Parul® University
Vadodara, Gujarat

NAAC GRADE A++

# Requirements Engineering

Study Guide

AMARJIT SEAL
AIML, PIET
Parul University

# 1. INTRODUCTION TO REQUIREMENTS ENGINEERING

## Definition

Requirements Engineering is the disciplined application of proven principles, methods, tools, and notation to describe a proposed system's intended behavior and its associated constraints. It establishes the foundation for software design and development by clearly understanding and documenting what the customer needs and what the system must do.

## Core Objectives

- Understand customer/stakeholder needs clearly
- Define system boundaries and scope
- Establish feasibility and cost-benefit analysis
- Create unambiguous specifications for developers
- Ensure all stakeholders have consistent understanding
- Manage changes throughout the development lifecycle

## Importance in SDLC

Requirements Engineering is critical because:

- **60-70% of defects** stem from poor requirements
- Reduces rework and scope creep
- Prevents costly misunderstandings
- Guides design, testing, and implementation
- Provides traceability and validation baseline

---

# 2. PROBLEM RECOGNITION

## Definition

Problem recognition is the initial phase where the core issues or needs that prompt software development are identified. It establishes the basic understanding of the problem domain and the nature of the solution.

## Key Activities

1. **Stakeholder Identification**: Identify all parties affected by or interested in the solution
2. **Problem Definition**: Clearly articulate the pain points and challenges
3. **Scope Establishment**: Define project boundaries and constraints
4. **Feasibility Analysis**: Assess technical, economic, and organizational feasibility

5. **Solution Visioning**: Outline the general approach to address the problem

### Key Questions to Ask

- Who is behind the request?
- What problems will this solution address?
- Who are the stakeholders?
- What is the business context?
- What are the constraints (time, budget, technology)?
- Is the proposed solution technically feasible?
- What are the success criteria?

### Output of Problem Recognition

- Problem Statement Document
- Stakeholder List
- Initial Project Scope
- High-level Business Case
- Feasibility Report

---

# 3. REQUIREMENTS ENGINEERING TASKS AND PROCESSES

## Overview

Requirements Engineering encompasses **seven distinct tasks** that form a structured process:

$$\text{Inception} \rightarrow \text{Elicitation} \rightarrow \text{Elaboration} \rightarrow \text{Negotiation} \rightarrow \text{Specification} \rightarrow \text{Validation} \rightarrow \text{Management}$$

## 3.1 Inception (Task 1)

**Objective**: Establish basic understanding of the problem and solution nature

**Key Activities**:

- Identifying stakeholders and their roles
- Establishing communication paths
- Defining project scope and boundaries
- Answering key business questions
- Creating a project charter or vision statement

**Deliverables**:

- Stakeholder Register

- Project Charter
- Initial Scope Statement
- Communication Plan

**Key Questions**:

- Why is the system being developed?
- Who are the stakeholders?
- What are the system boundaries?
- What success criteria define this project?

---

## 3.2 Elicitation (Task 2)

**Objective**: Gather all software requirements from diverse sources

**Definition**: The process of discovering, extracting, and documenting requirements through communication with stakeholders and domain analysis.

### Elicitation Techniques

1. **Interviews**
    - Structured: Follow predefined questions
    - Semi-structured: Mix of prepared and exploratory questions
    - Unstructured: Open-ended exploration
    - Best for: Detailed understanding, relationship building
    - Drawback: Time-consuming, one-on-one intensive
2. **Focus Groups**
    - Multiple stakeholders discussing issues collectively
    - Best for: Identifying common themes, building consensus
    - Drawback: Dominant voices may overshadow others
3. **Document Analysis**
    - Reviewing existing documentation, standards, procedures
    - Best for: Understanding business processes, constraints
    - Drawback: Documents may be outdated or inaccurate
4. **Observation/Ethnography**
    - Watching users perform their tasks
    - Best for: Understanding workflow nuances, workarounds
    - Drawback: Subjects may behave differently when observed

5. **Surveys/Questionnaires**
    - Structured questions to large user groups
    - Best for: Quantifying preferences, reaching many users
    - Drawback: Low response rates, limited clarification
6. **Prototyping**
    - Building mockups or working models
    - Best for: Clarifying unclear requirements, getting feedback early
    - Drawback: Users may assume prototype is near-final product
7. **JAD (Joint Application Design) Sessions**
    - Structured workshops with mixed stakeholder groups
    - Best for: Building consensus, resolving conflicts quickly
    - Drawback: Requires skilled facilitators
8. **User Stories**
    - Short, informal descriptions from user perspective
    - Format: "As a [role], I want [feature] so that [benefit]"
    - Best for: Agile development, user-centric focus
9. **Mind Mapping**
    - Visual exploration of problem space
    - Best for: Brainstorming, organizing complex information
10. **Role-Playing**
    - Users act out scenarios
    - Best for: Understanding user emotions, edge cases

## Problems in Elicitation

- **Scope Problems**: Unclear boundaries, scope creep
- **Understanding Gaps**: Ambiguity, miscommunication
- **Volatility**: Requirements changing frequently
- **Hidden Requirements**: Tacit knowledge not articulated
- **Conflicting Views**: Different stakeholders want different things

## Mitigation Strategies

- Diverse elicitation techniques
- Frequent stakeholder engagement
- Prototyping and feedback cycles

- Traceability documentation
- Change management process

---

## 3.3 Elaboration (Task 3)

**Objective**: Refine and detail gathered requirements

**Key Activities**:

- Organizing requirements hierarchically
- Creating graphical models (DFDs, ER diagrams, state diagrams)
- Adding context and dependencies
- Defining interfaces
- Detailing business rules and constraints

**Modeling Techniques**:

- **Data Flow Diagrams (DFDs)**: Show data movement
- **Entity-Relationship Diagrams (ERDs)**: Depict data structure
- **State Diagrams**: Illustrate system states and transitions
- **Sequence Diagrams**: Show actor-system interactions
- **Class Diagrams**: Define object structure (OOP systems)
- **Context Diagrams**: Show system boundaries and interfaces

**Deliverables**:

- Requirements models and diagrams
- Data dictionary
- Interface specifications
- Business rules documentation

---

## 3.4 Negotiation (Task 4)

**Objective**: Resolve conflicts and prioritize requirements

**Key Activities**:

- Identifying conflicting requirements
- Resolving conflicts through discussion and compromise
- Prioritizing requirements (Must-Have, Should-Have, Nice-to-Have)
- Assessing resource availability

- Setting realistic expectations

**Prioritization Methods**:

1. **MoSCoW Method**: Must have, Should have, Could have, Won't have
2. **Weighted Ranking**: Assign weights and calculate priority scores
3. **Risk-Based**: High-risk items prioritized first
4. **Value-Based**: Items providing maximum business value prioritized

**Conflict Resolution Strategies**:

- Negotiation with stakeholders
- Compromise on scope or timeline
- Phased implementation approach
- Re-scoping project if necessary

**Output**:

- Prioritized Requirements List
- Conflict Resolution Documentation
- Approved Requirements Baseline

---

## 3.5 Specification (Task 5)

**Objective**: Document requirements in formal, unambiguous manner

**Definition**: Creating detailed, structured documentation that clearly defines what the system must do

**See detailed section below on Requirements Specification**

---

## 3.6 Validation (Task 6)

**Objective**: Ensure requirements are correct, complete, and feasible

**Definition**: Verification that requirements meet stakeholder needs and are free from defects

**See detailed section below on Requirements Validation**

---

## 3.7 Management (Task 7)

**Objective**: Control and track requirements throughout development

**Key Activities**:

- Maintaining requirements baseline

- Managing change requests
- Tracking requirement status
- Maintaining traceability
- Communicating changes to all stakeholders

**Change Control Process**:

1. Submission of change request
2. Assessment of impact
3. Decision (approve/reject/defer)
4. Implementation if approved
5. Communication to all parties

**Traceability Matrix**: Maps requirements to design, code, and test cases to ensure full coverage

---

# 4. REQUIREMENTS SPECIFICATION (SRS)

## Definition

Requirements Specification is the process and output of formally documenting all software requirements in a structured, comprehensive document called the Software Requirements Specification (SRS).

## Purpose of SRS

- Serves as contract between client and developer
- Communication medium between stakeholders
- Baseline for design and development
- Reference for testing and validation
- Foundation for project planning and estimation

## Characteristics of Good Requirements

**SMART Criteria**:

- **S**pecific: Clear and precise
- **M**easurable: Can be verified objectively
- **A**chievable: Realistic and feasible
- **R**elevant: Necessary for system success
- **T**raceable: Can be traced to design and test cases

**Additional Qualities**:

- Unambiguous: Single interpretation

- Complete: Sufficient detail
- Consistent: No contradictions
- Prioritized: Ranked by importance
- Verifiable: Can be tested objectively

## Types of Requirements in SRS

### 1. Functional Requirements

**Definition**: Describe what the system must do; specify functions and behaviors

**Characteristics**:

- Focus on system behavior
- Directly observable in final product
- Can be tested objectively
- Define features and operations

**Examples**:

- User authentication with username/password
- Generate monthly financial reports
- Process payment transactions
- Search and filter database records
- Send automated email notifications

**Format**: "The system shall [action]"

- "The system shall allow users to login with credentials"
- "The system shall generate sales reports in PDF format"

### 2. Non-Functional Requirements

**Definition**: Specify qualities and constraints on how the system performs functions

**Sub-categories**:

**a) Performance Requirements**

- Response time < 2 seconds for user actions
- System throughput: 1000 transactions/minute
- Database query response: < 500ms

**b) Security Requirements**

- All data encrypted using AES-256
- Password complexity: minimum 8 characters

- Multi-factor authentication for sensitive operations
- Compliance with ISO 27001 standards

**c) Reliability Requirements**

- System uptime: 99.9%
- Mean time between failures (MTBF) > 1000 hours
- Recovery time objective (RTO) < 4 hours

**d) Usability Requirements**

- Web interface compatible with Chrome, Firefox, Safari
- Average training time: < 2 hours
- 95% of users complete task without assistance
- WCAG 2.1 Level AA accessibility compliance

**e) Scalability Requirements**

- Support 100-10,000 concurrent users
- Database can handle 1 million records
- Linear performance degradation with load

**f) Maintainability Requirements**

- Code modularity with <500 lines per function
- API documentation for all components
- 80% code test coverage minimum
- Deployable within 30 minutes

**g) Compatibility Requirements**

- Browser support: IE11+, Chrome 90+, Firefox 88+
- Operating systems: Windows 7+, macOS 10.12+, Linux
- Database: Oracle 12c, MySQL 8.0, PostgreSQL 13+

**h) Legal/Compliance Requirements**

- GDPR data protection compliance
- PCI-DSS for payment processing
- SOX compliance for financial data
- HIPAA for healthcare information

## SRS Document Structure

**Standard IEEE 830 Format**:

1. Introduction
   1.1 Purpose
   1.2 Scope
   1.3 Definitions, Acronyms, Abbreviations
   1.4 References
   1.5 Overview

2. Overall Description
   2.1 Product Perspective
   2.2 Product Functions
   2.3 User Characteristics
   2.4 Constraints
   2.5 Assumptions and Dependencies

3. Specific Requirements
   3.1 Functional Requirements
   3.2 Non-Functional Requirements
   3.3 External Interface Requirements
   3.4 Data Requirements

4. Appendices

5. Index

## Requirements Organization

- **By Feature**: Group related functionality
- **By Actor/User Role**: Organize by who performs action
- **By System Module**: Organize by architectural component
- **By Priority**: Must-have, should-have, nice-to-have

## Specification Methods

1. **Natural Language**
   - Prose description in user language
   - Easy to understand but ambiguous
   - Most common in industry
2. **Structured Natural Language**
   - Formatted templates with specific sections
   - Reduces ambiguity while remaining understandable
   - Example: "The system SHALL [action] WHEN [condition]"
3. **Formal Specification**
   - Mathematical notation (Z, B notation, temporal logic)
   - Unambiguous and verifiable

  - Difficult to understand for non-technical stakeholders
  - Used in safety-critical systems

4. **Graphical Specification**
   - UML diagrams, flowcharts, state machines
   - Visual communication
   - Often combined with textual specification
5. **Use Cases**
   - Described in detail below

---

# 5. USE CASES AND FUNCTIONAL SPECIFICATION

## Definition of Use Cases

A use case is a description of interactions between a system and its external actors that accomplish a specific goal. It captures what the user wants to do and how the system responds.

## Components of Use Case

### 1. Actor

- External entity (user or external system) that interacts with the system
- Types:
  - **Primary Actor**: Initiates the use case
  - **Secondary Actor**: Participates in the use case
- Examples: Customer, Admin, External API, Payment Gateway

### 2. Preconditions

- System state and actor assumptions that must be true before use case starts
- Example: "User is logged in" or "Product exists in database"

### 3. Main Flow (Happy Path)

- Step-by-step description of normal, successful interaction
- Numbered sequence
- Example:
  a. User enters username and password
  b. System validates credentials
  c. System grants access
  d. System displays home page

## 4. Alternative Flows (Alternate Scenarios)

- Variations on main flow or different paths
- Handle different conditions or user choices
- Example:
  Alt 4a: Password incorrect
  a. System displays error message
  b. System re-displays login page

## 5. Exception Flows (Error Scenarios)

- Handle error conditions or exceptional situations
- System recovery steps
- Example:
  Exc: Database unavailable
  a. System displays "Service temporarily unavailable"
  b. Log error for admin notification

## 6. Postconditions

- System state after use case completion
- What has been achieved or changed
- Example: "User is logged in and viewing home page" or "Order is created in system"

## Use Case Template

USE CASE: [UC-001] User Login

ACTOR: User (Primary)

PRECONDITION:

- User is not currently logged in
- System is operational

MAIN FLOW:

1. User navigates to login page
2. User enters username and password
3. System validates credentials against database
4. System verifies credentials are correct
5. System creates session for user
6. System redirects to home page
7. User sees personalized dashboard

ALTERNATIVE FLOWS:

Alt 1: User has "Remember Me" enabled
3a. System retrieves stored credentials
3b. System auto-fills login form
4. Continue to step 3

Alt 2: User forgot password
3a. User clicks "Forgot Password"
3b. System displays password reset form
3c. Use Case: Reset Password (UC-002)
3d. Return to main flow step 2

EXCEPTION FLOWS:
Exc 1: Invalid credentials
4a. System displays "Invalid username or password"
4b. System clears password field
4c. System re-displays login form
Return to step 2

Exc 2: Account locked (too many failed attempts)
4a. System displays "Account temporarily locked"
4b. System suggests contacting support
Exit use case

Exc 3: Database connection fails
3a. System displays "System unavailable, please try again later"
3b. Log error for technical team
Exit use case

Exc 4: User enters SQL injection attempt
3a. System sanitizes input
3b. System treats as invalid credentials
Continue to Exc 1

POSTCONDITION:

- User is authenticated
- User session created
- User viewing personalized dashboard

FREQUENCY: Multiple times daily
PRIORITY: Critical

## Use Case Diagram

- Visual representation showing:
    - System boundary (rectangle)
    - Actors (stick figures or symbols)
    - Use cases (ovals/ellipses)
    - Relationships (associations, includes, extends)

- Shows high-level system functionality and interactions

## Functional Specification Components

1. **Context Diagram**
    - Shows system as single process
    - Displays external entities and data flows
    - Defines system boundary
2. **User Requirements**
    - Grouped by user role/type
    - Describes what user wants to accomplish
    - Format: "User shall be able to..."
3. **System Requirements**
    - Describes internal system operations
    - How system achieves user goals
    - Format: "System shall provide..."
4. **Data Flow Diagrams (DFDs)**
    - Level 0 (context): Highest level overview
    - Level 1: Major processes and data stores
    - Detailed levels: Sub-processes and details
5. **Entity-Relationship Diagram (ERD)**
    - Shows data structures and relationships
    - Defines database schema conceptually
6. **Business Rules**
    - Rules governing system behavior
    - Constraints on data and processes
    - Example: "Discount available only if order > $100"
7. **Interface Specifications**
    - Screen mockups or wireframes
    - API specifications
    - Report formats
    - Menu structures

# 6. REQUIREMENTS VALIDATION

## Definition

Requirements validation is the process of ensuring that documented requirements correctly represent customer needs and are of sufficient quality to guide design and development.

## Purpose of Validation

- Detect defects in requirements early
- Ensure customer satisfaction
- Verify requirements completeness and consistency
- Check feasibility and realism
- Prevent costly downstream changes

## Validation Techniques

### 1. Reviews and Inspections

- **Formal Technical Review (FTR)**
  - Structured meeting with checklist
  - Authors present, stakeholders review
  - Formal findings documented
  - Best for: Comprehensive review, large documents
- **Informal Review**
  - Less formal one-on-one discussions
  - Collaborative problem-solving
  - Best for: Quick clarifications, feedback
- **Inspection**
  - Detailed systematic review against criteria
  - Trained inspectors follow process
  - High-defect detection rate
  - Best for: Critical requirements
- **Walkthrough**
  - Author presents requirements step-by-step
  - Audience asks questions and provides feedback
  - Collaborative tone
  - Best for: Understanding and consensus building

## 2. Prototyping

- Build working model or mockup
- Users interact and provide feedback
- Refine requirements based on user reaction
- Advantages: Discovers missing/unclear requirements
- Disadvantages: Time and cost, users may expect final product

## 3. Test Case Generation

- Develop test cases from requirements
- If test cases cannot be written, requirement is unclear
- Ensures testability of requirements
- Catches missing details

## 4. Simulation/Modeling

- Build executable models
- Simulate system behavior
- Verify logic and interactions
- Useful for complex systems

## 5. Automated Analysis Tools

- Static analysis for syntax errors
- Consistency checking
- Completeness verification
- Traceability checking

## 6. Stakeholder Sign-off

- Formal approval by key stakeholders
- Documented acceptance
- Commitment to requirements baseline

## Validation Checklist

For each requirement, verify:

**Correctness**:

- ☐ Requirement matches stakeholder needs
- ☐ Requirement aligns with business objectives
- ☐ No discrepancies with other requirements

**Clarity**:

- ☐ Unambiguous wording
- ☐ No multiple interpretations possible
- ☐ Jargon explained or avoided

**Completeness**:

- ☐ Sufficient detail for implementation
- ☐ All necessary context included
- ☐ No critical information missing

**Testability**:

- ☐ Can be verified objectively
- ☐ Test criteria defined
- ☐ Measurable success criteria

**Feasibility**:

- ☐ Technically achievable
- ☐ Within resource constraints
- ☐ Within time and budget constraints

**Traceability**:

- ☐ Linked to source (stakeholder/document)
- ☐ Can trace to design elements
- ☐ Can trace to test cases

**Consistency**:

- ☐ No conflicts with other requirements
- ☐ Compatible with constraints
- ☐ Aligned with assumptions

**Prioritization**:

- ☐ Properly prioritized
- ☐ Dependencies identified
- ☐ Implementation sequence defined

---

# 7. REQUIREMENTS ANALYSIS

## Definition

Requirements analysis is the process of examining, refining, and elaborating elicited requirements to ensure they are feasible, consistent, complete, and unambiguous.

## Objectives of Analysis

- Gain deeper understanding of requirements
- Identify conflicts and inconsistencies
- Decompose complex requirements
- Define traceability relationships
- Assess feasibility and impact
- Establish requirements baseline

## Analysis Activities

### 1. Context Analysis

- Define system boundaries
- Identify external interfaces
- Document assumptions
- Create context diagram showing:
  - System as single process
  - External entities (actors)
  - Data flows in/out of system

### 2. Requirements Decomposition

- Break down complex requirements into simpler components
- Hierarchical structuring
- Relate high-level to detailed requirements
- Example: "Process Order" decomposes to:
  - Validate Order
  - Calculate Tax
  - Process Payment
  - Generate Invoice

### 3. Conflict Identification and Resolution

- **Types of conflicts**:
  - Requirement conflicts (mutually exclusive requirements)
  - Stakeholder conflicts (different people want different things)
  - Technical conflicts (technically infeasible combinations)

- **Resolution methods**:
  - Negotiation and compromise
  - Phased implementation (do conflicting items in different releases)
  - Re-scoping project
  - Adding constraints or conditions

## 4. Consistency Analysis

- Check for contradictions between requirements
- Verify naming conventions consistent
- Ensure similar requirements formatted similarly
- Validate cross-references accurate

## 5. Completeness Analysis

- Verify all necessary functionality included
- Check all stakeholder needs addressed
- Ensure all interfaces specified
- Verify error conditions handled

## 6. Requirements Modeling

**Data Flow Diagrams (DFDs)**:

- Show data movement through system
- Identify data stores and transformations
- Help verify completeness

**Entity-Relationship Diagrams (ERDs)**:

- Define data structure
- Identify entities and relationships
- Specify data requirements

**State Diagrams**:

- Show system states and transitions
- Identify state-based requirements
- Verify complete state coverage

**Sequence Diagrams**:

- Show actor-system interactions over time
- Verify message flow and timing
- Identify alternative scenarios

### 7. Feasibility Assessment

- **Technical Feasibility**: Can technology support requirements?
- **Resource Feasibility**: Do we have skilled people?
- **Time Feasibility**: Can we deliver within schedule?
- **Financial Feasibility**: Is budget adequate?
- **Organizational Feasibility**: Aligned with organizational goals?

### 8. Impact Analysis

- Assess impact of requirement changes
- Identify dependencies
- Evaluate cost of implementation
- Determine priority and scheduling

## Analysis Phases

**Phase 1: Requirements Understanding**

- Review and comprehend all requirements
- Ask clarification questions
- Document assumptions
- Create glossary of terms

**Phase 2: Requirements Organization**

- Group related requirements
- Establish hierarchies
- Define dependencies
- Create work breakdown structure

**Phase 3: Requirements Modeling**

- Create graphical representations
- Document data flows
- Define interfaces
- Specify algorithms or logic

**Phase 4: Requirements Finalization**

- Resolve all conflicts
- Achieve stakeholder consensus
- Establish baseline
- Prepare for design phase

# 8. IMPORTANT EXAM QUESTIONS WITH ANSWERS

## SHORT ANSWER QUESTIONS (2-5 marks each)

**Q1: Define Requirements Engineering and state its importance.**

**Answer**: Requirements Engineering is the disciplined application of proven principles, methods, tools, and notation to describe a proposed system's intended behavior and its associated constraints.

Importance:

- Establishes foundation for design and development
- Reduces defects and rework (60-70% of defects stem from poor requirements)
- Prevents scope creep and manages stakeholder expectations
- Provides traceability for testing and verification
- Acts as contract between client and developer
- Reduces development cost and time by preventing misunderstandings
- Ensures all stakeholders have consistent understanding

---

**Q2: Distinguish between Functional and Non-Functional Requirements.**

**Answer**:

| Aspect | Functional | Non-Functional |
|---|---|---|
| Definition | Describe what system must do | Describe how well system does it |
| Focus | System behavior and features | System qualities and constraints |
| Examples | Login, payment, search | Performance, security, reliability |
| Testing | Direct functionality testing | Quality attribute testing |
| Visibility | Observable in final product | May not be directly visible |
| User-facing | Usually user-visible | Often transparent to user |
| Examples | "Generate monthly report" | "Report generated in < 2 seconds" |

---

**Q3: List and briefly explain the seven tasks of Requirements Engineering.**

**Answer**:

1. **Inception**: Establish basic understanding of problem and solution; identify stakeholders
2. **Elicitation**: Gather requirements from diverse sources using various techniques

3. **Elaboration**: Refine and detail requirements; create models and diagrams
4. **Negotiation**: Resolve conflicts, prioritize requirements, set realistic expectations
5. **Specification**: Document requirements formally in SRS document
6. **Validation**: Ensure requirements meet stakeholder needs and are of quality
7. **Management**: Control changes, maintain traceability, track requirement status

---

**Q4: What are the main elicitation techniques? Explain any three.**

**Answer**:
Main techniques: Interviews, Focus Groups, Document Analysis, Observation, Surveys, Prototyping, JAD, User Stories, Mind Mapping, Role-playing

**Three important techniques**:

1. **Interviews**
   - One-on-one or small group discussions
   - Can be structured, semi-structured, or unstructured
   - Advantage: Detailed understanding, relationship building
   - Disadvantage: Time-consuming, expensive
2. **Prototyping**
   - Creating mockups or working models
   - Advantage: Visual feedback, clarifies unclear requirements, early user involvement
   - Disadvantage: Users may expect prototype to be final product
3. **JAD Sessions**
   - Joint Application Design workshops
   - Structured meetings with mixed stakeholder groups
   - Advantage: Quick consensus building, resolves conflicts rapidly
   - Disadvantage: Requires skilled facilitator, may dominate scheduling

---

**Q5: What is a Use Case? Describe its key components.**

**Answer**: A use case is a description of interactions between a system and external actors that accomplish a specific goal or business objective.

**Key Components**:

1. **Actor**: External entity (user or system) that interacts with the system
2. **Precondition**: System state that must be true before use case starts
3. **Main Flow**: Step-by-step sequence of successful interaction (happy path)
4. **Alternative Flows**: Variations or different paths through the use case

5. **Exception Flows**: Error conditions and how system recovers
6. **Postcondition**: System state after use case completes successfully

---

**Q6: What is a Software Requirements Specification (SRS) document? What should it contain?**

**Answer**: An SRS is a formal document that specifies all requirements for a software system in an unambiguous, complete, and verifiable manner. It serves as a contract between client and developer.

**SRS Contents**:

1. Introduction (purpose, scope, references)
2. Overall Description (product perspective, functions, user characteristics)
3. Specific Requirements (functional, non-functional, interface, data requirements)
4. Appendices
5. Index

**Key characteristics of good SRS**:

- Clear and unambiguous
- Complete and consistent
- Organized and structured
- Traceable and verifiable
- Modifiable and maintainable

---

**Q7: Explain the difference between Verification and Validation.**

**Answer**:

| Verification | Validation |
|---|---|
| "Are we building the product right?" | "Are we building the right product?" |
| Checks if product meets specifications | Checks if product meets customer needs |
| Internal check by development team | External check with stakeholders |
| Done during development | Done after development |
| Reviews, inspections, testing | User acceptance testing, UAT |
| Example: Code matches requirements | Example: System solves actual problem |

---

**Q8: What are the problems that occur during requirements elicitation?**

**Answer**:

1. **Scope Problems**: Unclear project boundaries, scope creep

2. **Understanding Gaps**: Ambiguities, miscommunications, tacit knowledge
3. **Volatility**: Requirements changing during elicitation or development
4. **Conflicting Views**: Different stakeholders have different/conflicting needs
5. **Hidden Requirements**: Critical requirements not initially expressed
6. **Resource Constraints**: Insufficient time/budget for thorough elicitation
7. **Communication Barriers**: Language/domain expertise differences

**Mitigation strategies**:

- Use diverse elicitation techniques
- Frequent stakeholder engagement
- Prototyping and feedback cycles
- Formal documentation and sign-off
- Change management process
- Clear communication and glossaries

---

**Q9: Describe the requirements validation process and techniques used.**

**Answer**: Validation ensures requirements correctly represent customer needs and are quality.

**Validation Techniques**:

1. **Reviews and Inspections**
    - Formal Technical Reviews with stakeholders
    - Structured walkthroughs
    - Peer reviews
    - Inspection against criteria
2. **Prototyping**
    - Build mockups or working models
    - Get user feedback on prototype
    - Refine requirements based on feedback
3. **Test Case Generation**
    - Write test cases from requirements
    - If test cases cannot be written, requirement needs clarification
    - Ensures testability
4. **Simulation and Modeling**
    - Build executable models
    - Simulate system behavior

   - Verify logic and interactions
5. **Automated Tools**
   - Syntax checking
   - Consistency verification
   - Completeness checking
   - Traceability analysis
6. **Stakeholder Sign-off**
   - Formal approval
   - Documented acceptance
   - Commitment to baseline

---

**Q10: What is requirements traceability? Why is it important?**

**Answer**: Requirements traceability is the ability to track each requirement from its source through design, implementation, and testing to ensure complete coverage and verification.

**Types of Traceability**:

- Forward traceability: Requirement → Design → Code → Test
- Backward traceability: Test → Code → Design → Requirement

**Importance**:

1. Ensures all requirements are implemented
2. Ensures all code is traced back to requirements
3. Helps assess impact of changes
4. Provides visibility and accountability
5. Facilitates testing and validation
6. Aids in change management
7. Ensures compliance and quality

**Traceability Matrix**:

- Table showing relationships between requirements and other artifacts
- Rows: Requirements
- Columns: Design elements, code modules, test cases
- Identifies gaps and ensures coverage

---

## MEDIUM ANSWER QUESTIONS (5-10 marks each)

**Q11: Explain the Requirements Engineering process with a diagram. Discuss each phase in detail.**

**Answer**:

```
                 Inception
                     ↓
                Elicitation
                     ↓
                Elaboration
                     ↓
                Negotiation
                     ↓
                Specification
                     ↓
                Validation
                     ↓
                Management
```

**Detailed Explanation of Each Phase**:

**1. Inception (Problem Recognition)**

- Establish project charter
- Identify stakeholders
- Define project scope and boundaries
- Answer fundamental business questions
- Create vision statement
- Output: Project charter, stakeholder register, initial scope

**2. Elicitation**

- Gather requirements from stakeholders
- Use interviews, workshops, prototypes
- Discover functional and non-functional needs
- Document user stories
- Identify constraints and assumptions
- Output: Elicited requirements document, use cases

**3. Elaboration**

- Refine gathered requirements
- Add detail and context
- Create requirement models (DFD, ERD)
- Define interfaces
- Document business rules
- Output: Detailed requirement specifications, models

**4. Negotiation**

- Identify conflicting requirements
- Resolve conflicts through discussion
- Prioritize requirements (MoSCoW method)
- Set realistic expectations
- Plan phased implementation if needed
- Output: Prioritized requirements, conflict resolution log

**5. Specification**

- Document requirements formally in SRS
- Use structured templates
- Define functional and non-functional requirements
- Include use cases and interface specifications
- Ensure unambiguous, complete documentation
- Output: Software Requirements Specification (SRS)

**6. Validation**

- Review requirements with stakeholders
- Check completeness, consistency, feasibility
- Generate test cases
- Use prototyping for clarification
- Conduct formal technical reviews
- Output: Validated requirement baseline

**7. Management**

- Maintain requirements baseline
- Process change requests
- Track requirement status
- Maintain traceability matrix
- Communicate changes to all stakeholders
- Output: Change logs, traceability documentation

---

**Q12: Develop a use case for an "Online Banking System - Transfer Funds" operation.**

**Answer**:

USE CASE: UC-005 Transfer Funds

ACTOR: Customer (Primary), Bank System (Secondary)

PRECONDITION:

- Customer is logged into online banking system
- Customer has at least two active accounts
- Account has sufficient balance
- System is operational

MAIN FLOW:

1. Customer selects "Transfer Funds" from menu
2. System displays transfer form with source account dropdown
3. Customer selects source account
4. System displays available balance for source account
5. Customer selects destination account (own or external)
6. Customer enters transfer amount
7. System validates amount ($> 0$, $\leq$ available balance)
8. System displays transfer summary
9. Customer reviews and confirms transfer
10. System generates OTP (One-Time Password)
11. System displays OTP prompt
12. Customer enters OTP from SMS/email
13. System validates OTP
14. System processes transfer
15. System deducts amount from source account
16. System credits amount to destination account
17. System generates transaction receipt
18. System displays receipt to customer
19. System sends confirmation email/SMS
20. Use case ends - Transfer completed successfully

ALTERNATIVE FLOWS:

Alt 1: Transfer to new external account
5a. Customer selects "Add New Recipient"
5b. System displays external account form
5c. Customer enters recipient bank details
5d. System validates bank details format
5e. System marks account for verification
5f. Continue to main flow step 6

Alt 2: Customer requests to schedule transfer
9a. Customer selects "Schedule Transfer"
9b. System displays date/time selection
9c. Customer selects future date and time
9d. Continue to main flow step 10

Alt 3: Customer cancels transfer
9a. Customer clicks "Cancel"
9b. System displays confirmation dialog
9c. Customer confirms cancellation
9d. System returns to main menu
Use case ends

EXCEPTION FLOWS:

Exc 1: Insufficient balance
7a. System displays "Insufficient balance in selected account"
7b. System displays available balance
7c. Customer can modify amount (go to step 6)
or cancel transfer (go to Alt 3)

Exc 2: Invalid OTP entered
13a. System displays "Invalid OTP"
13b. System provides option to resend OTP
13c. If user requests new OTP:
- System generates new OTP (max 3 attempts)
- Customer enters new OTP
- Continue to step 13
13d. If user exceeds 3 attempts:
- System blocks transaction
- System locks account temporarily
- Transfer cancelled

Exc 3: Account locked (suspicious activity detected)
7a. System detects large/unusual transaction
7b. System displays security verification needed
7c. System may require additional verification
7d. If verification fails, transaction blocked

Exc 4: Destination account invalid
6a. System validates destination account
6b. If account number invalid or doesn't exist
6c. System displays "Invalid recipient account"
6d. Customer corrects account details
Return to step 6

Exc 5: Bank system unavailable
14a. System displays "Service temporarily unavailable"
14b. System offers to retry or schedule for later
14c. No funds deducted from account

POSTCONDITION:

- Source account balance reduced by transfer amount
- Destination account credited with transfer amount
- Transaction recorded in both accounts
- Confirmation sent to customer
- Audit trail created for compliance

FREQUENCY: Multiple times daily
PRIORITY: High
COMPLEXITY: Medium

---

**Q13: What are the steps in developing a functional specification? Give an example functional specification for a simple ATM system.**

**Answer**:

**Steps in Developing Functional Specification**:

1. **Identify User Roles**: Define all types of users (customer, admin, operator)
2. **Gather User Goals**: What does each user want to accomplish?
3. **Define System Context**: Boundary, external entities, interactions
4. **Create Use Cases**: Document actor-system interactions
5. **Specify Business Rules**: Rules governing system behavior
6. **Define Data Requirements**: What data is needed, structure, relationships
7. **Interface Design**: Screens, menus, reports, APIs
8. **Document Constraints**: Performance, security, compliance requirements
9. **Review and Validate**: With stakeholders
10. **Create Test Cases**: Ensure specification is testable

**Example Functional Specification: ATM System**

**1. SYSTEM OVERVIEW**
An ATM (Automated Teller Machine) allows customers to perform banking transactions without visiting a branch.

**2. USER ROLES**

- Customer: End user performing transactions
- Bank: Provides services
- Support Staff: Maintains machine

**3. FUNCTIONAL REQUIREMENTS**

**3.1 Authentication**

- System shall validate customer card
- System shall verify PIN (Personal Identification Number)
- System shall allow 3 attempts before blocking card
- FR-1.1: Customer must insert valid ATM card
- FR-1.2: System must display PIN entry screen
- FR-1.3: System must validate PIN against database

**3.2 Withdrawal**

- FR-2.1: System shall display withdrawal amount options ($20, $50, $100, $200, etc.)
- FR-2.2: System shall allow custom amount entry
- FR-2.3: System shall validate amount (available balance, maximum limit)
- FR-2.4: System shall dispense requested cash
- FR-2.5: System shall deduct amount from account
- FR-2.6: System shall print receipt

**3.3 Deposit**

- FR-3.1: System shall display deposit instructions
- FR-3.2: System shall accept envelope with cash/cheques
- FR-3.3: System shall credit provisional amount to account
- FR-3.4: System shall verify amount later (human verification)
- FR-3.5: System shall print deposit receipt

**3.4 Balance Inquiry**

- FR-4.1: System shall retrieve account balance from database
- FR-4.2: System shall display current balance
- FR-4.3: System shall display recent transactions (last 10)
- FR-4.4: System shall offer print receipt option

**3.5 Change PIN**

- FR-5.1: System shall verify current PIN
- FR-5.2: System shall accept new PIN (4-6 digits)
- FR-5.3: System shall confirm new PIN
- FR-5.4: System shall update PIN in database

## 4. NON-FUNCTIONAL REQUIREMENTS

### 4.1 Performance

- NR-1.1: PIN validation within 2 seconds

- NR-1.2: Cash dispensing within 30 seconds
- NR-1.3: Transaction processing within 5 seconds

**4.2 Security**

- NR-2.1: Encrypt PIN during transmission
- NR-2.2: Encrypt data stored on machine
- NR-2.3: No display of PIN on screen
- NR-2.4: Block card after 3 failed PIN attempts
- NR-2.5: Transaction timeout after 5 minutes of inactivity
- NR-2.6: Audit trail of all transactions

**4.3 Availability**

- NR-3.1: System uptime 99.5%
- NR-3.2: Graceful handling of network failures
- NR-3.3: Offline mode for limited functionality

**4.4 Usability**

- NR-4.1: Simple menu navigation
- NR-4.2: Large, readable text
- NR-4.3: Audio feedback for accessibility
- NR-4.4: Clear error messages

**5. USE CASES**

**USE CASE: Withdraw Cash**

1. Customer inserts card
2. System reads card
3. System prompts for PIN
4. Customer enters PIN
5. System validates PIN
6. System displays main menu
7. Customer selects "Withdraw"
8. System displays amount options
9. Customer selects or enters amount
10. System validates balance
11. System dispenses cash
12. System deducts amount from account
13. System prints receipt

14. System ejects card
15. Customer retrieves card, cash, and receipt

---

**Q14: Describe requirements validation techniques with examples. What is the importance of requirements validation?**

**Answer**: [Already covered in detail above - see Q9]

---

**Q15: Discuss the role of requirements analysis in software engineering. What modeling techniques are used?**

**Answer**:

**Role of Requirements Analysis**:

1. **Understand Requirements Better**: Gain deeper comprehension beyond surface level
2. **Identify Issues Early**: Find inconsistencies, gaps, and conflicts
3. **Ensure Feasibility**: Verify requirements are achievable with available resources
4. **Define Baselines**: Establish requirements baseline for design and development
5. **Plan Implementation**: Determine dependencies, sequencing, and priorities
6. **Enable Communication**: Create clear documentation for all stakeholders
7. **Reduce Risk**: Identify and mitigate potential problems early
8. **Improve Quality**: Ensure requirements are clear, complete, and testable

**Modeling Techniques Used in Requirements Analysis**:

**1. Data Flow Diagrams (DFDs)**

- Shows how data flows through system
- Components: Processes, data stores, external entities, data flows
- Levels: Context (level 0), detailed levels
- Helps verify completeness and identify data requirements
- Example: DFD for banking system showing money flow, customer data flow

**2. Entity-Relationship Diagrams (ERD)**

- Shows data structure and relationships
- Components: Entities, attributes, relationships
- Defines database schema
- Example: Entities (Customer, Account, Transaction) and relationships between them

**3. State Diagrams/State Machines**

- Shows system states and transitions
- Components: States, transitions, events, actions

- Verifies complete state coverage
- Example: Order states (New, Processing, Shipped, Delivered, Cancelled)

**4. Sequence Diagrams**

- Shows actor-system interactions over time
- Components: Actors, messages, time sequence
- Verifies message flow and timing
- Example: ATM withdrawal sequence showing steps over time

**5. Class Diagrams** (for OOP systems)

- Shows object classes and relationships
- Components: Classes, attributes, methods, relationships
- Defines system structure
- Example: Classes for Banking System (Customer, Account, Transaction)

**6. Context Diagrams**

- High-level view of system and environment
- Shows system as single process
- Identifies external entities and data flows
- Example: ATM system with Bank, Customer, and Regulator as external entities

---

## LONG ANSWER QUESTIONS (10-15 marks each)

**Q16: Compare different SDLC models (Waterfall, Agile, Spiral, V-Model) with respect to requirements engineering activities.**

**Answer**:

| Aspect | Waterfall | Agile | Spiral | V-Model |
|---|---|---|---|---|
| **RE Timing** | Upfront, comprehensive | Iterative, continuous | Incremental, iterative | Upfront with validation |
| **RE Detail** | Complete before design | Just-enough-needed | Evolves with risk | Complete before design |
| **Elicitation** | Single phase | Throughout sprints | Spiral cycles | Early and thorough |
| **Change** | Difficult/expensive | Embraced, frequent | Managed systematically | Managed through phases |
| **Documentation** | Heavy | Light, working software | Moderate, risk-focused | Heavy, aligned with testing |

| | | | | |
|---|---|---|---|---|
| **Validation** | End of requirements phase | Continuous with stakeholders | Each spiral cycle | Against test cases |
| **Traceability** | Formal matrix required | Lightweight tracking | Risk-based traceability | Strong end-to-end |
| **Scalability** | Large projects | Small-medium teams | Large, complex, risky | Large, safety-critical |

**Detailed Analysis**:

**1. Waterfall Model**

- Follows linear, sequential phases
- RE Characteristics:
    - Comprehensive elicitation upfront
    - Detailed SRS document created before design
    - Validation through formal reviews
    - Difficult to accommodate changes
    - Assumes requirements are stable
- Advantages: Clear baseline, comprehensive documentation
- Disadvantages: Late discovery of problems, inflexible

**2. Agile Model**

- Iterative development with frequent releases
- RE Characteristics:
    - Just-enough requirements (user stories)
    - Continuous elicitation in sprints
    - Frequent stakeholder engagement
    - Quick validation through demonstrations
    - Embraces requirement changes
- Advantages: Flexible, customer-centric, quick feedback
- Disadvantages: Light documentation, less comprehensive upfront

**3. Spiral Model**

- Risk-driven iterative model
- RE Characteristics:
    - Requirements refined each spiral
    - Risk assessment drives RE depth
    - Prototyping for risky areas

   - Gradual refinement of requirements
   - Multiple feedback cycles
- Advantages: Risk-aware, handles complex projects
- Disadvantages: More complex, expensive

**4. V-Model**

- Verification and Validation at each level
- RE Characteristics:
   - Comprehensive requirements at start
   - Linked to test cases
   - Each requirement has acceptance criteria
   - Strong traceability
   - Validation through testing
- Advantages: Quality-focused, good traceability
- Disadvantages: Less flexible, assumes stable requirements

---

**Q17: You are a requirements engineer for a "Hospital Management System" project. Develop comprehensive requirements specification document (SRS) outline with detailed sections.**

**Answer**:

HOSPITAL MANAGEMENT SYSTEM
SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

Version: 1.0
Date: January 2026
Prepared by: Requirements Engineering Team

---

1. INTRODUCTION

1.1 Purpose
This SRS document specifies all functional and non-functional requirements for the Hospital Management System (HMS). This document serves as the contract between the hospital stakeholders and the development team.

1.2 Scope
The HMS encompasses:

- Patient Management (registration, medical records)
- Appointment Scheduling
- Billing and Invoicing

- Pharmacy Management
- Lab Test Management
- Staff Management
- Report Generation
- Inventory Management

Out of Scope:

- Telemedicine
- Mobile app (Phase 2)
- Insurance claim processing

1.3 Definitions and Acronyms

- HMS: Hospital Management System
- EMR: Electronic Medical Record
- OPD: Out-Patient Department
- IPD: In-Patient Department
- QR: Quality Requirement
- SLA: Service Level Agreement

1.4 References

- IEEE 830 Standard for SRS
- Hospital Standards (JCI)
- HIPAA Compliance Regulations

1.5 Document Overview
This document contains overview, specific requirements,
and appendices organized in sections 2-4.

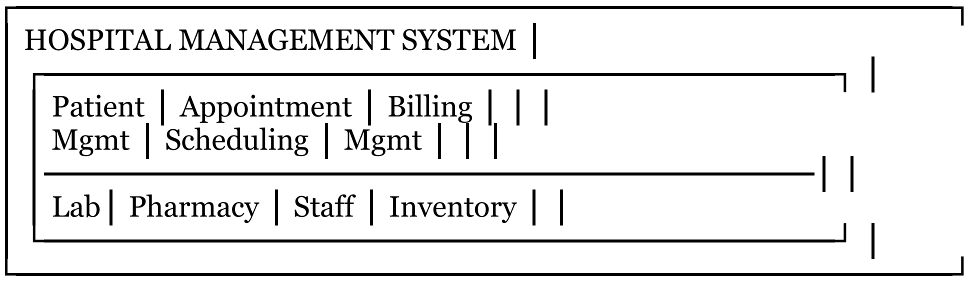
---

2. OVERALL DESCRIPTION

2.1 Product Perspective
HMS is a standalone system that integrates hospital operations.
It interfaces with:

- Patient registration at front desk
- Doctors' workstations
- Lab equipment
- Pharmacy systems
- Billing systems
- Government health registries (reporting)

System Context Diagram:



↕ ↕ ↕
Patients Government Insurance
Health Dept Companies

2.2 Product Functions

- Register and manage patient information
- Schedule and manage appointments
- Maintain electronic medical records
- Process billing and invoicing
- Manage pharmacy inventory
- Manage lab test procedures
- Generate reports
- Manage staff schedules

2.3 User Characteristics

- Front Desk Staff: Basic education, high school level training
- Doctors: Medical education, need detailed clinical features
- Lab Technicians: Technical school, require lab-specific features
- Pharmacists: Pharmacy education, inventory-focused
- Hospital Administrator: Management education, analytics-focused

2.4 Constraints

- Technical: Must run on Windows Server 2016+
- Compliance: HIPAA, patient data privacy
- Performance: Must handle 1000+ concurrent users
- Budget: $500,000 implementation cost
- Timeline: 18-month implementation schedule
- Staffing: Team of 20 developers

2.5 Assumptions and Dependencies

- Patient data already available (migration planned)

- Internet connectivity available at all locations
- Necessary hardware infrastructure in place
- Staff trained on system usage
- Government health registries accessible via APIs

---

3. SPECIFIC REQUIREMENTS

## 3.1 FUNCTIONAL REQUIREMENTS

### 3.1.1 Patient Management Module

FR-1.1: Patient Registration

- System shall allow front desk staff to create new patient records
- System shall capture: Name, DOB, Gender, Address, Contact
- System shall generate unique patient ID
- System shall validate mandatory fields before saving
- System shall check for duplicate registrations
- System shall allow search by Name, ID, Phone
- Performance: Registration completed in < 5 minutes

FR-1.2: Patient Medical History

- System shall maintain complete medical history
- System shall store: Previous illnesses, surgeries, allergies
- System shall record family medical history
- System shall maintain medication history
- System shall allow doctor to update history
- System shall maintain audit trail of changes

FR-1.3: Electronic Medical Records (EMR)

- System shall store complete patient EMR
- System shall link appointments to medical records
- System shall link lab reports to medical records
- System shall allow doctor access to EMR during consultation
- System shall maintain confidentiality (access control)
- System shall be searchable by date, doctor, visit type

FR-1.4: Patient Document Management

- System shall store digital copies of:

  - Insurance cards
  - ID proofs
  - Previous medical reports
  - Lab reports
  - Prescription scans
- System shall support PDF, Image file formats
- System shall maintain document versions

3.1.2 Appointment Management Module

FR-2.1: Appointment Scheduling

- System shall display available appointment slots
- System shall allow patient/staff to book appointments
- System shall consider doctor schedules and leave
- System shall display doctor specialization
- System shall allow appointment type selection (consultation, follow-up)
- System shall confirm booking via SMS/Email

FR-2.2: Appointment Reminder

- System shall send SMS 24 hours before appointment
- System shall send email reminder 24 hours before
- System shall allow patient to confirm/reschedule via SMS
- System shall allow cancellation up to 24 hours before

FR-2.3: Appointment History

- System shall maintain complete appointment history
- System shall show status (completed, missed, cancelled)
- System shall allow rescheduling of appointments
- System shall track no-show rates

3.1.3 Billing Module

FR-3.1: Invoice Generation

- System shall generate invoices automatically after consultation
- System shall include: Date, Services, Rates, Total, Tax
- System shall calculate tax based on applicable rates
- System shall apply discounts if applicable
- System shall generate unique invoice number

- System shall print and email invoice

FR-3.2: Payment Processing

- System shall accept cash, card, check, online payments
- System shall update account after payment
- System shall generate receipt
- System shall maintain payment history
- System shall generate payment reports

FR-3.3: Insurance Processing

- System shall support insurance claim submission
- System shall track insurance coverage
- System shall identify eligible procedures
- System shall manage deductibles and co-pays

3.1.4 Pharmacy Module

FR-4.1: Medicine Inventory

- System shall maintain medicine database (name, cost, quantity)
- System shall track stock levels
- System shall trigger reorder alerts (when stock < threshold)
- System shall maintain expiry dates
- System shall flag expired medicines

FR-4.2: Prescription Management

- System shall display doctor's prescriptions
- System shall dispense medicines based on prescription
- System shall track medicine dispensing
- System shall check for drug interactions
- System shall maintain medicine usage history

FR-4.3: Vendor Management

- System shall maintain vendor information
- System shall process purchase orders
- System shall track received stock
- System shall update invoice and payments

3.1.5 Lab Management Module

FR-5.1: Test Requisition

- System shall allow doctor to order lab tests
- System shall include test name, urgency level
- System shall generate sample collection instructions
- System shall display required sample type

FR-5.2: Test Execution

- System shall track sample collection and receipt
- System shall assign to lab technician
- System shall link to lab equipment if automated
- System shall record test results
- System shall validate results (within normal range check)
- System shall require verification by lab supervisor

FR-5.3: Result Reporting

- System shall generate lab reports
- System shall include patient name, test results, reference values
- System shall flag abnormal results
- System shall send results to doctor
- System shall allow patient portal access to results

## 3.2 NON-FUNCTIONAL REQUIREMENTS

### 3.2.1 Performance Requirements

NR-1.1: Response Time

- Login: < 5 seconds
- Page load: < 3 seconds
- Search results: < 2 seconds
- Report generation: < 30 seconds
- Transaction processing: < 5 seconds

NR-1.2: Throughput

- System shall handle 1000 concurrent users
- System shall process 500 transactions/minute
- System shall support 50 simultaneous file uploads

NR-1.3: Scalability

- Database scalable to 10 million patient records
- Linear performance degradation with increased load

- Horizontal scaling with additional servers

### 3.2.2 Security Requirements

NR-2.1: Authentication

- Username and password required
- Password minimum 8 characters, mixed case, numbers
- Multi-factor authentication for sensitive operations
- Session timeout after 15 minutes of inactivity
- Failed login logging (max 5 attempts before lockout)

NR-2.2: Authorization

- Role-based access control (Admin, Doctor, Staff, Patient)
- Doctor access only to own patients' records
- Staff access limited to assigned functions
- Admin full access
- Patient access to own records only

NR-2.3: Data Encryption

- All data in transit encrypted (HTTPS/TLS 1.2+)
- Sensitive data encrypted at rest (AES-256)
- Passwords hashed using bcrypt or equivalent
- Credit card information PCI-DSS compliant

NR-2.4: Audit and Compliance

- Complete audit trail of all data access
- HIPAA compliance (patient privacy)
- Data retention policies (7 years for medical records)
- Compliance reporting for regulators
- Right to be forgotten (GDPR-like requirements)

### 3.2.3 Reliability Requirements

NR-3.1: Availability

- System uptime: 99.9% (max 8.76 hours downtime/year)
- RTO (Recovery Time Objective): < 1 hour
- RPO (Recovery Point Objective): < 15 minutes
- Backup frequency: Every 6 hours

NR-3.2: Fault Tolerance

- Automatic failover to backup server
- Graceful degradation in case of failure
- Offline mode for critical functions
- Error recovery without data loss

3.2.4 Usability Requirements

NR-4.1: User Interface

- Intuitive navigation, minimal training required
- Consistent UI across modules
- Large, readable fonts (font size > 12)
- High color contrast for accessibility
- Support for multiple resolutions

NR-4.2: Accessibility

- WCAG 2.1 Level AA compliance
- Keyboard navigation support
- Screen reader compatible
- Audio alerts for important events
- Support for text enlargement

3.2.5 Maintainability Requirements

NR-5.1: Code Quality

- Modular design, <500 lines per function
- 80% code test coverage
- Well-documented APIs
- Version control of all code

NR-5.2: Deployability

- Automated deployment process
- Zero-downtime updates where possible
- Rollback capability for failed deployments
- Database migration scripts

3.2.6 Compatibility Requirements

NR-6.1: Browser Compatibility

- Chrome 90+
- Firefox 88+

- Internet Explorer 11+
- Safari 14+
- Edge 90+

NR-6.2: Database

- Microsoft SQL Server 2016+
- Oracle 12c+
- PostgreSQL 13+

NR-6.3: Operating System

- Windows Server 2016+
- Linux (Ubuntu 20.04+)
- macOS 10.14+

## 3.3 EXTERNAL INTERFACE REQUIREMENTS

### 3.3.1 User Interface

[Screen mockups and wireframes would be included]

- Login screen
- Patient registration form
- Appointment calendar
- Doctor dashboard
- Billing invoice
- Lab report view
- Pharmacy inventory view

### 3.3.2 Hardware Interface

- Network: LAN 100Mbps minimum
- Printers: Thermal for receipts, laser for reports
- Card readers: For insurance cards
- Lab equipment: Digital interfaces

### 3.3.3 Software Interface

- Database: ODBC/JDBC connections
- Report Generation: Crystal Reports
- Email: SMTP protocol
- SMS Gateway: Third-party API
- Government Health Portal: HTTPS APIs

## 3.4 DATA REQUIREMENTS

3.4.1 Logical Data Model [ERD would be included]

- Entities: Patient, Appointment, Doctor, Bill, Medicine, Test, Staff
- Relationships: Patient-Appointment, Patient-Bill, Doctor-Appointment, etc.

3.4.2 Data Dictionary

**Patient Table**:

- PatientID (PK): Unique identifier, Integer
- Name: Patient name, String(100)
- DOB: Date of birth, Date
- Gender: M/F, Char(1)
- Phone: Contact number, String(10)
- Email: Email address, String(100)
- Address: Full address, String(255)
- BloodGroup: A/B/AB/O±, String(5)
- CreatedDate: Registration date, DateTime
- ModifiedDate: Last update, DateTime

[Similar details for other entities...]

---

4. VERIFICATION AND ACCEPTANCE CRITERIA

4.1 Test Cases
[Sample test cases would be included]

- TC-1: Verify patient can be registered successfully
- TC-2: Verify duplicate patient prevention
- TC-3: Verify appointment scheduling with valid doctor
- TC-4: Verify invalid appointment time rejection
- TC-5: Verify billing calculation accuracy

4.2 Acceptance Criteria

- All functional requirements implemented and tested
- Non-functional requirements verified through performance testing
- Security testing completed
- User acceptance testing passed
- Deployment checklist completed

---

5. APPENDICES

A. Glossary of Terms
B. Use Case Diagrams
C. Data Flow Diagrams
D. Interface Mockups
E. Security Requirements Detail
F. Performance Testing Plan
G. Deployment Plan
H. Training Plan
I. User Manual Outline
J. Change Log

---

**Q18: Explain the concept of "Requirements Traceability" with a practical example. Create a sample traceability matrix.**

**Answer**: [Already covered in detail above with examples]

---

# 9. KEY FORMULAS AND DIAGRAMS

## Requirements Engineering Process Flow Diagram

START
↓
INCEPTION ← Identify stakeholders, define scope
↓
ELICITATION ← Gather requirements, techniques
↓
ELABORATION ← Create models, detail specs
↓
NEGOTIATION ← Resolve conflicts, prioritize
↓
SPECIFICATION ← Document SRS
↓
VALIDATION ← Reviews, testing
↓
MANAGEMENT ← Change control, traceability
↓
DESIGN & DEVELOPMENT ← Implementation
↓
TESTING ← Validation against requirements
↓
DEPLOYMENT
↓
END

[Iterative feedback loops possible in many phases]

## Requirement Quality Assessment Matrix

Requirement Quality = Clarity × Completeness × Consistency × Feasibility × Testability

Each factor rated 1-5:
1 = Poor
5 = Excellent

Example:
Requirement: "Generate monthly report"

- Clarity: 2 (vague)
- Completeness: 2 (missing format, content details)
- Consistency: 4 (aligns with other requirements)
- Feasibility: 5 (definitely achievable)
- Testability: 2 (unclear how to verify)

Quality Score = (2+2+4+5+2) / 5 = 3.0 / 5 (needs improvement)

Improved: "System shall generate monthly financial report in PDF format containing revenue, expenses, and profit summary by department, delivered to email by 5th of next month"

- Clarity: 4
- Completeness: 5
- Consistency: 4
- Feasibility: 5
- Testability: 4

Quality Score = (4+5+4+5+4) / 5 = 4.4 / 5 (good)

---