Requirements Engineering

From Elicitation to Validation

A Complete Study Guide

Functional Req.	Non-Functional Req.	Use Cases
SRS / SRD	Validation	Management

FSEN106 | by Resolute Femi

Renance Technology | Federal University of Technology, Akure

Table of Contents

Introduction to Requirements Engineering

What is Requirements Engineering?

Requirements engineering is the process of establishing the services that the customer requires from a software system, along with the constraints under which the system operates and is developed. It is one of the most critical phases of the software development lifecycle because mistakes made at this stage are the most costly to correct later. The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.

Requirements engineering bridges the gap between what the customer needs and what the development team builds. Without properly engineered requirements, software projects risk delivering a product that fails to meet user expectations, exceeds budget, or misses deadlines.

What is a Requirement?

A **requirement** may range from a high-level abstract statement of a service or constraint to a detailed mathematical functional specification. This duality is intentional: requirements serve a dual function as a basis for a contract bid AND as the basis for the contract itself.

Because of this dual role, requirements must be written in a way that is both open to interpretation (so stakeholders understand the general intent) AND precisely defined (so developers can implement it correctly).

User Requirements vs. System Requirements

Requirements are expressed at two levels of abstraction depending on the intended audience. User requirements are written for non-technical stakeholders, while system requirements provide detailed specifications for the development team.

User Requirements	System Requirements
Written for customers and end-users in natural language, supplemented with diagrams and tables.	Detailed technical description of what the system should do, written for developers and architects.
May include abstract descriptions of system services and operational constraints.	Defines functions, services, and operational constraints with precise, testable detail.

User Requirements	System Requirements
Focuses on WHAT the system should do at a high level, avoiding implementation details.	Specifies exactly HOW the system behaves including error handling, data formats, and protocols.
Example: "The system shall allow users to log in with their student ID."	Example: "The system shall authenticate users via SHA-256 hashed passwords stored in a PostgreSQL database using bcrypt with a salt factor of 12."

The Importance of Getting Requirements Right

Research consistently shows that requirements errors are the most expensive type of software defect. A defect found during requirements phase costs a fraction of a unit to fix. The same defect found after deployment can cost 100 times more. This cost amplification occurs because fixing a requirements error late in development requires redesigning architecture, rewriting code, redoing tests, and potentially retraining users.

- Poor requirements are a leading cause of project failure.
- Ambiguous requirements lead to different interpretations by different developers.
- Missing requirements result in features the customer expected but never received.
- Conflicting requirements cause inconsistencies that may only surface during integration testing.
- Over-specified requirements can waste development effort on unnecessary features.

1. Functional and Non-Functional Requirements

Functional Requirements

Functional requirements describe what the system should do. They declare the specific services the system should provide, how the system should react to particular inputs, and how the system should behave in particular situations. Functional requirements depend on the type of software being developed, who the users are, and the overall type of system.

Characteristics of Functional Requirements:

- **Describe services:** Each functional requirement specifies a service the system must provide, such as searching records, generating reports, or processing transactions.
- **Define system reactions:** They specify how the system responds to particular inputs or events, including error conditions and edge cases.
- **Describe behaviour:** They outline system behaviour in specific situations, such as concurrent user access or partial data input.
- **May state prohibitions:** Functional requirements may also explicitly state what the system should NOT do to prevent misuse or unsafe behaviour.
- **Must be complete:** All required services must be defined; nothing the user expects should be missing.
- **Must be consistent:** No two functional requirements should conflict or contradict each other.

Functional Requirements Example: MHC-PMS (Mental Health Care Patient Management System)

1. A user shall be able to search the appointments lists for all clinics in the system.
2. The system shall generate each day, for each clinic, a list of patients expected to attend appointments that day.
3. Each staff member using the system shall be uniquely identified by an 8-digit employee number assigned by HR.
4. The system shall allow authorised clinicians to update patient records, including diagnoses and prescriptions.
5. The system shall send automated reminder notifications to patients 24 hours before their scheduled appointment.

Non-Functional Requirements

Non-functional requirements are constraints on the services or functions offered by the system. Unlike functional requirements that describe specific behaviours, non-functional requirements describe system properties and qualities. They include timing constraints, standards, constraints on the development process, and quality attributes such as reliability, performance, maintainability, and security.

Non-functional requirements often apply to the system as a whole rather than to individual features. This means that even if every single functional requirement is perfectly implemented, the system can still fail if non-functional requirements are not met. For example, a hospital patient management system might correctly list all appointments (functional) but be so slow that staff cannot use it in time-critical situations (non-functional failure).

Critical Insight

Non-functional requirements are often MORE CRITICAL than functional requirements. If a non-functional requirement such as performance, security, or reliability is not met, the entire system may be unusable or unsafe, even if all functional requirements are perfectly satisfied.

Types of Non-Functional Requirements

Type	Description	Example
Product Requirements	Specify that the delivered product must behave in a particular way regarding execution, reliability, or memory usage.	Execution speed, reliability, memory and disk usage constraints.
Organisational Requirements	Requirements that are a consequence of organisational policies and procedures, including development standards and delivery constraints.	Process standards, implementation requirements, delivery requirements, coding standards.
External Requirements	Requirements that arise from factors external to the system and its development process, such as laws and regulations.	Interoperability with external systems, legislative compliance, ethical requirements.
Performance Req.	Statements about how fast the system must respond or how many transactions it must handle per unit of time.	The MHC-PMS shall be able to process up to 1,000 patient records per minute under normal load.
Space Req.	Statements about how much memory or disk space the software is permitted to use on the target hardware.	The system footprint on the server shall not exceed 512 MB of storage excluding the database.
Usability Req.	Requirements that define how easy the system must be for the intended target user class to operate effectively.	Medical staff shall be able to perform all primary system functions after 4 hours of training.

Type	Description	Example
Reliability Req.	Statements about how reliable the system must be, including acceptable failure rates and recovery time.	Mean time to failure shall exceed 5,000 hours. System downtime shall not exceed 4 hours per month.
Portability Req.	Requirements about how easily the system can be moved to a new hardware or software environment.	System shall run on Windows 10, Ubuntu 22.04, and macOS 14 without modification.
Security Req.	Requirements specifying the protection of the system and its data from unauthorised access, misuse, or attack.	All patient data shall be encrypted at rest using AES-256. All connections shall use TLS 1.3 or higher.
Maintainability Req.	Requirements specifying how easily the system can be modified, updated, or extended after deployment.	Any licensed developer familiar with Python shall be able to understand and modify a module within 2 hours.

Domain Requirements

Domain requirements are a distinct category derived from the application domain of the system. They reflect the characteristics, conventions, and rules of the specific domain in which the system operates. Domain requirements may introduce new functional requirements, add constraints to existing requirements, or define specific computations that must be performed using domain specific formulas or methods.

- Domain requirements may be expressed in the technical language of the application domain, which non-domain experts may not understand.
- They can be difficult for software engineers who do not have domain expertise to fully understand or correctly implement.
- If domain requirements are not satisfied, the system may be unworkable even if it is technically correct from a software perspective.
- **Example:** A railway signalling system must comply with safety interlocking rules that prevent conflicting signals. A banking system must implement interest calculations according to specific regulatory formulas.

2. The Software Requirements Document (SRD)

The Software Requirements Document (SRD), also called the Software Requirements Specification (SRS), is the official statement of what the system developers should implement. It is not a design document and should not describe how the system achieves its goals, only what those goals are. It should include both user requirements (for customers) and system requirements (for developers).

SRD vs. Design Document

The SRD specifies WHAT the system should do. A design document specifies HOW the system does it. Mixing these two concerns is a common mistake that leads to premature design commitments that may limit implementation options and cause rework when requirements change.

Who Uses the SRD?

The SRD serves as a communication bridge between all stakeholders involved in a software project. Each stakeholder group uses it differently:

Stakeholder	How They Use the SRD
System Customers	Specify requirements; read and verify the document matches their actual needs. Sign off on the SRD as part of the contract.
System Managers	Plan bids for the system, estimate project cost and timeline, and plan the development process based on scope.
System Engineers / Developers	Understand the system to be developed. Use requirements as the basis for design decisions and implementation.
System Test Engineers	Develop validation test plans and test cases directly from requirements to verify the delivered system meets specifications.
System Maintenance Engineers	Understand the system structure, the relationships between components, and the rationale behind design decisions for future modifications.
Project Management	Track progress, manage scope changes, and assess the impact of proposed changes against the baseline SRD.

IEEE Standard Structure of an SRD (IEEE Std 830)

The IEEE has defined a recommended standard structure for Software Requirements Specifications under IEEE Std 830. Following this structure ensures completeness, consistency, and that no essential components are omitted. While the exact structure may vary, the following sections are standard practice:

Section	Purpose	Key Content
Preface	Defines expected readership and document history.	Version history, reasons for changes, document scope, intended audience.
Introduction	Describes the need for the system.	System functions overview, target users, constraints, assumptions, dependencies.
Glossary	Defines all technical and domain terms.	Alphabetical list of terms and definitions. Does not assume specialised knowledge.
User Requirements Definition	Describes services provided to the end user.	Written using natural language and diagrams. High-level, accessible to non-technical stakeholders.
System Architecture	High-level overview of anticipated system structure.	Distribution of functions across modules, major components, and their relationships.
System Requirements Specification	Detailed description of all requirements.	Both functional and non-functional requirements in precise, testable form.
System Models	Graphical models of system structure and behaviour.	Data flow diagrams, entity-relationship diagrams, UML use case and sequence diagrams.
System Evolution	Anticipates future changes.	Fundamental assumptions the system rests on; planned evolutionary changes; known future requirements.
Appendices	Detailed supporting information.	Hardware specifications, database schemas, external interface descriptions, regulatory references.
Index	Navigation aids for the document.	Alphabetical index, requirements index, and cross-reference tables.

3. Requirements Specification

Requirements specification is the process of writing down the user and system requirements in a formal requirements document. Requirements must be written clearly enough to be understandable to both customers (who validate them) and system developers (who implement them). This creates a tension: the more precise the requirements, the harder they are to read and understand; the simpler they are, the more ambiguous they become.

Notations for Writing Requirements

Notation	Description	When Used
Natural Language	Requirements written as sentences in plain English, supplemented by diagrams and tables.	User requirements; high-level system requirements where accessibility matters more than precision.
Structured Natural Language	Requirements written using predefined templates with fixed fields. Reduces flexibility but improves consistency.	System requirements specifications where a standard format is required across the organisation.
Design Description Languages	Uses a language similar to a programming language but with abstract features (like pseudocode).	Rarely used for requirements. More appropriate for system design and algorithm specification.
Graphical Notations (UML)	Use-case diagrams, sequence diagrams, activity diagrams, and state charts define requirements graphically.	System requirements; interaction and behavioural requirements; where visual clarity adds value.
Mathematical Specifications	Based on formal mathematical notation like Z, VDM, or finite state machines. Completely unambiguous.	Safety-critical systems (avionics, medical devices) where ambiguity is unacceptable. Hard to validate with customers.

Guidelines for Writing Natural Language Requirements

Natural language is the most commonly used notation because it is accessible to both technical and non-technical stakeholders. However, it requires discipline to write well. The following guidelines help produce clear, unambiguous requirements:

- **Use a standard format consistently:** Every requirement in the document should follow the same format. This makes the document easier to read and reduces the chance of missing information.
- **Distinguish mandatory from desirable:** Use **SHALL** for mandatory requirements and **SHOULD** for desirable but optional ones. Avoid ambiguous words like "will" or "may."
- **Use text highlighting:** Use bold or italic to pick out key terms, actors, and system names within requirements text.
- **Avoid computer jargon in user requirements:** User requirements should be readable by non-technical customers. Reserve technical language for system requirements.
- **Include rationale:** For each requirement, include an explanation of why it is necessary. This helps developers make correct trade-off decisions and helps reviewers understand intent.
- **Avoid ambiguity:** Every requirement should have exactly one possible interpretation. Words like "fast," "reliable," "easy to use," and "appropriate" are ambiguous and must be replaced with measurable criteria.
- **Make requirements testable:** Every requirement should be verifiable. If you cannot write a test that confirms a requirement is met, the requirement is too vague.
- **Assign unique identifiers:** Every requirement should have a unique identifier (e.g., FR-001, NFR-012) to enable traceability and change management.

The Ambiguity Problem

The key challenge with natural language requirements is ambiguity. Writers and readers may interpret the same words differently. Consider: "The system shall support a large number of users." What is "large"? 10? 10,000? 1 million?

Better: "The system shall support a minimum of 500 concurrent users without degradation in response time."

One strategy is to supplement natural language with structured templates or formal notations wherever ambiguity is a concern. Another is to review each requirement by asking: "How would I test this?" If the test is unclear, the requirement is ambiguous.

4. Requirements Engineering Processes

Requirements engineering processes vary enormously between organisations and between different types of applications. A large safety-critical aerospace system requires a very different RE process than an agile web application startup. However, there are some generic activities common to all processes: requirements elicitation, requirements analysis, requirements validation, and requirements management.

These activities are often interleaved and iterative rather than strictly sequential. In practice, requirements engineering is a spiral activity where new requirements emerge during development and the process repeats. The output of RE feeds directly into system design and validation planning.

The Four Core RE Activities

1

Requirements Elicitation

Working with stakeholders to discover the requirements of the system. Also called requirements discovery or requirements gathering. This includes identifying all relevant stakeholders and understanding their goals, needs, constraints, and domain.

2

Requirements Analysis

Analysing the collected requirements to check for consistency, completeness, and conflicts. Priorities are identified and trade-offs are resolved. Conflicting requirements from different stakeholders must be negotiated and agreed upon.

3

Requirements Validation

Checking that the requirements document accurately captures what customers and other stakeholders actually want the system to do. Validation asks: "Are we building the right system?" (as opposed to verification which asks: "Are we building the system right?")

4

Requirements Management

Managing changes to requirements as new requirements emerge during design, development, and testing phases. Requirements always change; management ensures those changes are tracked, assessed, and incorporated in a controlled way.

In practice, these four activities are not performed sequentially. Requirements engineering is an iterative spiral process: as understanding develops, earlier activities must be revisited. New stakeholders may be identified during analysis; validation may reveal missing requirements that require new elicitation; changes during development require management. This iterative nature is a feature, not a flaw.

5. Requirements Elicitation and Analysis

Requirements elicitation and analysis involves discovering requirements by working directly with all relevant stakeholders. It is one of the most challenging aspects of software engineering because: stakeholders often do not know precisely what they want; they express requirements in domain-specific terms that developers may not understand; different stakeholders have conflicting requirements; organisational and political factors influence what requirements are stated; and requirements change throughout the process as new stakeholders are identified or business conditions change.

Stages of the Elicitation and Analysis Process

Stage	Activity	Output
Requirements Discovery	Interact with stakeholders to uncover domain requirements, business goals, system context, user goals, and environmental constraints.	Initial draft requirements: a raw, unorganised list of stakeholder needs.
Requirements Classification and Organisation	The unstructured collection of requirements is grouped into coherent clusters based on related functionality, subsystems, or requirement types.	Organised requirement set with logical structure and preliminary categorisation.
Requirements Prioritisation and Negotiation	Prioritise requirements and resolve conflicts through structured negotiation with all relevant stakeholders. Some requirements may be deferred to future releases.	Agreed, prioritised requirements with conflict resolutions documented and rationale recorded.
Requirements Documentation	Requirements are formally documented in a structured requirements document, to be refined iteratively as the process continues.	Draft requirements document suitable for review, validation, and baseline agreement.

Elicitation Techniques

Different elicitation techniques are appropriate in different contexts. Skilled requirements engineers choose and combine techniques based on the type of system, the availability of stakeholders, and the nature of the requirements being sought.

Interviews	Formal or informal questioning of stakeholders. Two types: closed interviews use pre-defined questions for systematic information gathering; open interviews allow free discussion to explore topics in depth.	Good for understanding high-level goals, priorities, and known requirements. May miss tacit knowledge that stakeholders do not realise they hold. Works best when combined with other techniques.
Ethnography	An observational technique where an analyst embeds in the working environment and observes work as it actually happens, rather than as stakeholders describe it.	Very effective for discovering implicit requirements related to existing work practices that stakeholders take for granted and never mention. Also reveals social and political constraints on the system.
Scenarios and Stories	Real-life narrative examples of how a system would be used in specific situations. Scenarios describe interactions step by step and can be used to develop formal use cases.	Helps stakeholders concretise vague needs by thinking through specific examples. Can reveal missing requirements and edge cases. Works well for validating that developers understand user needs.
Use Cases (UML)	A formal UML technique that identifies actors (users or external systems) and describes the interactions (use cases) between actors and the system being developed.	Provides a structured framework for discovering all possible system interactions. Use case diagrams give an overview; use case descriptions provide detailed interaction steps, preconditions, and postconditions.
Prototyping	A working model of part of the system is built quickly using low-fidelity tools (paper prototypes, wireframes, or throwaway code) to help stakeholders understand and clarify requirements.	Particularly effective when stakeholders find it difficult to articulate requirements in the abstract. Seeing a prototype often surfaces requirements that stakeholders did not know they had.
Workshops and Focus Groups	Structured group meetings involving multiple stakeholders to elicit requirements collaboratively. A skilled facilitator manages the discussion to ensure all voices are heard.	Useful for resolving conflicting viewpoints across stakeholder groups and reaching consensus quickly. Faster than conducting many individual interviews and promotes shared understanding.
Questionnaires and Surveys	Written questions distributed to a large number of stakeholders when individual interviews are impractical. Can be structured (multiple choice) or unstructured (open-ended).	Useful for gathering statistical data about user priorities across large populations. Limited in depth and cannot follow up on interesting answers without additional contact.

Problems and Challenges in Requirements Elicitation

- **Stakeholders do not know what they want:** Users often describe what they currently do rather than what they need. They may not be able to articulate requirements until they see a prototype.
- **Domain language barriers:** Requirements are expressed in stakeholder domain language. Developers may misunderstand technical terminology used in the application domain.
- **Conflicting requirements:** Different stakeholders have different priorities and conflicting views on what the system should do. Negotiation and compromise are necessary.
- **Organisational and political factors:** Political considerations within an organisation may influence what requirements are stated or suppressed. Some stakeholders may attempt to use requirements to further personal agendas.
- **Requirements volatility:** Requirements change during the analysis process as the business environment evolves, new stakeholders are identified, or the implications of existing requirements become clearer.
- **Unstated assumptions:** Stakeholders may have implicit requirements that they assume are obvious and will be included automatically. These must be surfaced and made explicit.
- **Incomplete stakeholder availability:** Key stakeholders may not have time for extensive interviews or workshops, leading to gaps in the requirements.

6. Requirements Validation

Requirements validation is the process of checking that the requirements in the SRD define the system that the customer actually wants. It is critically important because errors in requirements documents are the most expensive type of software defect. Finding a requirements error after the system has been delivered may cost 100 times more to fix than finding the same error during the requirements phase, because by then the system has been designed, coded, tested, and deployed based on that erroneous requirement.

The Cost of Late Requirements Errors

The cost of fixing a requirements error **after delivery** can be up to **100 times** the cost of fixing it during the requirements phase. This makes requirements validation one of the highest-ROI activities in the entire software development process. Time and effort invested in thorough validation pays dividends throughout the rest of the project.

What Validation Checks For

Requirements validation systematically checks the requirements document for a set of critical quality properties. Every requirement should satisfy all of the following checks:

Validity	Does the system provide the functions that best support customer needs? Are all stated needs actually needed? Does the requirement represent what the customer genuinely wants?	Ask: "Will satisfying this requirement actually deliver value to the user?" Validate with customers directly.
Consistency	Are there any conflicting requirements in the document? No two requirements should contradict each other or impose incompatible constraints on the system.	Check: Can all requirements be satisfied simultaneously? Use a requirements matrix to identify conflicts.
Completeness	Are all required functions and constraints included in the requirements document? Nothing the customer expects should be missing from the specification.	Ask stakeholders: "Is anything missing?" Use checklists based on the problem domain to identify gaps.
Realism	Can the requirements be implemented within budget, on schedule, and using available technology? Unrealistic requirements must be renegotiated.	Validate with developers and architects. Cost unrealistic requirements; consider phasing delivery.

Verifiability	Can each requirement be tested? Can a test be written that will objectively confirm the requirement has been met? Requirements that cannot be tested are worthless.	For each requirement, draft a test case. If a test cannot be written, the requirement is ambiguous or unmeasurable.

Validation Techniques

Requirements Reviews	Manual analysis of the requirements document by a team of reviewers. Both customer and contractor staff should be involved. The review team systematically checks for errors, inconsistencies, omissions, and conflicts. Reviews can be formal (structured inspection with roles) or informal (walkthrough).	Most cost-effective validation technique when done early. A review involving domain experts, developers, testers, and customers together can surface different classes of problems simultaneously.
Prototyping	An executable model of the system is demonstrated to end-users and customers who interact with it and verify that it meets their needs. The prototype makes abstract requirements concrete and tangible.	Particularly effective for discovering missing requirements and validating usability requirements. May reveal that stakeholders imagined the system differently from how developers understood the requirements.
Test-Case Generation	Requirements should be testable. The process of generating test cases for each requirement serves as a validation technique: if a test case cannot be written for a requirement, the requirement may be unmeasurable, ambiguous, or incomplete.	Every requirement should correspond to at least one test case. Test-case generation during requirements validation creates a test plan that can be used directly when testing the final system.
Automated Analysis	Requirements management tools (such as IBM DOORS) can automatically check for common defects such as duplicate requirements, missing attributes, and broken cross-references between related requirements.	Useful for large documents with hundreds or thousands of requirements where manual review of every cross-reference would be impractical.

7. Requirements Management

Requirements management is the process of managing changing requirements during the requirements engineering process and throughout the system development lifecycle. Requirements inevitably change during development: new requirements emerge as stakeholders better understand what they need; existing requirements need modification as the business environment evolves; and technical constraints discovered during implementation force trade-offs that require requirements revision.

Requirements management does not try to prevent change (which is impossible) but instead ensures that changes are tracked, that their impact on the rest of the system is properly assessed, that the cost and schedule impact of changes is understood, and that approved changes are incorporated in a controlled and traceable way.

Why Requirements Change

- **Changing business environment:** The business context and technical environment of the system changes after installation. What was a correct requirement when specified may no longer be appropriate.
- **Divided stakeholders:** The people who pay for the system are not always the same people who use it. Conflicting priorities between sponsors and users emerge as development progresses.
- **Diverse communities:** Large systems are usually built for diverse communities of users with different and sometimes incompatible requirements. Reaching consensus takes time and iteration.
- **Regulatory changes:** New legislation or updated regulations may require system modifications after requirements have been agreed and development has started.
- **Technology changes:** New platforms, APIs, or infrastructure changes may make originally specified implementations impractical or create new possibilities.
- **Market changes:** Competitive pressures or shifts in the market may require new features or changed priorities that affect the requirements baseline.

Requirements Management Planning

Effective requirements management requires establishing a set of policies and processes before development begins. The requirements management plan defines how requirements will be identified, stored, tracked, and changed:

Component	Description
Requirements Identification	Each requirement is assigned a unique identifier (e.g., FR-001, NFR-023) so it can be cross-referenced with other requirements, design components, test cases, and code modules in traceability assessments.

Component	Description
Change Management Process	A defined set of activities to assess the impact and cost of proposed changes to requirements. The process decides if a proposed change should be accepted, deferred, or rejected based on cost-benefit analysis.
Traceability Policies	Policies that define the relationships between each requirement and between requirements and system design. Pre-requirements traceability links requirements to their source. Post-requirements traceability links them to the design components and tests that implement them.
Tool Support	Requirements management for large systems involves large volumes of information. Specialist tools such as IBM DOORS, Jira, and Azure DevOps store requirements, manage change requests, and track traceability relationships automatically.
Baseline Management	At key project milestones, the requirements set is formally "baselined." Changes to baselined requirements must go through the change management process, providing stability and preventing scope creep.

The Change Management Process

When a proposed change to requirements arrives, a structured process should be applied to decide whether the change is worth implementing and how to incorporate it:

Step 1	**Problem Analysis and Change Specification** The proposed change is analysed to determine whether it represents a valid problem or enhancement request. The problem or desired improvement is formally specified so that all stakeholders share a common understanding of what change is being proposed.
Step 2	**Change Analysis and Costing** The effect of the proposed change is assessed using traceability information. All requirements, design components, code modules, and test cases that would be affected are identified. The cost of implementing the change is estimated in terms of effort, time, and risk.

Step 3	**Change Acceptance or Rejection** Based on the analysis and costing, stakeholders decide whether the change should be accepted, deferred to a future release, or rejected. This decision is formally documented with the rationale.
Step 4	**Change Implementation** If accepted, the requirements document and system design are modified to reflect the approved change. The system is retested in all affected areas. Version control records the change history.

Key Terms and Definitions

Requirement	A statement of a service the system should provide or a constraint on how it should operate or be implemented. Requirements range from high-level abstractions to detailed mathematical specifications.
User Requirement	High-level statements of system services and constraints written for customers in natural language supplemented with diagrams. Focuses on WHAT the system should do from a user perspective.
System Requirement	A detailed description of what the system should do, written as a precise specification for developers and architects. Defines the exact functions, services, and operational constraints.
Functional Requirement	Describes what the system should do: the services, responses to inputs, and behaviour in specific situations. Should be complete (all services defined) and consistent (no conflicts).
Non-Functional Requirement	Constraints on services or system functions including timing, reliability, security, performance, and standards. Often more critical than functional requirements as failure may render the entire system unusable.
Domain Requirement	Requirements derived from the application domain that reflect characteristics and conventions of that domain. May be expressed in domain language and can be difficult for software engineers outside the domain to understand.
SRD / SRS	Software Requirements Document / Software Requirements Specification. The official statement of what developers should implement. Specifies WHAT, not HOW. Serves all project stakeholders.
Requirements Elicitation	The process of working with stakeholders to discover and gather requirements. Also called requirements discovery or requirements gathering. Involves interviews, observation, scenarios, and prototyping.
Requirements Validation	The process of checking that the requirements document correctly captures what the customer wants. Checks for validity, consistency, completeness, realism, and verifiability.
Requirements Management	Managing changing requirements throughout the development lifecycle. Ensures changes are tracked, impact-assessed, and incorporated in a controlled way using change management processes.

Use Case	A UML technique that describes a system interaction between an actor and the system. Use case diagrams show the scope of the system; use case descriptions document detailed interaction flows.
Actor	A role played by a person or external system that interacts with the system under development. Actors are identified in use case modelling.
Ethnography	An observational elicitation technique where an analyst embeds in the working environment to observe actual work practices and discover implicit requirements that stakeholders cannot articulate.
Traceability	The ability to link each requirement to its source and to the system components, design elements, and tests that implement it. Enables impact analysis when requirements change.
IEEE 830	IEEE standard defining the recommended practice for Software Requirements Specifications. Defines the standard structure for SRDs to ensure completeness and consistency.
Baseline	A formally agreed version of the requirements document at a specific point in time. Changes to a baseline must go through a formal change management process.
Stakeholder	Any individual, group, or organisation that has an interest in the system or is affected by its development or operation. Requirements must address the needs of all relevant stakeholders.

Summary and Quick Review

1	Requirements express what customers need from a system and the constraints under which it must operate. They range from high-level abstractions to precise technical specifications.
2	**Functional requirements** describe WHAT the system should DO (services, reactions to inputs, behaviours). **Non-functional requirements** describe HOW WELL it should perform (performance, security, reliability).
3	Non-functional requirements are often MORE CRITICAL than functional requirements. Failure to meet a non-functional requirement (e.g., response time) can render the entire system unusable even if all functional requirements are met.
4	The **Software Requirements Document (SRD/SRS)** is the official statement of what should be built. It specifies WHAT, not HOW. It serves all project stakeholders from customers to maintenance engineers.
5	Requirements should use **SHALL** for mandatory requirements and **SHOULD** for desirable ones. Every requirement must be complete, consistent, unambiguous, and testable.
6	The four core RE activities are: **Elicitation** (discover requirements), **Analysis** (check for consistency and conflicts), **Validation** (verify correctness), and **Management** (handle changes).
7	Elicitation techniques include: interviews, ethnography, scenarios, use cases, prototyping, and workshops. Each technique is suited to different situations and types of requirements.
8	Validation checks for five properties: **Validity** (right functions?), **Consistency** (no conflicts?), **Completeness** (nothing missing?), **Realism** (feasible?), and **Verifiability** (testable?).
9	Requirements change throughout development. **Requirements management** tracks and controls changes through identification, change management process, traceability policies, and tool support.
10	The change management process has three steps: Problem Analysis and Specification, Change Analysis and Costing, and Change Implementation. All changes to a baseline must go through this process.
11	Errors found in requirements **after delivery** cost up to **100 times more** to fix than errors found during requirements engineering. Validation is therefore one of the highest-ROI activities in software development.

12 The IEEE Std 830 defines the recommended structure for SRDs, including sections for Preface, Introduction, Glossary, User Requirements, System Architecture, System Requirements, System Models, System Evolution, Appendices, and Index.

Exam Preparation Note

This document covers Requirements Engineering as studied in FSEN106, based on Sommerville's *Software Engineering (10th ed.)*.

For exam preparation, focus especially on: **types of requirements** (functional, non-functional, domain), **the SRD structure** (IEEE Std 830), **the four RE activities** (elicitation, analysis, validation, management), **validation checks** (VCCV: Validity, Consistency, Completeness, Verifiability), **elicitation techniques**, and **change management steps**.