should implement.
- It includes: user requirements and specifications of system requirement.
- **Software requirements specification (SRS)** includes in-depth descriptions of the software that will be developed.
- A **system requirements specification (SyRS)** collects information on the requirements for a system.
- “Software” and “system” are sometimes used interchangeably as SRS. But, a **software requirement specification provides greater detail than a system requirements specification.**

Users of a Requirements Document



Structure of a Requirements Documents

- Preface
- Introduction
- Glossary
- User Requirements Definition
- System Architecture
- System Requirements Specification
- System Models
- System Evolution
- Appendices
- Index

Preface:

- This should define the expected readership of the document and describe its version history,
- including a rationale for the creation of a new version and
- a summary of the changes made in each version.

Introduction

- This should describe the **need for the system**.
- It should briefly describe the **system's functions** and explain **how it will work with other systems**.
- It should also describe **how the system fits into the overall business or strategic objectives.**

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

- Description of the services provided for the user.
- The non-functional system requirements should also be described in this section.
- This description may use natural language, diagrams, or other notations that are understandable to customers.
- Product and process standards that must be followed should be specified.

System Architecture

- This chapter should present a high-level overview of the anticipated system architecture,
- showing the distribution of functions across system modules.
- Architectural components that are reused should be highlighted.

System Requirements Specification/Detailing of specific requirements

- This should describe the functional and non-functional requirements in more detail.
- If necessary, further detail may also be added to the non-functional requirements.
- Interfaces to other systems may be defined

System Models

- This might include graphical system models showing the relationships between the system components, the system, and its environment.

- Examples of possible models are object models, data-flow models, or semantic data models

System Evolution

- This should describe the **fundamental assumptions on which the system is based**, and **any anticipated changes due to hardware evolution, changing user needs, and so on.**

- This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.

Appendices

- These should provide detailed, specific information that is related to the application being developed;

- for example, hardware and database descriptions.

  - Hardware requirements define the minimal and optimal configurations for the system.

  - Database requirements define the logical organization of the data used by the system and the relationships between data.

Index

- Several indexes to the document may be included.

- As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.

Writing Requirement Specification

- Natural Language sentences:
  - The requirements are written using numbered sentences in natural language.
  - Each sentence should express one requirement
- Structured natural language:
- Design description languages:
- Graphical Notations:
- Mathematical specifications:

Natural Language Specifications

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

1. Invent a standard format

   Less omissions and requirements are easier to check.

2. Use language consistently to distinguish between mandatory and desirable requirements.
   - Mandatory requirements are requirements that the system must support.
   - Desirable requirements are not essential.

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

4. Do not assume that readers understand technical software engineering language.

   You should avoid the use of jargon, abbreviations, and acronyms.

5. Whenever possible, you should try to associate a rationale with each user requirement.

   The rationale should explain why the requirement has been included.

Natural language Specification - Example

Insulin Pump Control System:

An insulin pump is a medical system that simulates the operation of the pancreas (an internal organ).

The software controlling this system is an embedded system, which collects information from a sensor and controls a pump that delivers a controlled dose of insulin to a user.

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

3.6 The system shall run a self-test routine every minute with the conditions to be tested and the associated actions defined in Table 1. (*A self-test routine can discover hardware and software problems and alert the user to the fact the normal operation may be impossible.*)

Structured Specification

- In Structured natural language, all requirements are written in a standard way.

- Structured language notations use templates to specify system requirements.

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.
Requirements	Two previous readings so that the rate of change of sugar level can be computed.
Pre-condition	The insulin reservoir contains at least the maximum allowed single dose of insulin.
Post-condition	r0 is replaced by r1 then r1 is replaced by r2.
Side effects	None.

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

1. A description of the function or entity being specified.

2. A description of its inputs and where these come from.

3. A description of its outputs and where these go to.

4. Information about the information that is needed for the computation.

5. A description of the action to be taken.

6. Preconditions and post conditions of the functions

7. A description of the side effects (if any) of the operation

Advantages of Structured Specifications

- Variability in the specification will be reduced.
- Requirements are organized more effectively.
- Sometimes still it is difficult to write unambiguous and clear requirement.
  - We can add extra information to natural language requirements. Ex. Tables, graphs etc,

- **Tables** are particularly useful when there are a n**umber of possible alternative situations.**

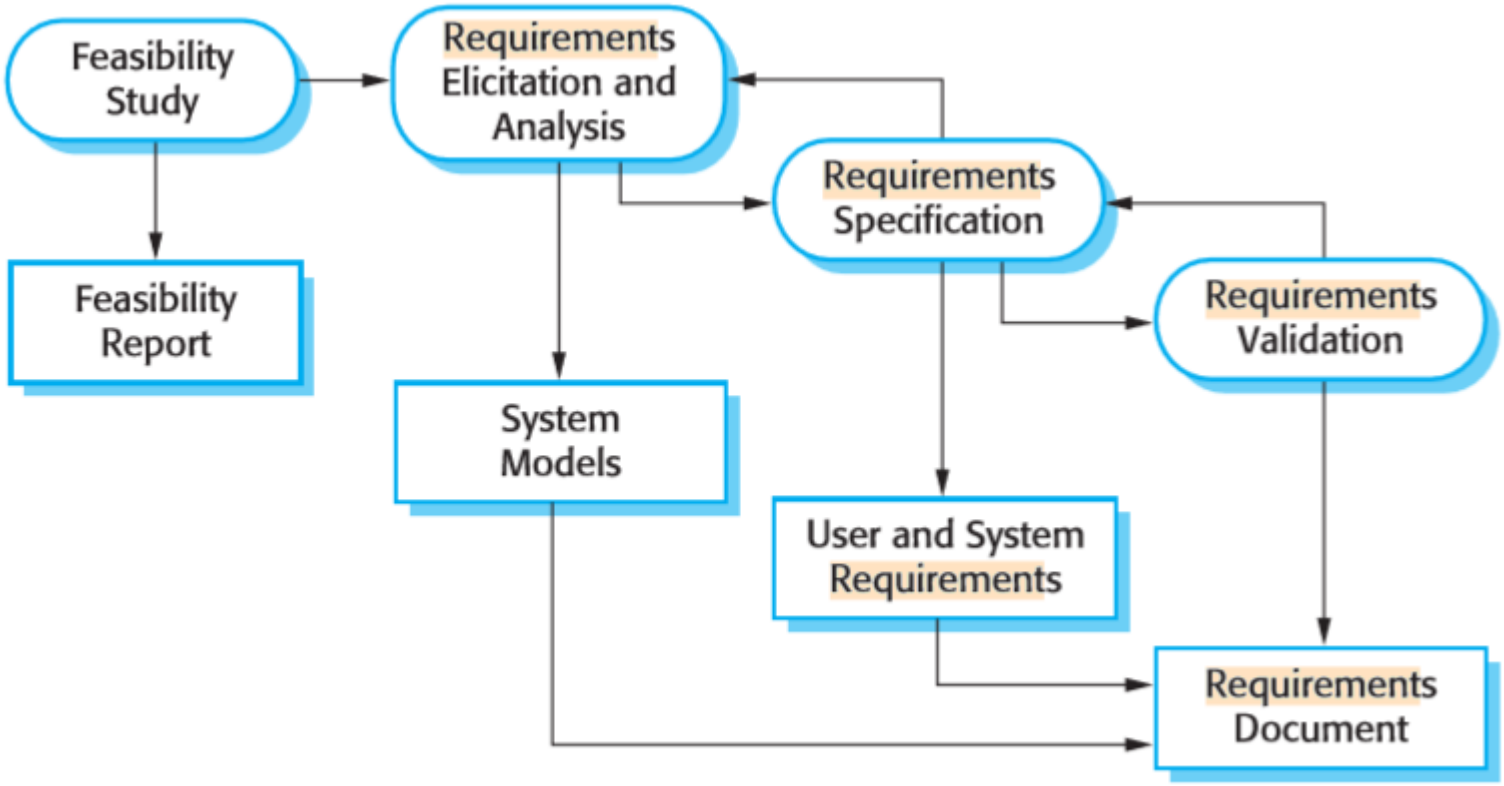
- The insulin pump bases its computations of the insulin requirement on the rate of change of blood sugar levels.

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

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

Requirements Engineering Process

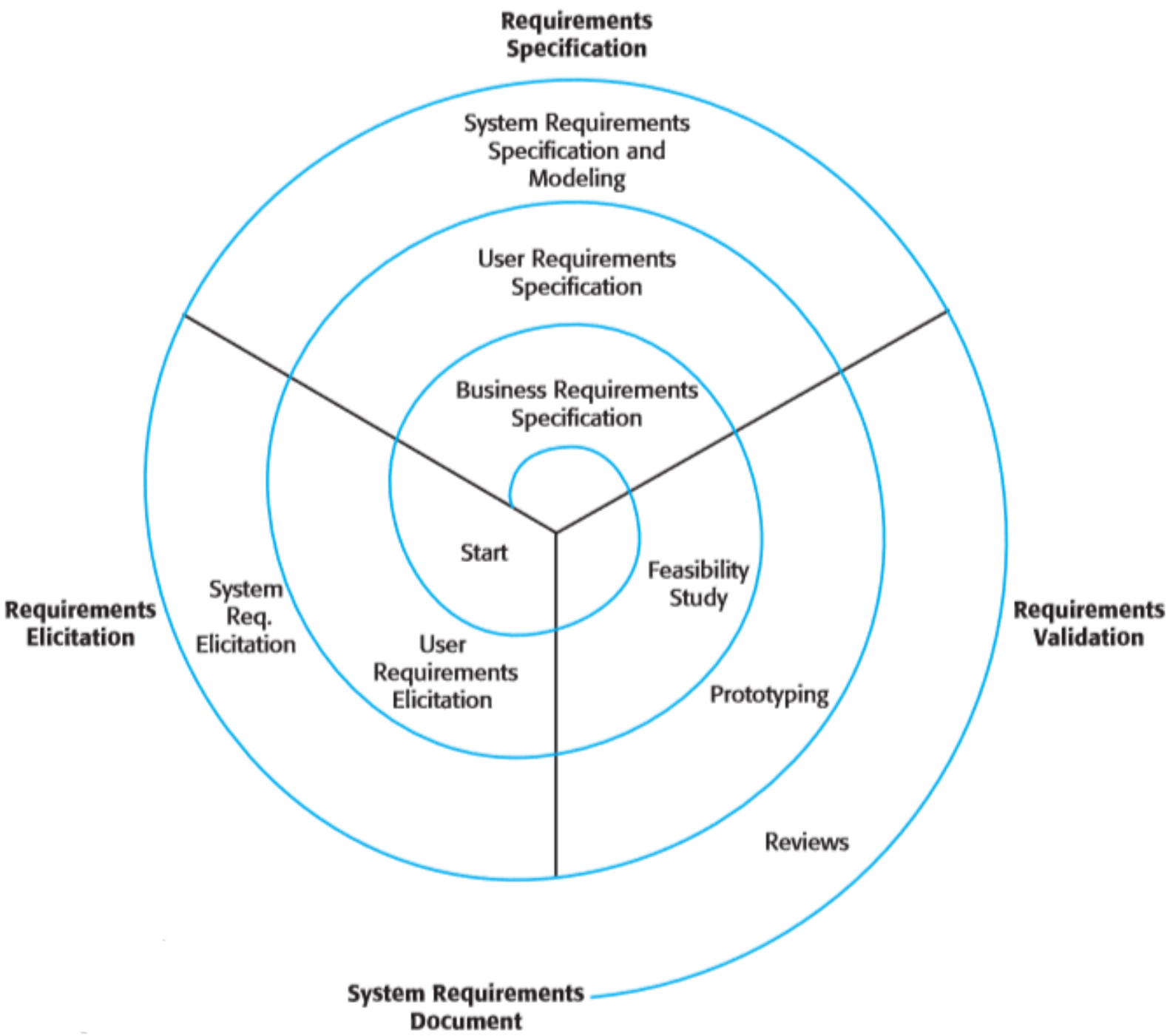
- It includes four high level activities



- **Each loop of the spiral is called a Phase of the Requirement Engineering process.**

- The Radius of the spiral at any point represents the expenses(cost) of the project so far, and

- the angular dimension represents the progress made so far in the current phase



Requirement Validation

- Requirements validation is the process of checking that requirements actually define the system that the customer really wants.

- The cost of fixing a requirements problem by making a system change is usually much greater than repairing design or coding errors

Types of checks : requirements validation process

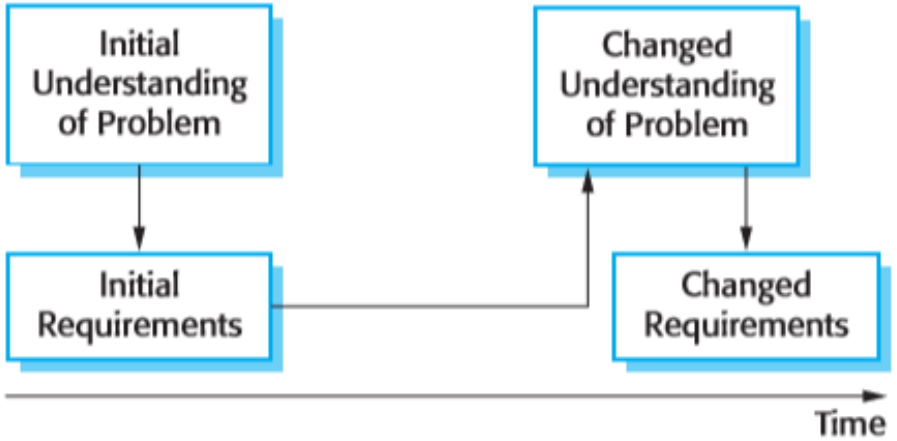
- Validity checks: identify additional or different functions that are required
- Consistency checks: Requirements in the document should not conflict
- Completeness checks: It should define all functions and the constraints intended by the system user
- Realism checks: ensure that they can actually be implemented. It also take account of the budget and schedule for the system development.
- Verifiability To reduce the potential for dispute between customer and contractor, system requirements should always be written so that they are verifiable.

Requirement Management

- Requirements management planning
- Requirements change management

Requirement Management

- The requirements for large software systems (problems that cannot be completely defined) are always changing.
  - Because the problem cannot be fully defined, the software requirements are bound to be incomplete.
  - The stakeholders' understanding of the problem is constantly changing.



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

There are several reasons why change is inevitable

- The **business and technical environment** of the system always **changes after installation**.
- **New hardware** may be introduced, it may be necessary to interface the system with other systems,
- **business priorities may change**
- **New legislation and regulations** may be introduced that the system must necessarily abide by.

There are several reasons why change is inevitable

- The people who pay for a system and the users of that system are rarely the same people.
  - **impose requirements** because of organizational 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.

There are several reasons why change is inevitable

- **Large systems usually have a diverse user community**
  - Many users have different requirements and priorities that may be conflicting or contradictory.
  - The **final system requirements are inevitably a compromise** between them and,
  - **with experience, it is often discovered that the balance of support given to different users has to be changed**

Requirement Management

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

- You need to keep track of individual requirements to maintain links between dependent requirements so that the impact of requirements changes can be assessed.

Requirement Management Planning

During requirement management stage, you have to decide on..

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

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

3. **Traceability policies**
   - It define the relationships between each requirement and between the requirements and the system design that should be recorded.
   - It should also define how these records should be maintained.

4. **Tool support**
   - Requirements management involves the processing of large amounts of information about the requirements.
   - Tools : specialist requirements management systems, spreadsheets or simple database systems.

Need of Requirement Management Tools

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 is simplified if active tool support is available.

3. **Traceability management**
   - Tool support for traceability allows related requirements to be discovered.
   - Some tools are available which use natural language processing techniques to help discover possible relationships between requirements.

Requirement Change Management

- **It should be applied to all proposed changes to a system's requirements after the requirements document has been approved.**

- All change proposals are treated consistently and changes to the requirements document are made in a controlled way.

Principal stages to a change management process

1. **Problem analysis and change specification**
   - The process starts with an identified requirements problem or, sometimes, with a specific change proposal.
   - During this stage, the problem or the change proposal is analyzed to check that it is valid.
   - This analysis is fed back to
     - the change requestor who may respond with a more specific requirements change proposal, or
     - decide to withdraw the request

2. Change analysis and costing

- The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements.
- The cost of making the change is estimated both in terms of modifications to the requirements document and, if appropriate, to the system design and implementation.
- Once this analysis is completed, a decision is made whether or not to proceed with the requirements change.

3. Change implementation

- The **requirements document** and, where necessary, the **system design** and implementation, **are modified**.

- You should **organize the requirements document** so that **you can make changes to it without extensive rewriting or reorganization**.

- As with programs, changeability in documents is achieved by minimizing external references and making the document sections as modular as possible.
  - Thus, individual sections can be changed and replaced without affecting other parts of the document.