# UNIT – 2

1. **Requirements Engineering**

   Requirements Engineering is the process of figuring out what a customer needs from a system and defining the rules and limits under which the system must work.

2. **What is a Software Requirement?**

   Software requirements explain what a system should do and the rules it must follow. It can be:

   ✓ High-level – A general idea of what the system should accomplish.

   ✓ Detailed – A specific, technical breakdown of how the system should function.

   **Why Requirements Can Vary:**

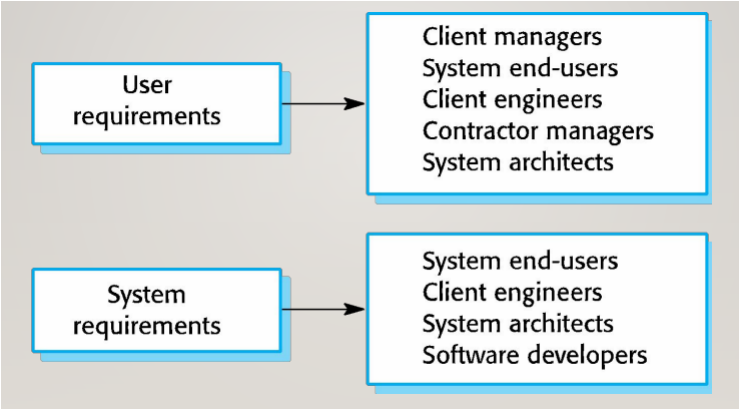
   - If the requirement is used in a contract bid, it is often kept broad and open to interpretation so different companies can propose solutions.
   - If the requirement is part of a contract, it must be clear and precise to avoid confusion.

   **Importance:**

   - Guides development process.
   - Ensures alignment with user needs.
   - Reduces risks of project failure.

3. **TYPES OF REQUIREMENTS.**

   **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.



| No. | Users Requirements | System Requirements |
|---|---|---|
| 1 | Written for Customers. | Written for Implementation Team. |
| 2 | In Natural language. | In Technical Language. |
| 3 | Describe the services & features provide by system. | Describe the detail description of services, features & complete operations of system. |
| 4 | May include diagrams & tables. | May include system models. |
| 5 | Understandable by system users who don't have technical knowledge. | Understandable by implementation team wh have technical knowledge. |
| 6 | For Client Managers, System Users, Contractor Managers, System Architects | For System Architects, Software Developers Client Engineers, System Users |

## 4. FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS.

**Functional Requirements**

a) These define the services and actions that a system should perform. They depend on:

- The type of software
- The users
- The system environment

b) Functional requirements describe how the system reacts to inputs and different conditions.

c) They can also state what the system should NOT do.

**Example:** In an online banking system:
Users should be able to transfer money between accounts.
The system should generate a transaction receipt.
The system should not allow money transfer without authentication.

**Non-Functional Requirements**

a) These describe the qualities, constraints, and limitations of the system, such as:

- Performance (e.g., system must load within 2 seconds)
- Security (e.g., only authorized users can access the system)
- Usability (e.g., system must be easy to use for beginners)
- Compatibility (e.g., system must work on Windows, Mac, and Linux)

b) These requirements often apply to the entire system rather than a single feature.

**Example**: In an e-commerce website:
The website should load in under 3 seconds.
The website should encrypt customer payment information for security.
The website should be accessible on both mobile and desktop devices.

**Goals vs. Verifiable Requirements**

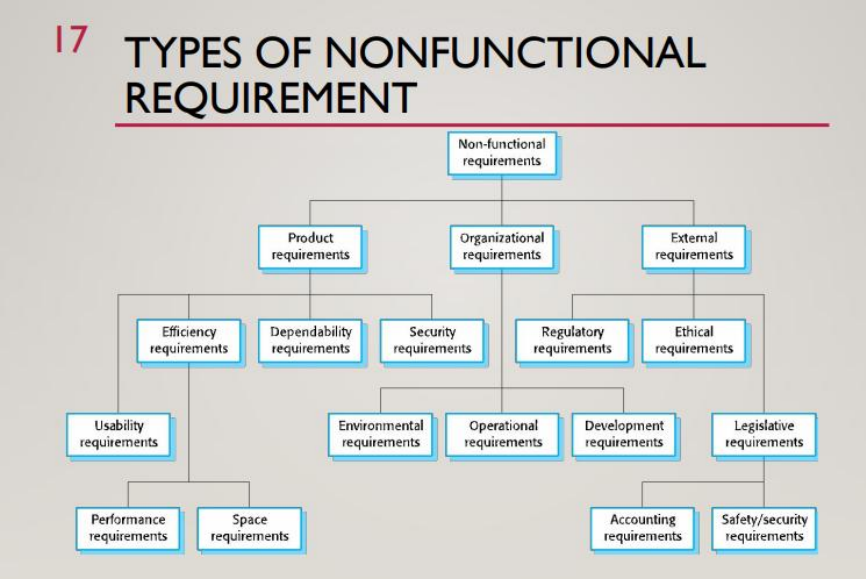
- Goal: A general wish or intention (hard to measure).
- Verifiable Requirement: A specific rule that can be tested.

**Example**

- Goal: "The system should be easy to use." (Not measurable)
- Verifiable Requirement: "A new user should be able to complete registration in under 3 minutes." (Can be tested)

**Domain Requirements** – Special rules based on the system's industry or field.

- 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.




**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.

| Feature | Functional Requirements | Non-Functional Requirements |
|---|---|---|
| **Definition** | Describes **what** the system should do. | Describes **how** the system should perform. |
| **Focus** | Features, services, and functionalities. | Performance, security, usability, etc. |
| **Scope** | Applies to specific system functions. | Applies to the whole system. |
| **Example Questions** | - What actions should the system perform? | - How fast should the system respond? |
| | - What should happen when a user logs in? | - How secure should user data be? |
| **Examples** | - User can log in using email and password. | - System should respond within 2 seconds. |
| | - System should allow online payments. | - System must support 1,000 concurrent users. |
| **Testing Method** | Validated through **functional testing.** | Validated through **performance, security, or usability testing.** |

## 5. What is Requirements Specification?

Requirements specification is the process of documenting user and system requirements in a clear and structured manner.

User requirements: Written in simple language so non-technical users can understand.

System requirements: More detailed and may include technical aspects for developers.

Requirements should be complete and clear, especially if they are part of a contract.

## 6. Relationship Between Requirements & Design

- Requirements describe WHAT the system should do.
- Design describes HOW the system will do it.
- In reality, requirements and design overlap due to:
    - System architecture decisions affecting requirements.

- External system integration leading to design constraints.
- Domain requirements (e.g., regulatory rules) influencing system structure.

**7. Ways to Write System Requirements Specification.**

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

**Natural Language Specification**

- Most commonly used because it is easy to understand.
- Include diagrams and tables to clarify complex requirements.

Guidelines for Writing Clear Requirements:

✓ Use a standard format for all requirements.
✓ Use "shall" for mandatory requirements and "should" for optional ones.
✓ Highlight important terms using text formatting.
✓ Avoid technical jargon for user requirements.
✓ Provide a rationale explaining why a requirement is needed.

Common Problems with Natural Language:

- Lack of clarity: Hard to be precise while keeping it readable.
- Mixing up functional & non-functional requirements.
- Merging multiple requirements into one statement.

**Example: Insulin Pump Software System Requirements**

Functional Requirement:

- *"The system shall measure blood sugar and deliver insulin, if required, every 10 minutes."*

- *(This ensures sugar levels remain stable.)*

Non-Functional Requirement:

- *"The system shall run a self-test every minute to check for hardware and software errors."*
- *(This improves system reliability.)*

**Structured Specifications**

- A structured approach to writing requirements using predefined formats.
- Works well for embedded systems (e.g., medical devices, industrial control systems).
- Not ideal for business applications (e.g., e-commerce systems).

**Example: REQUIREMENT FOR AN INSULIN PUMP**

| *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. |

**Form-Based Specifications (Template Approach)**

Each requirement follows a structured format, including:

✓ Function definition

✓ Input & output details

✓ Processing logic

✓ Pre and post conditions

✓ Side effects

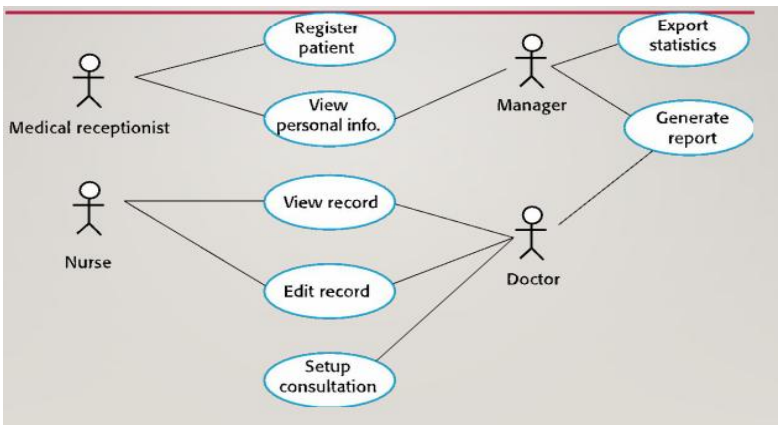
**Tabular Specification**

- Helps define multiple conditions and outcomes clearly.
- Useful for decision-making scenarios in software logic.

| 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)<br>If rounded result = 0 then CompDose = MinimumDose |

**Use Cases**

- Use cases describe how users interact with the system.
- Represented using UML sequence diagrams that shows step-by-step interactions between different system components.
- Helps developers understand how data flows through the system.
- Includes:
  - ✓ Actors (users or external systems interacting with the system).
  - ✓ Scenarios (step-by-step process of interaction).
  - ✓ Alternative flows (what happens if something goes wrong).



## 8. The Software Requirements Document

The Software Requirements Document is the official statement of what is expected from the software developers.

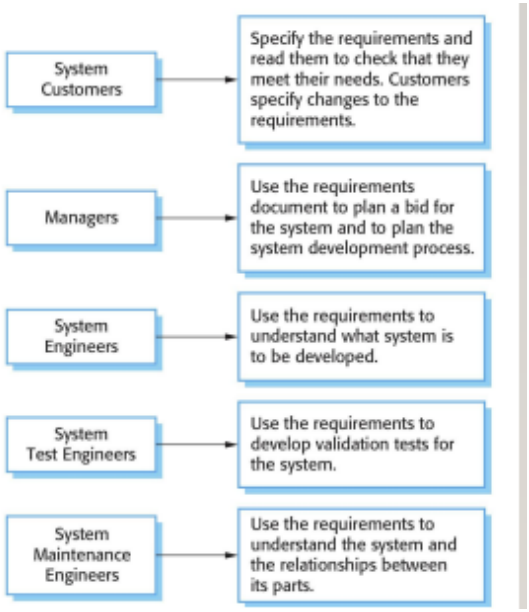
Key Points:

✓ It defines WHAT the system should do, not HOW it should do it.

✓ Includes both user requirements and system requirements.

✓ Serves as a contract between customers and developers.

✓ Helps ensure that all stakeholders understand the system's functionality.




## 9. Requirements Document Variability

The content of the SRS document depends on:

- Type of System – A simple mobile app will have fewer requirements than a large banking system.
- Development Approach – Agile projects may have less detailed documentation, while large government projects need very detailed documents.
- Industry Standards – Some projects follow IEEE SRS standards, especially for large systems (e.g., aerospace, medical software).

## 10. Structure of Requirement Doc.

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



## 11. REQUIREMENTS ENGINEERING PROCESSES

Requirements Engineering (RE) is a continuous and iterative process used to gather, analyse, validate, and manage software requirements. The specific steps may vary based on project type, people involved, and organization.

However, all RE processes include four key activities:

a) Requirements Elicitation – Gathering requirements from customers, users, and stakeholders.

b) Requirements Analysis – Understanding and refining the gathered requirements, identifying conflicts or missing details.

c) Requirements Validation – Checking if the requirements are correct, complete, and meet user needs.

d) Requirements Management – Keeping track of changes and updates to requirements throughout the project.

## 12. Requirements Elicitation and Analysis.

Requirements Elicitation and Analysis is also called Requirements Discovery. It involves working with stakeholders (users, managers, engineers, etc.) to gather information about what the system should do and what constraints exist.

**Steps/Process in Requirements Elicitation and Analysis:**

**a) Requirements Discovery** – This is the process of interacting with stakeholders of the system to discover their requirements.

**b) Requirements Classification & Organization** – This activity takes the unstructured collection of requirements, groups related requirements, and organizes them into coherent clusters.

**c) Requirements Prioritization & Negotiation –** When multiple stake holders are involved, requirements will conflict. This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiation.

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

**Problems in Requirements Elicitation:**

- Stakeholders don't always know what they want.
- Stakeholders may describe things in their own technical language.
- Different stakeholders may have conflicting needs.
- Organizational and political factors may affect system requirements.
- Requirements may change during the analysis process.

**13. Requirements Discovery Methods.**

**Interviews**

A common method where stakeholders are asked about their needs and expectations.

Can be:

- Closed Interviews – Predefined set of structured questions.
- Open Interviews – Free discussion to explore various issues.

Effective Interviewing Tips:

- Be open-minded and avoid assumptions.
- Use prompts (questions, prototypes) to encourage discussion.

Problems with Interviews:

- Stakeholders may use technical jargon that is hard to understand.
- Developers might misinterpret requirements due to lack of domain knowledge.

**Ethnography (Observing Users at Work)**

- A researcher spends time watching how people work instead of asking them.
- Helps identify hidden requirements that users don't realize are important.
- Social and organizational factors can be observed directly.

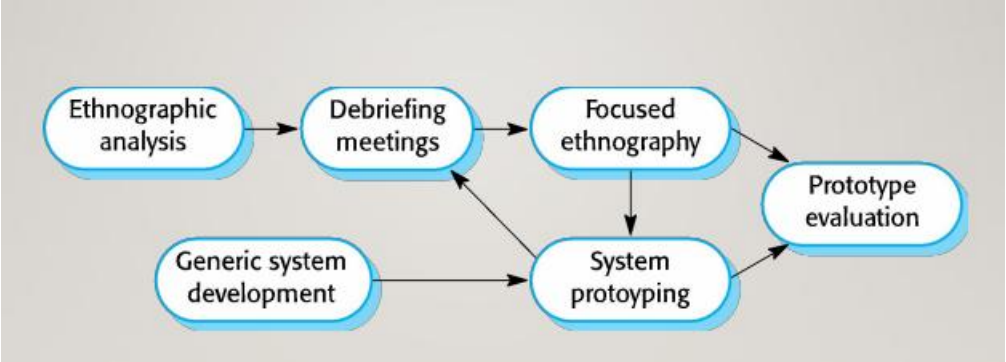
Limitations of Ethnography:

- Focuses on existing processes, so it may not suggest new features.
- Some historical practices might no longer be relevant.

**Focused Ethnography**

- Used in complex environments like air traffic control.
- Combines ethnography with prototyping – unanswered questions from prototypes guide further observation.

Problem: May study old practices that are outdated and no longer needed.




**Stories and Scenarios**

**User stories** describe real-world situations where the system will be used. **Scenarios** provide structured stories covering:

- Starting conditions
- Normal flow of events
- Errors or problems that might occur
- Other activities happening at the same time
- Final system state after the scenario ends

**Example: Photo Sharing in the Classroom (iLearn)**

A primary school teacher, Jack, wants his students to research the history of fishing in their village. As part of the project, students need to upload and comment on photos related to fishing.

---

📌 How Jack Sets Up the System

- Students use an iLearn wiki to collect stories and research.
- They access SCRAN, a history resource site, for newspaper archives and photos.
- Jack needs a photo-sharing platform where students can upload and discuss photos.
- He asks for recommendations from other teachers.
- Two teachers suggest using KidsTakePics, a photo-sharing site with moderation features.
- Since KidsTakePics is not directly connected to iLearn, Jack creates a teacher and class account.
- He adds KidsTakePics to the iLearn services, so students can log in and upload photos easily.

📌 Uploading Photos on iLearn

Normal Upload Process:

- User selects photos to upload from their tablet or laptop.
- They choose a project name and can add keywords for better organization.
- The uploaded photo is renamed automatically (combining the user's name and original filename).
- After uploading, the system:
  - ✓ Sends an email to the moderator for approval.
  - ✓ Displays a confirmation message to the user.

---

📌 What Can Go Wrong & Solutions

🔹 No Moderator Assigned

- If a project has no moderator, an email is sent to the school admin to assign one.
- The user is notified about potential delays.

🔹 Duplicate Photo Names

- If a photo with the same name was already uploaded, the user can:
  - ✅ Overwrite the existing file.
  - ✅ Rename the new file automatically.
  - ✅ Cancel the upload.

🔹 Other Activities During Upload

- The moderator may already be online and can approve photos in real-time.

🔹 System Status After Upload

- The user stays logged in.
- Uploaded photos are marked as 'awaiting moderation'.
- Photos are only visible to the moderator and the uploader until approved.

**14. REQUIREMENTS VALIDATION.**

- Requirements Validation ensures that the defined system matches what the customer actually wants.
- Fixing errors in requirements is very expensive, especially if found after system delivery (up to 100 times the cost of fixing a coding error).
- Validating requirements early saves time, cost, and effort.

**Key Checks in Requirements Validation**

✅ Validity – Does the system provide the right functions for the customer's needs?
✅ Consistency – Are there conflicting requirements?
✅ Completeness – Are all required functions included?

✅ Realism – Can the requirements be implemented within budget and technology limits?
✅ Verifiability – Can the requirements be tested and checked?

**Requirements Validation Techniques**

- 🔹 Requirements Reviews – Manual checking of requirements by experts.
- 🔹 Prototyping – Creating an executable model to verify requirements.
- 🔹 Test-Case Generation – Developing test cases to ensure the requirements can be tested.

**Requirements Reviews**

✓ Conducted regularly while requirements are being defined.
✓ Both the client and developers should participate.
✓ Can be formal (document-based) or informal (discussions).
✓ Good communication helps identify and fix problems early.

**Review Checklist**

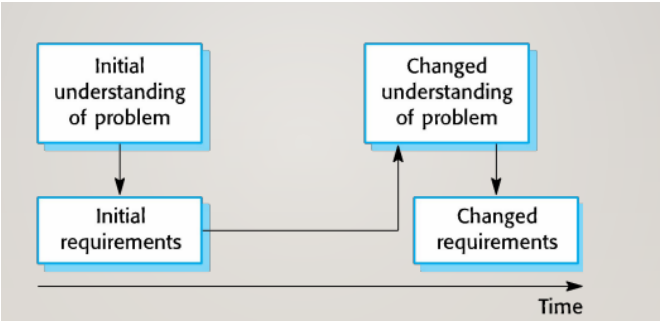
- 🔹 Verifiability – Can the requirement be tested?
- 🔹 Comprehensibility – Is the requirement clear and understandable?
- 🔹 Traceability – Can we track where the requirement came from?
- 🔹 Adaptability – Can the requirement be changed without affecting others too much?

## 15. REQUIREMENTS CHANGE & MANAGEMENT.

Requirements change over time due to new business needs, technology updates, user feedback, or legal regulations. Managing these changes is crucial to keep the system useful and up-to-date.

**Why Do Requirements Change?**

✓ Technology Updates – New hardware or software may need to be supported.
✓ Business Priorities Change – New policies or goals require changes in system support.
✓ Legal & Regulatory Changes – The system must comply with new laws.
✓ Different Stakeholder Needs – Customers (who pay) and end-users (who use the system) may have conflicting requirements.
✓ Large Systems Have Many Users – Different users have different priorities, and over time, the system may need to rebalance support for different user groups.




**What is Requirements Management?**

Requirements Management is the process of handling changes in requirements during development and after the system is in use.

✓ Tracks each requirement and its relationships with other requirements.
✓ Assesses the impact of changes before they are implemented.
✓ Uses a formal process to approve or reject requirement changes.

**Steps in Requirements Management**

- Requirements Identification – Each requirement is given a unique ID for tracking.
- Change Management Process – A structured way to analyse and approve changes.
- Traceability Policies – Defines how requirements link to each other and to the system design.
- Tool Support – Uses spreadsheets, databases, or specialized software to manage changes.

**Requirements Change Management Process**

**Step 1: Problem Analysis & Change Proposal**

- A proposed change is analysed to check if it is valid.
- Feedback is given to the requester, who may revise or withdraw the request.

**Step 2: Change Analysis & Costing**

- The impact of the change is assessed using traceability links.
- A decision is made whether to accept or reject the change.

**Step 3: Change Implementation**

- The requirements document, system design, and code are updated.
- The document should be structured for easy updates.

## 16. COCOMO Model

Boehm proposed COCOMO (Constructive Cost Estimation Model) in 1981.

COCOMO predicts the efforts and schedule of a software product based on the size of the software.

**Organic**

• A development project can be treated of the organic type, if the project deals with developing a well-understood.

• Application program, the size of the development team is reasonably small, and the team members are experienced in developing similar methods of projects.

• Examples of this type of projects are simple business systems, simple inventory management systems, and data processing system.

**Semidetached**

• A development project can be treated with semidetached type if the development consists of a mixture of experienced and inexperienced staff.

• Team members may have finite experience in related systems but may be unfamiliar with some aspects of the order being developed.

• Example of Semidetached system includes developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.

**Embedded**

• A development project is treated to be of an embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational method exist.

• For Example: ATM, Air Traffic control.