# Unit 4: Requirements Engineering (3 Hrs.)

## Concept of User and System Requirements

Requirements form the foundation of any software system, defining what it should do and how it should behave. They bridge the gap between stakeholder needs and technical implementation. Requirements are typically categorized into two main types: **user requirements** and **system requirements**.

### 1. User Requirements

User requirements describe **what** the system should do from the perspective of end-users and stakeholders. They are high-level, often written in natural language (with supporting diagrams), and focus on business needs rather than technical details.

**Characteristics**

- **Abstract & Goal-Oriented**: Describe functionality in broad terms (e.g., "The system shall generate monthly drug cost reports").
- **Audience**: Business managers, end-users, and non-technical stakeholders.
- **Flexible**: May evolve during early discussions before being refined into system requirements.

**Example (Mentcare System)**

**User Requirement:**

*"The Mentcare system shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month."*

**Purpose**

- Helps stakeholders agree on system objectives.
- Guides developers in deriving detailed system requirements.

### 2. System Requirements

System requirements specify **how** the system will fulfill user requirements. They are detailed, precise, and often structured for developers, testers, and architects.

**Characteristics**

- **Technical & Specific**: Define exact behavior, constraints, and interfaces.
- **Audience**: Developers, system architects, and QA teams.
- **Formal**: May include data formats, algorithms, and performance criteria.

**Example (Mentcare System)**

**System Requirements:**

1. *"On the last working day of each month, generate a drug summary report by clinic."*
2. *"Restrict report access to users on the management ACL."*
3. *"Separate reports by drug dose units (e.g., 10mg vs. 20mg)."*

**Purpose**

- Serves as a **contract** between clients and developers.
- Provides a **blueprint** for implementation and testing.

**Key Differences**

| Aspect | User Requirements | System Requirements |
|---|---|---|
| **Level of Detail** | High-level, abstract | Detailed, technical |
| **Audience** | Business stakeholders, end-users | Developers, testers, architects |
| **Focus** | What the system should do | How the system will do it |
| **Format** | Natural language + diagrams | Structured specifications (e.g., SRS) |

**Why Both Are Necessary**

1. **For Stakeholders**
    - User requirements ensure the system aligns with business goals.
    - Non-technical readers (e.g., clinic managers) need clarity without implementation details.
2. **For Developers**
    - System requirements eliminate ambiguity for accurate implementation.
    - Example: A "monthly report" (user req.) becomes "generate CSV every 1st day at 2 AM" (system req.).
3. **For Compliance**

- Regulators (e.g., medical ethics boards) may audit system requirements for safety.

## Functional and Non-functional Requirements

Requirements in software development are categorized into two fundamental types: **functional requirements** (what the system does) and **non-functional requirements** (how well the system performs). Together, they define the complete behavior and quality of a software system.

### 1. Functional Requirements: Defining System Capabilities

Functional requirements specify the **actions, services, and behaviors** the system must perform. They describe **what** the system should do in response to specific inputs or conditions.

#### Characteristics

- **Action-Oriented**: Define specific operations (e.g., calculations, data processing).
- **Input-Output Based**: Specify how the system reacts to user actions or external events.
- **Testable**: Can be verified through functional testing (e.g., unit tests, user acceptance tests).

#### Examples (Mentcare System)

1. *"A user shall be able to search appointment lists for all clinics."*
2. *"The system shall generate daily patient attendance lists per clinic."*
3. "Staff must log in using an 8-digit employee ID."

#### Challenges

- **Ambiguity**: Vague requirements lead to misinterpretations.
    - *Example*: "Search appointments" could mean either:
        - (User expectation) Search all clinics at once.
        - (Developer interpretation) Search one clinic at a time.
- **Incompleteness**: Missing edge cases (e.g., handling no-show patients)

### 2. Non-Functional Requirements: Defining System Quality

Non-functional requirements (NFRs) specify **constraints and quality standards** for the system. They describe **how well** the system performs its functions.

#### Categories of NFRs

NFRs can be classified into three broad types:

### A. Product Requirements

Define runtime behavior and system attributes:

- **Performance**: *"Downtime shall not exceed 5 seconds during working hours."*
- **Reliability**: *"System availability must be 99.9%."*
- **Usability**: *"Medical staff shall master all functions within 2 hours of training."*

### B. Organizational Requirements

Derived from company policies or processes:

- **Compliance**: *"Users must authenticate via health authority ID cards."*
- **Development Standards**: *"Use Python 3.10 for backend services."*

### C. External Requirements

Imposed by laws, regulations, or ethics:

- **Privacy**: "Conform to HStan-03-2006-priv for patient data."
- **Safety**: *"Comply with FDA regulations for medical devices."*

### Importance of NFRs

- **Critical for System Success**:
    - A slow but functional system frustrates users.
    - A fast but insecure system risks data breaches.
- **Influence Architecture**:
    - High-performance needs may require distributed systems.
    - Strict security needs may mandate encryption protocols.

## Requirements Engineering Process

Requirements Engineering (RE) is a systematic process for discovering, documenting, and validating the needs and constraints for a software system.

There are three main activities in the requirements engineering process:

1. **Requirements Elicitation**

   Gathering and understanding stakeholder needs through collaboration through various techniques like interviews, surveys, observations and workshops. These help to understand the system to be specified.

2. **Requirements Analysis and Specification**

   The elicited needs are converted into structured documentation. Two types of requirements may be included in this document. User requirements are abstract statements of the system requirements for the customer and end-user of the system; system requirements are a more detailed description of the functionality to be provided.

3. **Requirements Validation**

   The main goal of requirements validation is to ensure requirements are correct, complete, and aligned with stakeholder needs. During this process, errors in the requirements document are inevitably discovered. It must then be modified to correct these problems.

## Requirements Elicitation

Requirements elicitation is a critical phase in software engineering aimed at understanding stakeholders' needs and how a new system can support their work. The process involves interacting with stakeholders to gather information about the application domain, work activities, desired system features, performance expectations, and constraints. However, eliciting requirements is challenging due to stakeholders' varying levels of articulation, domain-specific jargon, conflicting priorities, and dynamic business environments.

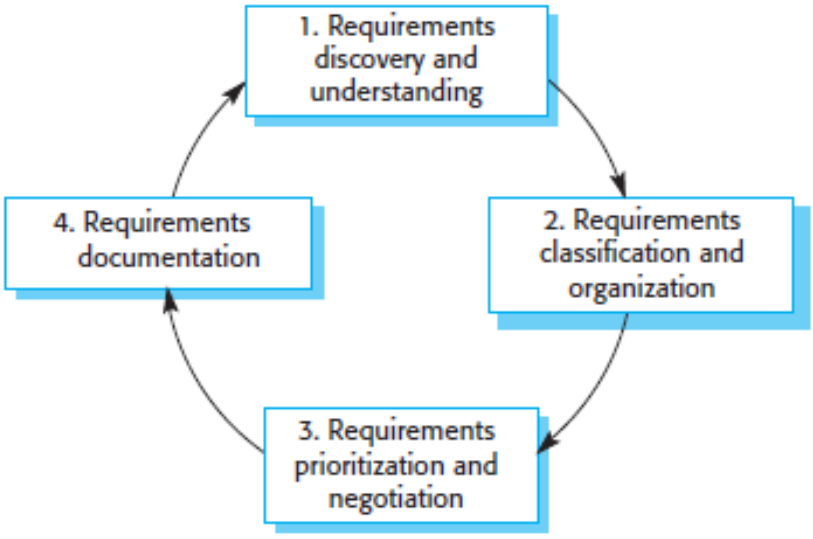


Figure 4.1: The requirements elicitation and analysis process

**Process Model of Requirements Elicitation and Analysis**

The process is iterative and consists of four key activities, as illustrated in figure 4.1:

1. **Requirements Discovery and Understanding**
    - Involves direct interaction with stakeholders to uncover their needs.
    - Includes gathering domain requirements from stakeholders and existing documentation.
    - Techniques: Interviews, observation, and document analysis.
2. **Requirements Classification and Organization**
    - Structures the collected requirements into coherent groups.
    - Methods: Viewpoints (grouping by stakeholder perspectives)
3. **Requirements Prioritization and Negotiation**
    - Addresses conflicts between stakeholders' requirements.
    - Involves negotiation and compromise to prioritize essential features.
    - Regular stakeholder meetings help ensure fairness and consensus.
4. **Requirements Documentation**
    - Formalizes the requirements in a clear, accessible format (e.g., text, diagrams).
    - May produce an early draft of the software requirements document.
    - Ensures stakeholders can review and provide feedback.

The process is cyclical, with continuous refinement until a stable set of requirements is achieved.

**Requirements Elicitation Techniques**

To gather accurate and comprehensive requirements, a mix of techniques is used:

**1. Interviewing**

- **Types:**
    - **Closed Interviews:** Structured with predefined questions.
    - **Open Interviews:** Flexible discussions to explore stakeholder needs.
- **Challenges:**
    - Stakeholders may struggle to articulate needs clearly.
    - Domain-specific jargon can lead to misunderstandings.
- **Best Practices:**
    - Use open-ended questions to encourage discussion.
    - Supplement interviews with prototypes or examples to aid visualization.

**2. Ethnography (Observation)**

- Involves observing stakeholders in their work environment to uncover implicit requirements.
- **Strengths:**
  - Reveals real-world workflows (vs. formal processes).
  - Identifies social and organizational factors affecting system use.

**3. Stories and Scenarios**

- **Stories:** High-level narratives describing system use (e.g., a teacher using an e-learning platform).
- **Scenarios:** Detailed, structured descriptions of specific tasks (inputs, outputs, steps).
- **Purpose:**
  - Facilitate discussions and refine requirements.
  - Help stakeholders relate to real-world usage.

## Requirements Specification

Requirements specification is the process of formally documenting user and system requirements in a structured manner. The goal is to produce a clear, unambiguous, and comprehensive requirements document that serves as a foundation for system design, development, and validation. However, achieving perfection is challenging due to differing stakeholder interpretations and inherent conflicts in requirements.

**Methods for Specifying Requirements**

**A. Natural Language**

The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.

- **Pros:** Intuitive, universally understood.
- **Cons:** Prone to ambiguity.
- **Guidelines for Clarity:**
  1. **Standardize format** (e.g., one sentence per requirement).
  2. Use **"shall"** for mandatory requirements, **"should"** for desirable ones.

3. Highlight **key terms** (bold/italic).
4. Avoid jargon; include **rationale** for each requirement.

**B. Structured Specifications**

- **Approach:** Use **templates or forms** to organize requirements systematically.
- **Example Fields:**
  - Function description, inputs/outputs, preconditions, postconditions.
- **Advantages:** Reduces ambiguity; improves traceability.
- **Example:**
  - Function: Calculate Insulin Dose
  - Inputs: Current blood sugar level (mg/dL)
  - Outputs: Insulin dose (units)
  - Precondition: Patient's blood sugar is above safe threshold.
  - Postcondition: Dose is logged in the patient's record.

**C. Use Cases**

- **Purpose:** Model **interactions** between users (actors) and the system.
- **Components:**
  - **Actors:** Roles (e.g., doctor, nurse).
  - **Use Cases:** Named interactions (e.g., "Register Patient").

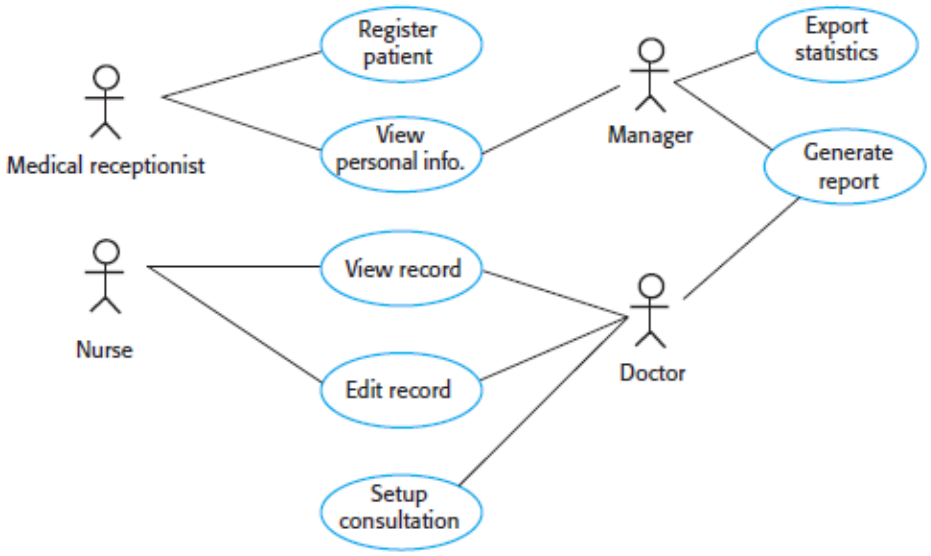


Figure 4.2: Use Case for Mentcare System

- **Strengths:** Visual (UML diagrams) + textual descriptions.
- **Limitations:** Stakeholders may find them too technical for early discussions.
- **Example:**

  **Use Case:** View Patient Record
  **Actor:** Doctor
  **Description:** The doctor accesses the patient's medical history, including lab results and prescriptions.

**D. Tabular and Graphical Models**

- **Tables:** Clarify complex logic (e.g., decision rules for medical dosing).
- **Diagrams:** Flowcharts, state machines, or UML models (e.g., sequence diagrams).

---

**The Software Requirements Document (SRS)**

**Purpose**

- Official **contract** between stakeholders and developers.
- Guides **design, testing, and maintenance**.

**Users**

- **Customers:** Verify requirements meet their needs.
- **Managers:** Plan budgets and timelines.
- **Engineers:** Develop and test the system.
- **Maintenance Teams:** Understand works on system evolution.

**Content**

1. **User Requirements:** High-level needs.
2. **System Requirements:** Detailed specifications.
3. **Non-Functional Requirements:** Security, performance, etc.
4. **Appendices:** Glossary, diagrams, use cases.

## Requirements Validation

Requirements validation is the process of ensuring that the documented requirements accurately define the system that the customer truly needs. This process is closely related to requirements elicitation and analysis, as it involves identifying and resolving problems or inconsistencies within the requirements.

**Importance of Requirements Validation**

Validating requirements is crucial because errors found at this stage are significantly less costly to fix than if they are discovered during system development or after deployment. Any change in the requirements often triggers corresponding changes in design, implementation, and testing, leading to additional time and cost.

**Key Validation Checks**

To ensure the quality and accuracy of requirements, several types of checks should be performed:

1. **Validity Checks**: Confirm that the requirements genuinely reflect the users' needs, accounting for any changes since the initial elicitation.
2. **Consistency Checks**: Ensure there are no conflicting or contradictory requirements in the document.

3. **Completeness Checks**: Verify that all intended functions and constraints are covered in the requirements.
4. **Realism Checks**: Assess whether the requirements can be feasibly implemented within the given budget, schedule, and technological constraints.
5. **Verifiability Checks**: Ensure each requirement is stated in a way that allows for clear testing and verification.

**Requirements Validation Techniques**

Several techniques are used to validate requirements effectively:

- **Requirements Reviews**: A team systematically examines the requirements for errors, inconsistencies, and omissions.
- **Prototyping**: A working model of the system is developed and tested with users to validate that it meets their needs and expectations.
- **Test-case Generation**: Tests are developed based on the requirements to ensure they are testable and to uncover any ambiguities or implementation challenges early.

## Requirements Change

Requirement change refers to any modification made to the requirements of a system after they have been initially defined and documented. These changes are common and often unavoidable in software and systems development due to evolving user needs, business goals, market conditions, or technical constraints.

**Reasons for Requirement Change**

1. **Evolving User Needs**: As users better understand the system or as their work processes change, they may request modifications to the requirements.
2. **Business Environment Changes**: New laws, regulations, competitor actions, or strategic shifts may require changes to system functionality or constraints.
3. **Technology Advancements**: Emerging technologies or platforms may lead to rethinking existing requirements to leverage new capabilities.

4. **Clarification of Requirements**: Initial requirements may be vague, ambiguous, or misunderstood. As development progresses, these issues become clear and require correction.
5. **Errors and Omissions**: Issues not identified during requirements validation—such as incomplete or inconsistent requirements—may necessitate updates during later phases.

**Impact of Requirement Changes**

Requirement changes can significantly affect the project:

- **Design and Implementation**: Changes in requirements often require corresponding updates to the system's design and codebase.
- **Testing and Validation**: Modified requirements must be retested, potentially adding to the testing workload and complexity.
- **Project Schedule and Budget**: Unplanned changes can lead to delays and increased costs, especially if they are introduced late in the development cycle.
- **Team Coordination**: Changes must be communicated clearly to all stakeholders, including developers, testers, and users, to avoid confusion and ensure consistent understanding.

**Requirement Change Management**

Effective change management involves:

- **Change Control Process**: A formal process for submitting, evaluating, approving, and tracking changes. This helps assess the impact and feasibility of each change.
- **Traceability**: Maintaining traceability between requirements, design, implementation, and tests ensures that the effect of a change is understood and managed throughout the system.
- **Stakeholder Involvement**: Changes should be discussed and agreed upon by all relevant stakeholders to ensure alignment with business objectives and user expectations.
- **Documentation Updates**: All changes must be documented to maintain an accurate and up-to-date requirements specification.