Bachelor of Computer Science (BSc in CS)

CS1303: Introduction to Software Engineering

Requirement Engineering

Prof. K. P. Hewagamage

Learning Outcomes

- Be able to identify the requirement engineering process
- Be able to describe why requirements are presented as user and system requirements
- Be able to write verifiable functional and non-functional requirements
- Be able to describe requirement definition activities of elicitation, analysis, and specification
- Be able to identify requirements validation and management processes

Sub-topics for the discussion

- Functional and non-functional requirements
- Requirements engineering processes
- Requirements elicitation
- Requirements specification
- Requirements validation
- Requirements change

What is a Requirement?

- A **requirement** is a condition or capability that must be satisfied.
  - A condition or capability needed by a user to achieve a goal. *(User need)*
  - A condition or capability required of the system to meet a standard or constraint. *(System rule)*
  - A documented version of either of the above.

Example - Requirement

User: "I want to track my fitness.“

System: "The app shall record daily step counts and display charts.“

🔁 Agile Viewpoint: User Stories

- As a [user], I want [feature] so that [benefit]
  As a user, I want to track my daily steps so that I can stay healthy.

Requirement Presentation

- User Need ➡️ Requirement ➡️ System Capability

Requirement Type	Perspective	Focus
User Requirement	From user view	What problem to solve
System Requirement	From system view	What the system must do

- Misunderstood or vague requirements are a **leading cause of software project failure**

  Example *“The system shall allow users to reset their password via email.”*

Lifecycle of a Requirement

1. **Need Identified**
   A stakeholder expresses a goal, problem, or desire.
2. **Problem Understood**
   The gap between the current state and the desired outcome is analyzed.
3. **Requirement Defined**
   The need is formalized into a clear, actionable requirement.
4. **Solution Proposed**
   A system feature or change is designed to fulfill the requirement.
5. **Dependencies Explored**
   One requirement may trigger or rely on others.
6. **Conflicts Managed**
   Requirements may conflict due to technical, resource, or stakeholder constraints.
7. **Prioritization & Trade-offs**
   Some requirements may be deferred, dropped, or negotiated.

Example – Changes to Requirement

- **Need**: Patients want to book appointments online.
- **Requirement**: The system shall provide a calendar for booking.
- **Conflict**: Doctors want to limit booking to weekdays only.
- **Trade-off**: Appointments allowed only on weekdays, 8–5 PM.

[Need] → [Problem] → [Requirement] → [Solution]

↓

[Triggers Other Requirements]

↓

[Conflicts Identified]

↓

[Resolution & Trade-offs]

In agile:

- Needs are identified incrementally.
- Requirements evolve as user stories.
- Solutions are refined through iterations and feedback.

What is Software Design?

- Software design is the **creative and technical process** of defining the architecture, components, interfaces, and data of a system to fulfill specified requirements.
- A design is not the final product but a **blueprint** for how the system will work.
- Multiple design options may fulfill the same requirement, each with different trade-offs (cost, performance, maintainability).
- A requirement describes *what* is needed.
  A design defines *how* it will be implemented.

Goal of Design: To find the most **usable, efficient, and maintainable** way to satisfy stakeholder needs.

Requirement Flow

- [Requirement] → [Design Options] → [Chosen Design] → [Implementation]

Design Challenges:

- Balancing conflicting requirements
- Adapting to evolving needs
- Working within constraints (time, budget, tech)

In Agile:

- Design evolves iteratively.
- Emphasis on *"just enough design"* up front.
- Refactoring is part of ongoing design work.

Agile and Requirements Engineering

- **In Agile, change is expected:**
  Agile methods embrace changing requirements—even late in development.
- **Heavy documentation is discouraged.**
  Detailed requirements specs become obsolete quickly. Agile prefers **lightweight, just-in-time documentation**
- **Incremental and iterative elaboration**
  Requirements are refined through collaboration, sprint planning, backlogs, and feedback.

Limitations of Agile Requirements

- Less suitable for safety-critical or regulated systems.
- Difficult to coordinate across large, distributed teams without formal specs.

Agile vs Plan-driven in RE

Topic	Agile Approach	Plan-Driven Approach
Requirement Stability	Assumes change	Assumes stability
Requirement Format	User stories, epics	SRS, use cases
Timing	Just-in-time	Upfront
Traceability	Lightweight (tool-based)	Full trace matrix
Validation	Through working software	Formal reviews, documents

Sprint 1 → Sprint 2 → Sprint 3 → ...
↑ Requirements evolve incrementally

RE in Agile: **INVEST Principle** – For Writing Good User Stories

Letter	Principle	Meaning
I	Independent	Stories should be self-contained. No dependency on others.
N	Negotiable	Not fixed contracts—open to discussion and refinement.
V	Valuable	Each story must deliver value to the user or customer.
E	Estimable	Teams should be able to estimate effort/time.
S	Small	Should be small enough to complete within a sprint.
T	Testable	Clear enough that tests can verify its completion.

Example:
As a student, I want to download my grade report so that I can keep a personal record.

RE in Agile: Backlog Grooming / Refinement

- A recurring meeting where the product owner and team review items in the **product backlog** to ensure they're ready for future sprints.

Activities include:

- Splitting large stories (epics) into smaller ones
- Clarifying acceptance criteria
- Estimating effort (story points)
- Re-prioritizing based on new insights

Outcome: A “Ready” backlog — stories that meet the **Definition of Ready** (clear, prioritized, and estimated).

RE in Agile: Acceptance Criteria

- Conditions that must be met for a user story to be considered complete from a **user or stakeholder** point of view.

Purpose:

- Defines boundaries of a story
- Ensures shared understanding
- Guides testing and validation

Example: For the story: *"As a user, I want to reset my password"*

Acceptance Criteria:

- User receives a reset email
- Password must be at least 8 characters
- Password reset link expires in 24 hours

RE in Agile: Definition of Done (DoD)

- A shared agreement within the team on what it means for a task or user story to be **fully complete**.

Typical DoD Items:

- Code is written properly
- Code is reviewed and merged
- Tests (unit/integration) pass
- Documentation is updated
- Product owner has reviewed the story

Ensures **quality and completeness**, not just "it works on my machine".

3 Dimension of Requirements

Requirement Type	Question Answered	Description	Example
Functional	**What**	Describes *what* the system should do: the behavior, features, and services.	"The system shall allow users to reset their password."
Non-Functional	**How**	Describes *how well* the system should perform under certain conditions.	"The system shall load the dashboard within 2 seconds."
Domain	**Why**	Reflects *why* certain requirements exist, based on domain-specific rules or context.	"Patient data must be retained for 10 years (per legal requirement)."

Functional Requirements: The “What” of the System

Functional requirements define **what the system should do** — the features, behaviors, and services that enable the system to fulfill its purpose.

Characteristics:

- Describe **inputs**, **outputs**, and **processing logic**
- Can be expressed as **user tasks** or **system services**
- Include **business rules**, **validations**, and **workflow logic**
- May also state what the system **must not do**

Example FR

Scenario	Functional Requirement
1. Online Banking	The system shall allow users to transfer funds.
2. eCommerce System	The system shall calculate discounts at checkout.
3. Student Management System	The system shall generate student transcripts.

FR in Agile?

In agile methods:

- Functional requirements are often captured as **user stories**.
- User stories focus on **functionality from the user's viewpoint**.

Example user story:

"As a customer, I want to track my orders so that I know when to expect delivery."

This implies functional features like:

→ Order retrieval

→ Status tracking

→ Notification triggers

Non-Functional Requirements: The “How” Dimension

- Non-functional requirements specify **how well the system performs** its functions. They describe **quality attributes**, performance levels, and technical constraints that guide the system's design and operation.

Key Characteristics:

- Focus on **system-wide behavior**, not individual features
- Often drive **architectural decisions**
- **Measure quality**, not functionality

Categories of Non-Functional Requirements:

Type	Description	Example
Performance	Speed, response time, throughput	"System must respond within 2 seconds."
Reliability & Availability	Uptime, fault tolerance, recovery	"99.9% uptime over a 30-day period."
Security	Access control, data protection, encryption	"Only admins can modify user permissions."
Usability	User interface intuitiveness, accessibility	"Users shall complete sign-up in under 3 steps."
Maintainability	Ease of debugging, updating, and extending the system	"Code modules must be independently deployable."
Interoperability	Ability to integrate with other systems	"System shall support HL7 messaging."

NF

- While some non-functional requirements are **triggered by domain rules** (e.g., security due to GDPR), the **domain requirement itself** is a separate concern — the **“Why”** — not part of the non-functional “How”.

Agile View:

- Captured in the **Definition of Done**
- Sometimes modeled as **"quality stories"**

As a user, I want the dashboard to load in under 2 seconds so that I can use it efficiently.

Metrics for specifying non-functional requirements

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

Domain Requirements: The “Why” Dimension

- Domain requirements are **constraints or rules** that arise from the **operating environment, industry practices, or external regulations**. They explain **why** certain system behaviors or qualities are necessary — often beyond stakeholder preferences.

Key Characteristics:

- **Originate outside** the system: from the business domain, legal frameworks, safety standards, etc.
- May **influence or trigger** functional or non-functional requirements
- Often **non-negotiable** (especially regulatory or legal)
- **Specific to a domain**: healthcare, education, finance, transportation, etc.

Examples

Domain	Domain Requirement	Resulting Requirements
Healthcare	Patient records must be stored for 10 years	Functional: Record archivalNon-functional: Storage reliability
Banking	All transactions must comply with AML laws	Functional: Fraud detectionNon-functional: Security logging
Education	Grades must follow government-defined scale	Functional: GPA calculation Non-functional: Auditability
Transportation	Must comply with ISO 26262 for functional safety	Functional: Alert generationNon-functional: Safety-critical architecture

Requirement Imprecision: The Danger of Ambiguity

Imprecise or ambiguous requirements are those that **lack clarity or specificity**, leading to **multiple interpretations** by developers, testers, or stakeholders.

Why It Matters:

- Leads to **miscommunication** between stakeholders and developers
- Causes **defects, rework, and user dissatisfaction**
- Can result in **non-compliance** with user expectations or legal obligations

Example – Imprecision Requirement

- The system shall allow users to search for patients.”

Stakeholder	Interpretation
User	Search across **all clinics** for patient name
Developer	Search within **a selected clinic only**, as chosen by user

- The system shall allow users to search for a patient by name across **all clinics** and **all scheduled appointments**, without requiring the user to specify a clinic.
- “The system shall allow users to search for a patient by name **within the currently selected clinic only**.

Requirement Completeness and Consistency

Quality Dimension	Definition
Completeness	All required system features, behaviors, and constraints are **fully described**. Nothing essential is missing.
Consistency	No **conflicting**, **ambiguous**, or **contradictory** requirements exist within the document.

While **completeness and consistency** are ideal goals, they are **difficult to achieve fully** in complex, evolving systems.

Issue Type	Example
Incomplete	No mention of error handling or boundary conditions in data input features.
Inconsistent	One section says “login required for all users”; another says “guests allowed.”

What is Requirements Engineering?

- **Requirements Engineering (RE)** is the **systematic and disciplined process** of identifying, analyzing, documenting, validating, and managing the requirements for a software or system project.

It focuses on understanding:

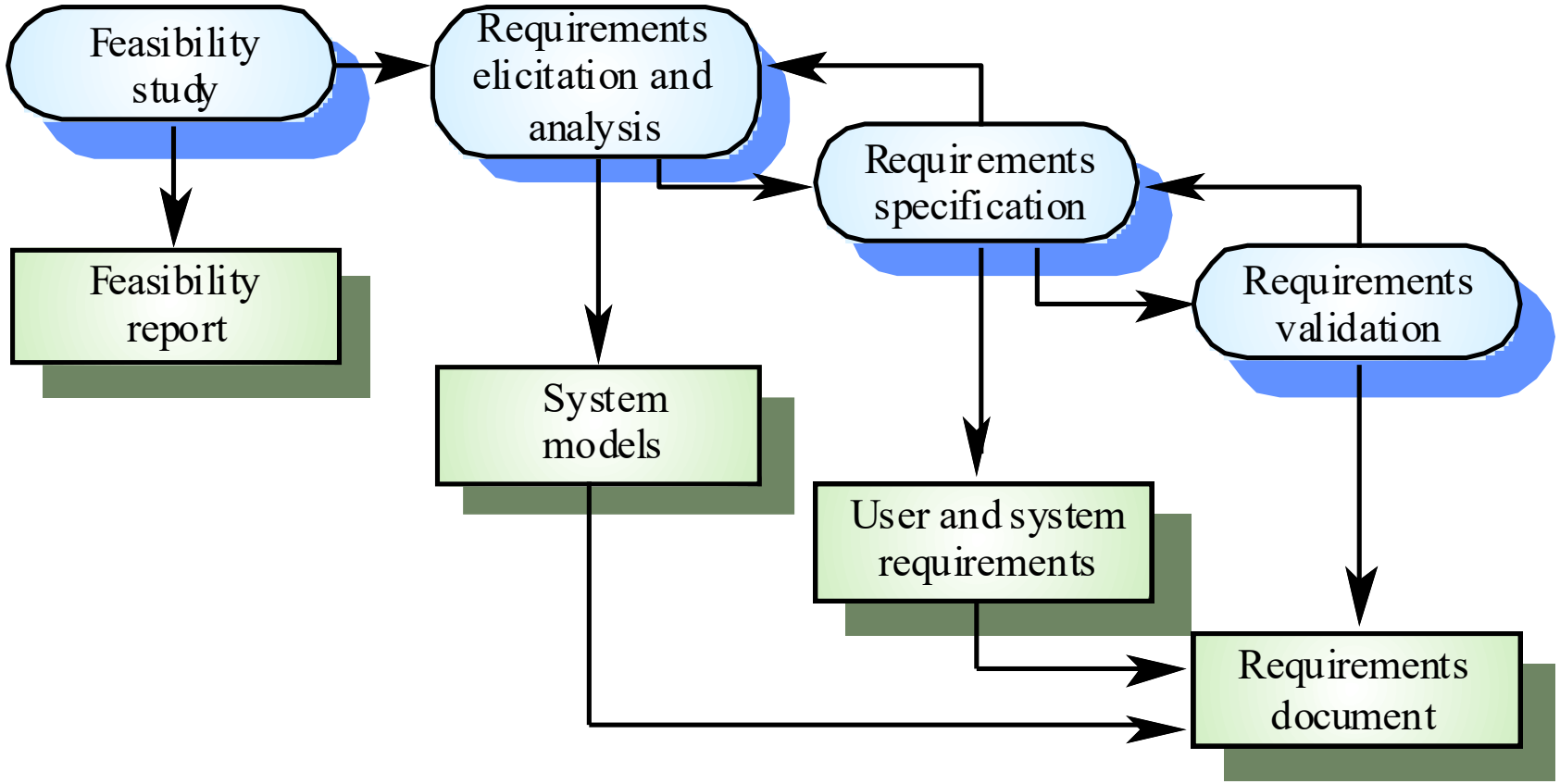
- **What** the system should do (functional requirements)
- **How well** it should do it (non-functional requirements)
- **Why** certain behaviors are necessary (domain requirements)

Why is it called “Engineering”?

The term **"engineering"** emphasizes that RE is **not just collecting or analyzing needs**, but applying **structured, repeatable, and rigorous methods** to design high-quality requirement solutions.

Engineering Quality	In RE This Means…
Systematic Process	Follows clear steps: elicitation → analysis → spec → validation
Rigor	Avoids ambiguity, ensures traceability and completeness
Problem-Solving	Resolves stakeholder conflicts, works within constraints
Tool/Method Use	Uses models (e.g., UML), templates, formal methods
Iterative Nature	Adapts to evolving systems and stakeholder expectations

The Requirements Engineering Process



Feasibility Study in Requirements Engineering

- A **feasibility study** is a short, focused assessment that determines **whether it is realistic and worthwhile** to proceed with the proposed software system.

Objectives:

- Align with **organizational goals**
- Ensure **technical and economic viability**
- Anticipate **risks and constraints**

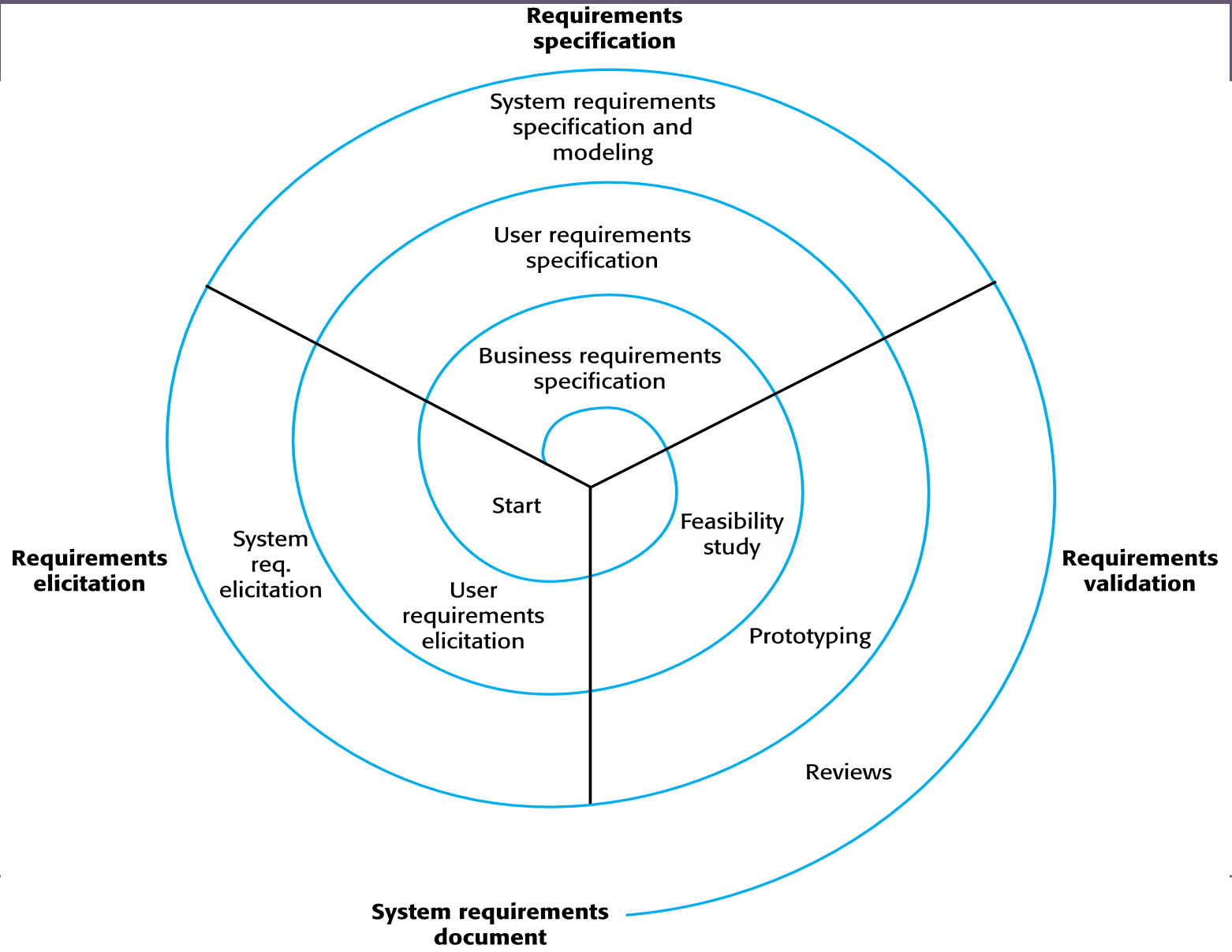
Checks Performed During a Feasibility Study:

Dimension	Question Asked
Organizational	Does it contribute to strategic goals?
Technical	Can it be built with available tech and skills?
Economic	Is it affordable within budget and expected ROI?
Integration	Will it work with existing systems and processes?

Common Types of Feasibility

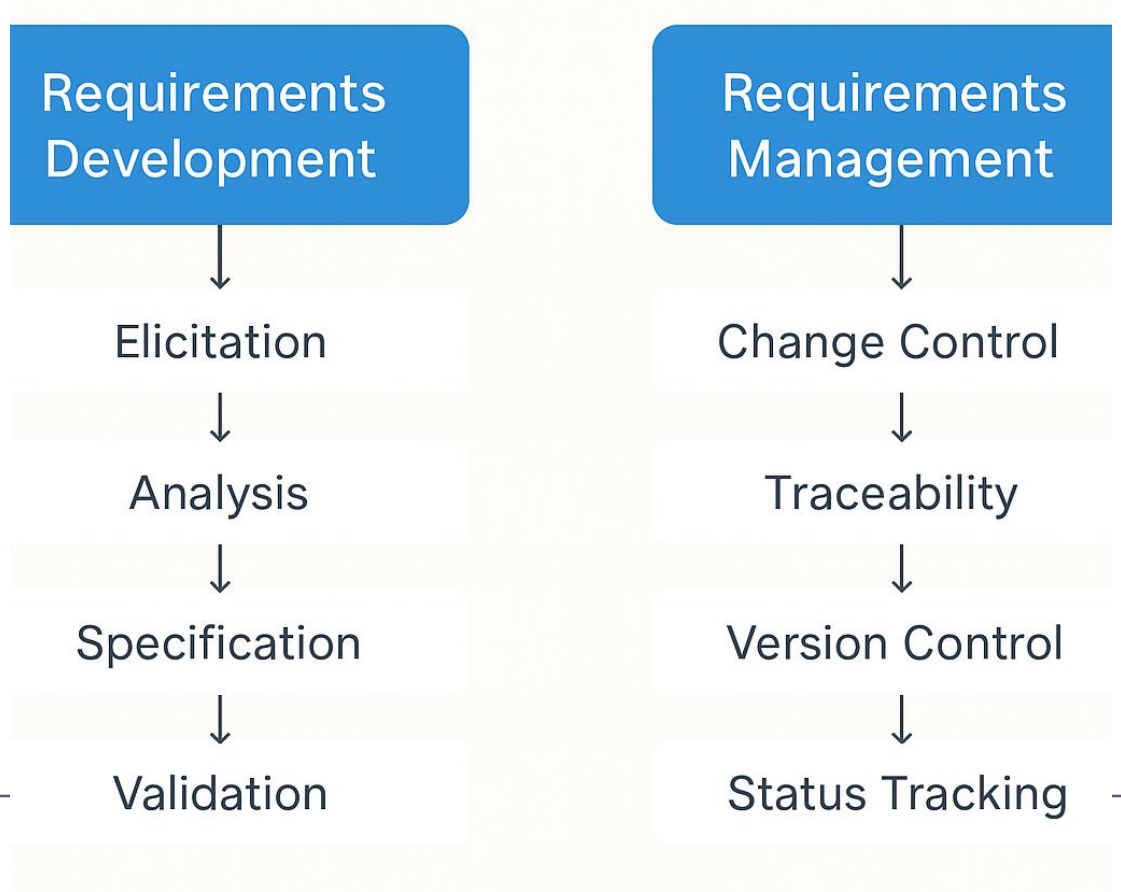
Type	Description
Technical	Assess if current technology supports the solution
Economic	Analyze costs vs. expected benefits (cost-based / time-based)
Legal	Compliance with laws, licenses, data protection, etc.
Operational	Can users, processes, and environment support the solution?
Schedule	Can the system be built within required timeframes?
Resource	Are personnel, infrastructure, and tools available?
Cultural	Will the system align with user and organizational culture?
Market/Real Estate	(For physical systems) Is the market location viable?
Financial	Will the funding and financial flows sustain development?

A spiral view of the requirements engineering process



Stages of Requirements Engineering

The Requirements Engineering (RE) process is broadly divided into **two main stages**:



1. Requirements Definition / Development

Goal: To **discover, analyze, document, and validate** the needs of users and stakeholders.

Key Activities:

- **Elicitation** – Gathering needs through interviews, workshops, observations
- **Analysis** – Resolving conflicts, prioritizing, modeling
- **Specification** – Writing clear, testable requirements
- **Validation** – Ensuring correctness, completeness, consistency

2. Requirements Management

Goal: To **handle requirement changes** and maintain clarity, consistency, and traceability throughout the project lifecycle

Key Activities:

- Change Control Management
- Requirements Traceability
- Version Control
- Impact Analysis
- Status Tracking
- Stakeholder Communication

Requirement Definition/Development

1. REQUIREMENT ELICITATION

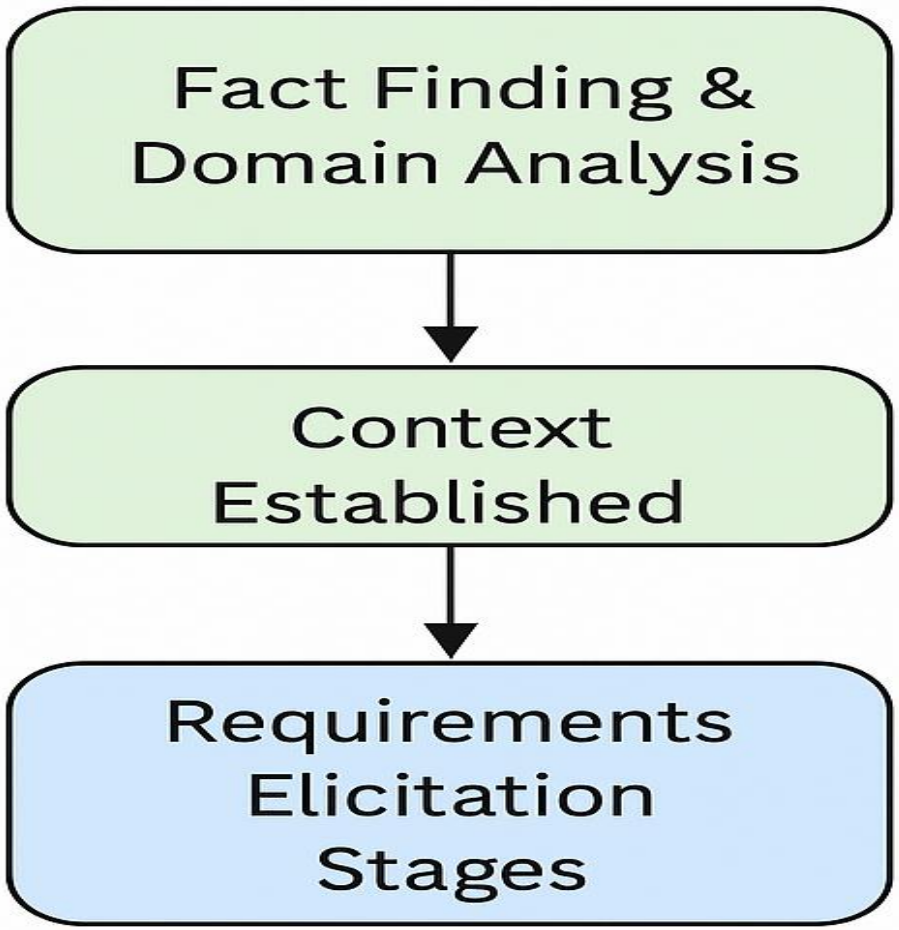
Requirements Elicitation

- Requirements elicitation is a communication-intensive process where system engineers collaborate with stakeholders to uncover functional and non-functional needs.

4 key elicitation activities :

1. Requirements Discovery
2. Requirements Classification and Organization
3. Requirements Prioritization and Negotiation
4. Requirements Specification

Requirements Elicitation



- provides context
- establishes boundari
- identifies stakeholde

- discovery
- classification
- prioritization
- specification

Fact Finding in Requirements Engineering

Elicitation without prior Fact Finding is like exploring without a map.

Fact finding is the process of **systematically collecting information** about the **organization**, **stakeholders**, and **current processes** to define the **problem context** and **system scope**.

Purpose:

- Understand the existing system and environment
- Identify problems, needs, and opportunities
- Establish the boundaries of the new system
- Support requirement elicitation with grounded insights

Activities in Fact Finding

Key Activities:

- Identify stakeholders and key users across levels
- Interview analysts with domain expertise
- Analyze existing systems (if any)
- Study current workflows, tasks, and procedures
- Investigate domain models and technologies
- Assess system constraints (cost, technical, policy)
- Conduct site visits, document reviews, and observations

Outputs of Fact Finding

Deliverables

- A clear **problem context** description/statement
- Defined **system scope** (boundaries and interfaces)
- List of **business objectives** and goals
- Overview of **existing processes** and stakeholders
- Preliminary identification of system constraints and risks

If everyone involved agrees upfront to the scope of the system, the instability of requirements can be minimized

Domain Analysis in Requirements Engineering

Domain analysis is the process of **studying the problem domain** to understand its concepts, rules, processes, and constraints.

It helps software engineers gain deep knowledge of **how the business works**, which is essential for deriving valid and useful requirements.

Objectives of Domain Analysis

- Understand the **environment** in which the system will operate
- Identify **domain concepts**, terms, and relationships
- Recognize **domain-specific constraints** (e.g., laws, standards, best practices)
- Discover **reusable knowledge** from similar systems or previous projects
- Bridge the knowledge gap between stakeholders and developers

Benefits of Domain Analysis

- Speeds up requirement elicitation
- Helps in **anticipating future needs and changes**
- Supports the development of a **better, more aligned system**
- Reduces miscommunication between stakeholders and engineers
- Encourages reuse of domain models and components

Identifying the Problem and Establishing the Scope

Identifying the Problem

- A problem can be expressed as:
  - A **difficulty** faced by users or customers in achieving their goals, or
  - An **opportunity** to enhance current operations (e.g., improve productivity, reduce cost).
- The solution often involves the development of a new or modified software system.

Writing an Effective Problem Statement

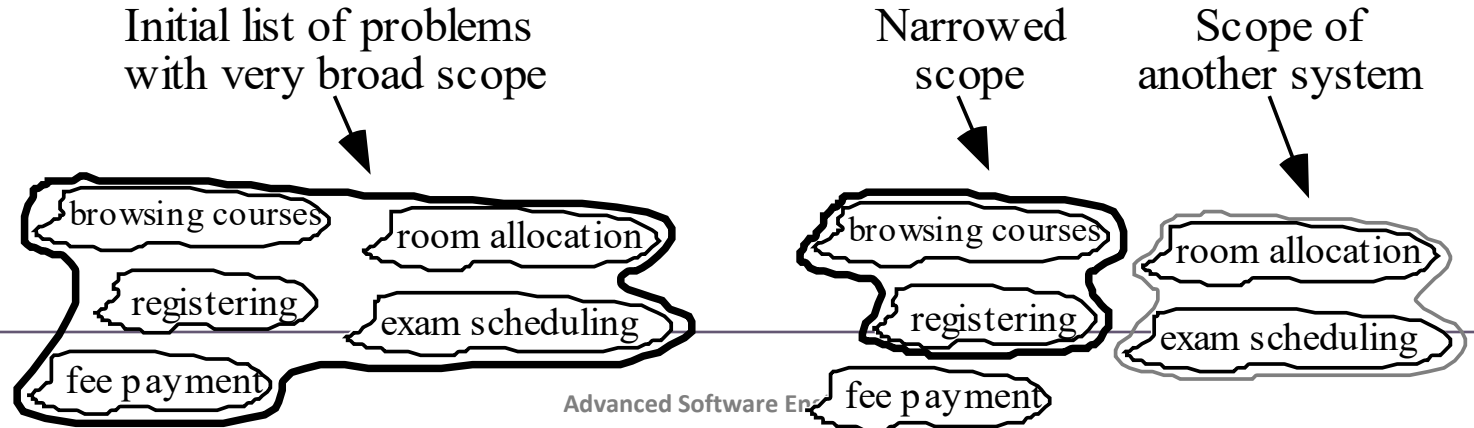
- Be specific and avoid vague language.
- Identify **key stakeholders** and their pain points.
- Include **measurable impacts** or expected benefits.
- Align with **business goals** or customer objectives.

Defining the Scope

- Clearly define **what is included** and **what is excluded** from the system.
- A good problem statement:
  - Is **short and focused**
  - Captures the **core issue or opportunity**
  - Helps prevent **requirement instability** during later stages.

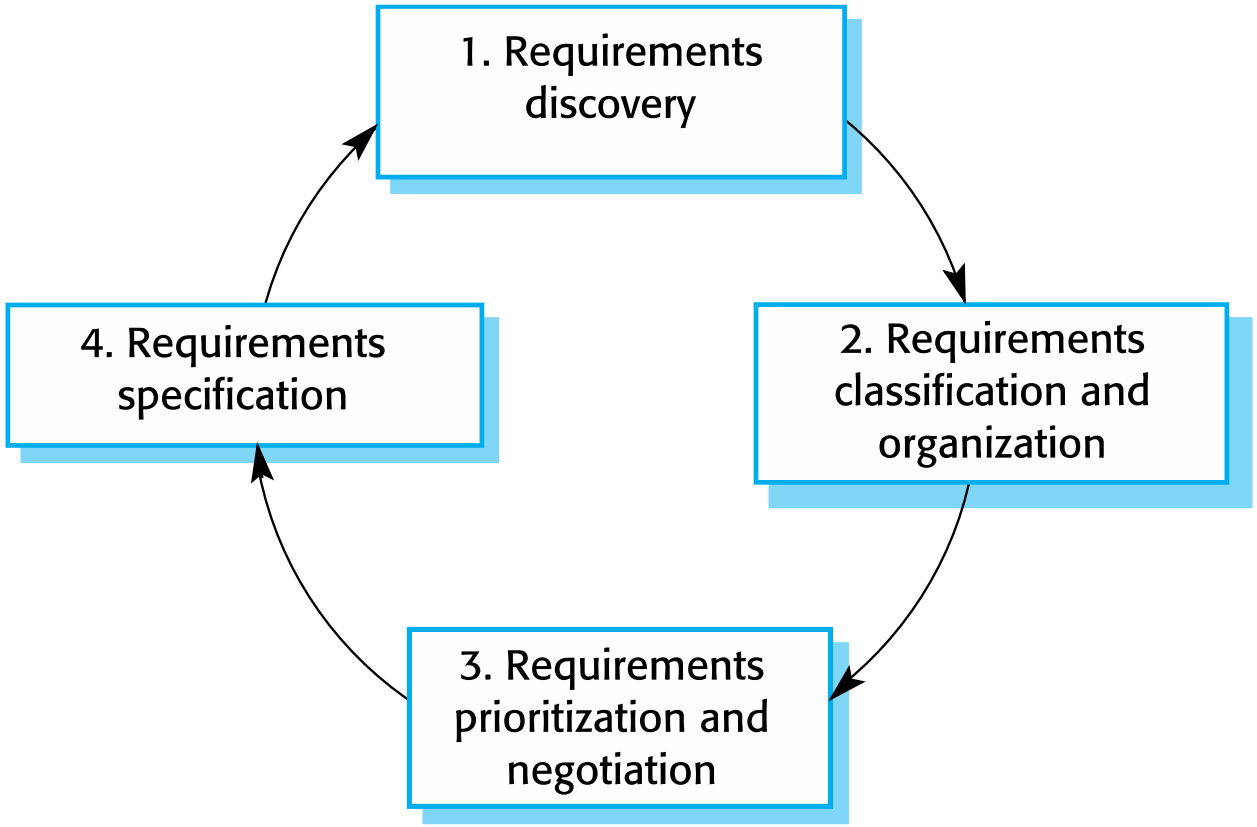
Defining the Scope

- Narrow the *scope* by defining a more precise problem
  - List all the things you might imagine the system doing
    - Exclude some of these things if too broad
    - Determine high-level goals if too narrow
- Example: A university registration system



REQUIREMENT ELICITATION

The Stages of Requirements Elicitation – Activities



What is Requirement Discovery?

- **sub-activity of requirements elicitation** and serves as the foundation for further analysis and specification.
- Requirement discovery is the process of identifying system requirements by engaging with stakeholders to understand their needs, goals, and constraints.
- “what is needed” from multiple stakeholder perspectives.

Common Problems in Requirement Discovery

- Stakeholders are unclear about their real needs.
- Requirements are expressed in inconsistent or vague terms.
- Conflicts arise between different stakeholders.
- Organizational politics and power structures may influence inputs.
- Requirements may evolve or shift as discovery progresses.

Importance of Requirement Discovery

- Ensures stakeholder expectations are properly understood.
- Helps uncover **implicit needs** that may not be directly stated.
- Facilitates early **alignment** between users and developers
- Reduces risk of **requirement gaps** or **misinterpretations**.

Two-Step Discovery Process

Wish List Creation

- Gather initial expectations from different stakeholder groups.
- Often informal or open-ended (what users hope to have).

Wish List Classification

- Organize discovered items into:
  - Functional requirements
  - Non-functional requirements
  - Environmental and technical constraints
  - Design constraints

Techniques Used in Requirement Discovery

Technique	Purpose
Interviews	Gain insights from individual or group stakeholders
Observation	Understand how users interact with current systems
Workshops	Facilitate group discussion and consensus on needs
Brainstorming	Generate and explore ideas in a collaborative setting
Prototyping	Use models or mock-ups to stimulate feedback
Document Analysis	Review existing documentation for relevant system or domain information
Surveys/Questionnaires	Gather information from a broad audience in a structured manner

Classification of Discovery Techniques

- **Conversational Methods**: Interviews, workshops, group discussions
- **Observational Methods**: Direct or indirect observation, shadowing
- **Analytical Methods**: Review of reports, system documents
- **Quantitative vs. Qualitative**: Questionnaires vs open-ended interviews

Guidelines for Effective Discovery

- Avoid asking “what do you want?”—instead ask “what do you do?”
- Use open-ended prompts to reveal workflows, frustrations, and expectations
- Involve a **diverse set of stakeholders** (end users, managers, technical staff)
- Be aware of **biases** and **preconceived solutions**
- Document “To Be Determined” (TBD) areas for follow-up

Filtering and Evaluating Discovered Requirements

- To ensure the **relevance, clarity, and feasibility** of each gathered requirement before proceeding to formal classification.

Key Activities:

- Ask "**Why do you need X?**" to uncover rationale.
- Convert "How to" solutions into "What is needed" (avoid premature design).
- Eliminate conflicting, duplicate, or unjustified requirements.
- Record rationale for traceability and future decisions.
- Conduct **feasibility checks**, cost-benefit assessments, and risk evaluations.

Requirement Classification

To **organize the evaluated requirements** into meaningful categories for further analysis, communication, and design.

Classification Types:

1. By Nature:

Functional, Non-Functional, Domain

2. By Source:

User, System, Regulatory / Legal Requirements

3. By Priority:

Essential, Must-have, Desirable, Should-have, Optional, Nice-to-have

4. By Constraints:

Technical, Environmental, Budgetary or Time-related

Outcome: A well-structured set of requirements ready for negotiation and specification.

Requirements Prioritization - Important

Not all requirements can be implemented due to limitations in **budget**, **time**, **technical feasibility**, or **strategic alignment**.

Prioritization helps in:

- Deciding **what to implement first**.
- Identifying **critical vs. optional features**.
- Managing **stakeholder expectations**.
- Planning **release cycles** effectively.

Criteria Used for Prioritizing Requirements

- Benefit to the User
  - Does it solve a user's major pain point?
  - What is the consequence of not having this requirement?
- Ease of Deployment
  - How easily can the requirement be adopted?
  - Includes factors like training needs, user resistance, hardware/software readiness.
- Ease of Development
  - How technically feasible is it?
  - Includes clarity of the requirement, developer skills available, tools and environment needed.
- Independence
  - Can the requirement be implemented without depending on others?
  - If dependent, priority might be lower unless the dependencies are resolved first.
- Cost and Risk Factors
  - High-cost/low-benefit requirements might be deprioritized.
  - Risk of failure or uncertainty may reduce priority.

MoSCoW Prioritization Technique

A widely used method to classify requirements based on stakeholder input:

Category	Description
Must	Essential. Without this, the system will fail or be incomplete.
Should	Important but not vital. Can be delayed or worked around temporarily.
Could	Desirable but not necessary. Often cosmetic or nice-to-have features.
Won’t (this time)	Agreed not to be implemented in the current release but may be reconsidered later.

Requirements Elicitation – Consolidation of Requirements

Purpose of Consolidation:

- Consolidation is the process of organizing and refining all the gathered and prioritized requirements into a coherent and consistent set.
- It ensures that the collected information aligns with the original business goals identified during the **fact finding** stage.

Why Consolidation is Essential:

- It prepares the requirements for specification by eliminating overlaps, resolving redundancies, and identifying gaps.
- This step validates that the refined requirements still represent the views and needs of all stakeholder groups.

Consolidation of Requirements

Key Objectives:

- Align all requirements with project goals and fact-finding outcomes.
- Cross-check for completeness, consistency, and non-conflicting requirements.
- Ensure that every requirement has a clear source and rationale.
- Confirm that **all stakeholder views are represented**, especially when conflicting opinions were resolved in earlier stages.

"An essential aspect of consolidation is to make sure that the resulting set of requirements reflects the collective agreement of all key stakeholders."

Consolidation Checklist

Checklist During Consolidation:

- Are there any duplicate requirements?
- Are conflicting requirements resolved or documented?
- Does each requirement link back to a stakeholder or use case?
- Are requirements grouped logically (e.g., by functionality, module, user type)?
- Are low-priority requirements clearly separated or labeled?
- Is the output ready to be passed to the specification phase?

Requirements Elicitation – Preliminary Specification

To document and organize the gathered requirements in a structured form that can later be refined during the requirements development phase.

Key Points:

- This is **not** the final SRS document, but a **draft-level structuring** of the elicited requirements.
- The aim is to **translate raw, informal stakeholder inputs** into clearly stated *user* and *system-level* requirements.

Types of Requirements Structured:

- Helps identify any **unclear, conflicting, or incomplete requirements**
- Supports **prioritization, validation, and consolidation** that follow
- Establishes a foundation for **formal specification** in the next phase

User Requirements

- High-level needs expressed by users or customers
- Captured in simple, non-technical language
- Focuses on what the user expects the system to do

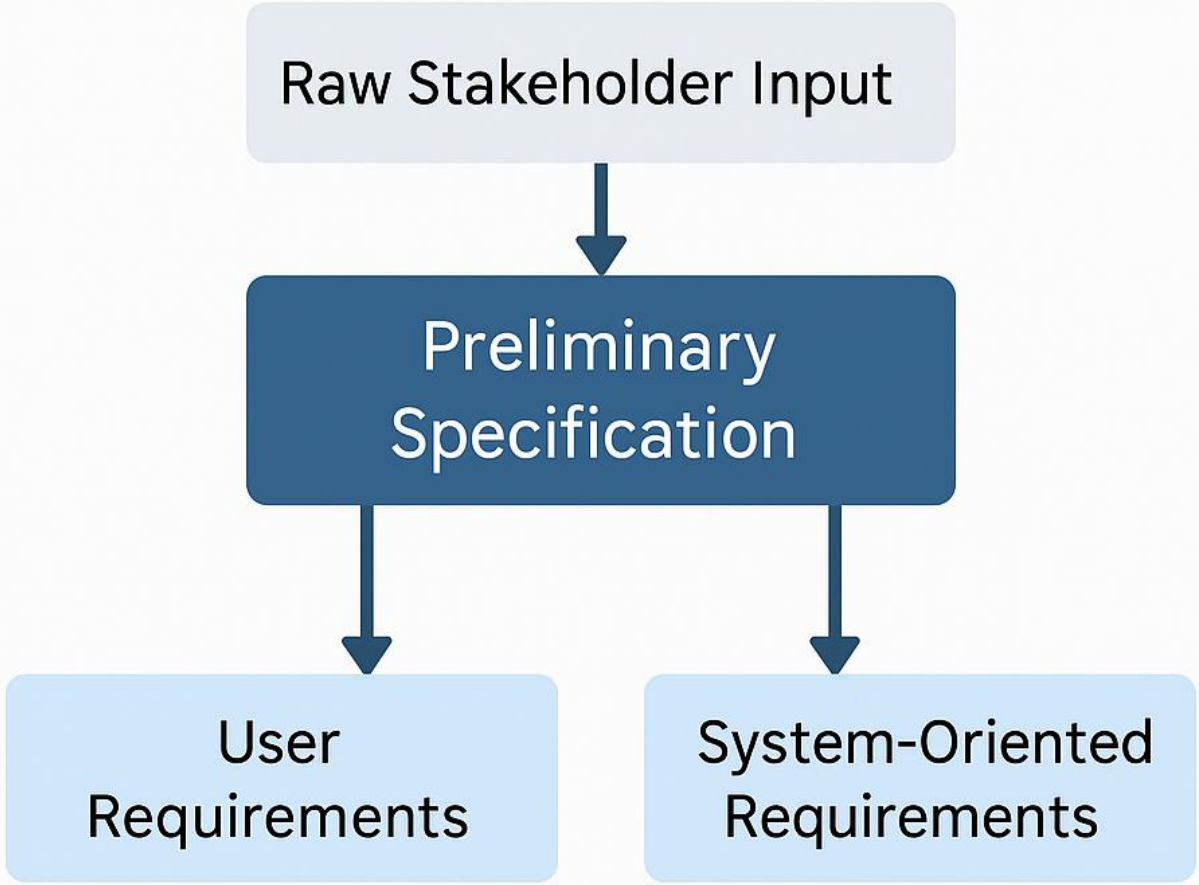
System-Oriented Requirements

- Technical interpretation of user needs
- May include constraints, data formats, performance metrics, and interfaces
- Structured to assist later design activities

Ways of writing a system requirements specification

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

Requirements Elicitation – Preliminary Specification



Common Challenges in Requirements Elicitation

- Unclear Needs
  - Stakeholders often **don’t know what they really want** or cannot articulate it clearly.
- Vague or Personal Language
  - Requirements are expressed using **individual perspectives or domain jargon**.
- Conflicting Views
  - Different stakeholders may propose **inconsistent or opposing requirements**.
- Organizational & Political Influences
  - Internal power structures and politics can **bias or distort requirements**.

Common Challenges in Requirements Elicitation

Evolving Requirements

Requirements may **change over time** due to:

- New stakeholders entering the project,
- Changes in the business environment or regulations.

REQUIREMENT ANALYSIS AND MODELING

Requirements Analysis & Modeling

To refine, organize, and analyze the elicited requirements to ensure clarity, consistency, completeness, and relevance before formal specification.

- Detect inconsistencies, omissions, ambiguities, and conflicting requirements.
- Validate that requirements align with stakeholder goals and system constraints.
- Establish clear relationships among requirements.
- Serve as a foundation for:
  - System Design
  - System Validation
  - Requirement Specification

Modeling in Requirements Analysis

- Helps visualize and structure complex requirements.
- Reveals gaps in elicitation.
- Assists in stakeholder communication and validation.

Common Modeling Techniques:

- **UML Models**:
  - Use Case Diagrams (functional context)
  - Class Diagrams (structure)
  - Activity Diagrams (workflow)
- **Data Flow Diagrams (DFD)** – Data processing logic
- **Entity Relationship Diagrams (ERD)** – Data structure logic
- **System Context Diagram** – System boundaries and external interactions

Requirements Classification

Classify requirements into meaningful categories to improve understanding and management:

- **Functional Requirements** – Describe system behavior and functionalities
- **Non-Functional Requirements** – Define system quality attributes (e.g., performance, security)
- **Design Constraints** – Technology, standards, or regulatory limits
- **Environmental Constraints** – Physical, hardware, or network conditions

Note: Requirements classification often overlaps with evaluation and prioritization.

Requirements Prototyping (Optional Support Tool)

Purpose:

- Clarify unclear requirements
- Gain early user feedback
- Reduce risk of misunderstanding

b. Types of Prototypes:

- **Throwaway Prototypes** – Used for clarification and discarded after
- **Evolutionary Prototypes** – Gradually evolved into final system

c. Best Practices:

- Clearly define scope: what is included and what is not
- Use scripts to guide evaluation
- Collect and document user feedback systematically

d. When to Avoid:

- When rapid delivery is prioritized over clarity
- When users confuse prototype with final system

Output of Analysis and Modeling

- Validated and classified requirements
- Conceptual models representing the system scope and requirements
- Prototype feedback (if used)
- Identified gaps for further elicitation
- Ready-to-refine input for Requirement Specification phase

Requirements Documentation

- Requirements documentation captures and communicates the outcome of the elicitation and analysis stages.

Key Goals:

- Serve as a reference for all stakeholders
- Guide development, testing, and validation
- Formally define what the system must do (not how)

Importance of Requirements Documentation

- Sets clear expectations for users and clients
- Guides developers and designers on what to build
- Supports testers in designing test cases
- Helps trainers and maintenance teams understand system behavior
- Provides legal and contractual reference (especially in client projects)

Software Requirements Specification (SRS)

An SRS is a formal document that describes:

- Functional requirements
- Non-functional requirements
- Constraints and assumptions

Produced as a result of:

- Requirements elicitation
- Requirements analysis

About SRS

A well-written SRS should be:

- **Clear** – understandable by all stakeholders
- **Complete** – covers all known requirements
- **Consistent** – no conflicting requirements
- **Verifiable** – each requirement can be tested
- **Modifiable** – structured to allow future updates
- **Traceable** – each requirement is linked to a business need

Formats

- Natural Language Statements
- Structured formats (e.g., numbered lists, use case tables)
- **Graphical Models**: Use Case Diagrams, DFDs, ERDs, Activity Diagrams
- **Formal specification languages** (used in critical systems)

Characteristic	Description
Complete	No missing or implied requirements
Clear	Single interpretation possible
Consistent	No contradictions
Feasible	Implementable within constraints
Traceable	Mapped backward to needs, forward to design/test
Testable	Can be verified by inspection/test
Modifiable	Easy to update
Readable	Easy to follow and navigate

Requ

Characteristic	Explanation
Cohesive	The requirement addresses one and only one thing.
Complete	The requirement is fully stated in one place with no missing information.
Consistent	The requirement does not contradict any other requirement and is fully consistent with all authoritative external documentation.
Non-Conjugated (Atomic)	The requirement is *atomic*, i.e., it does not contain conjunctions. E.g., "The postal code field must validate American *and* Canadian postal codes" should be written as two separate requirements: (1) "The postal code field must validate American postal codes" and (2) "The postal code field must validate Canadian postal codes".
Traceable	The requirement meets all or part of a business need as stated by stakeholders and authoritatively documented.

Requ

Characteristic	Explanation
Current	The requirement has not been made obsolete by the passage of time.
Feasible	The requirement can be implemented within the constraints of the project.
Unambiguous	The requirement is concisely stated without recourse to technical jargon, acronyms (unless defined elsewhere in the Requirements document). It expresses objective facts, not subjective opinions.
Mandatory	The requirement represents a stakeholder-defined characteristic, the absence of which will result in a deficiency. An optional requirement is a contradiction in terms.
Verifiable	The implementation of the requirement can be determined through one of four possible methods: inspection, demonstration, test or analysis.

SRS Document Structure (Simplified IEEE Template)

Section	Purpose
1. Introduction	States the **purpose**, **scope**, **intended audience**, **definitions**, **references**, and **document overview**. Sets the context and objectives.
2. Overall Description	Describes the **system context**, **product perspective**, **user characteristics**, **assumptions**, and **constraints**. Helps understand the environment in which the system will operate.
3. User Requirements	Describes what the **user expects** the system to do, in **natural language**, use cases, or scenarios. Should be understandable to non-technical stakeholders.
4. System Requirements	Presents the **detailed functional and non-functional requirements** of the system, including performance, reliability, and security requirements. Written for developers and technical stakeholders.

SRS Document Structure (Simplified IEEE Template)

Section	Purpose
5. Models and Diagrams	Includes **UML diagrams**, **DFDs**, **ERDs**, **System Context Diagram**, and others that illustrate the system structure and behavior.
6. SRS Characteristics Compliance	(Optional but recommended) Summarizes how the document meets qualities such as **completeness, clarity, consistency, modifiability, traceability, testability**. This helps in SRS reviews.
7. Appendices	Provides **glossary**, **references**, **external documents**, **version history**, and any supporting information.
8. Index	(Optional) Helps in quick navigation, especially in long documents. Can include diagrams, functions, or keywords.

Questions?

- How the SRS Supports the Software Lifecycle?
- Common Pitfalls in Requirements Documentation
- Common Problems with SRS?

Common Problems with SRS

Problem	Explanation
Mixing design and requirements	"How" vs. "What" confusion
Vague terms	“Quick”, “User-friendly”, “Etc.”
Unverifiable or non-testable requirements	Cannot be validated
Over-specification	Adds unnecessary constraints
Missing/assumed requirements	Leads to scope creep
Inconsistent use of terms	Misinterpretation risk

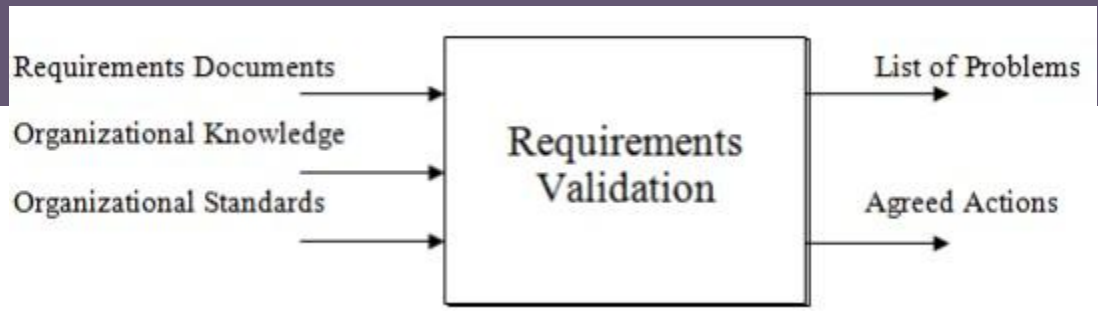
Requirements Validation & Review

Ensure the documented requirements are what the customer actually wants.

Validation Questions (5C Check):

- **Correct** – Do they reflect real needs?
- **Complete** – Are all necessary requirements included?
- **Consistent** – No contradictions?
- **Capable** – Technically and financially feasible?
- **Checkable** – Can be verified/tested?

Validation Techniques:



- Walkthroughs / Reviews (formal or informal)
- Prototyping
- Testing-based validation
- Model-based validation
- Viewpoint-based (multiple stakeholder perspectives)

Requirement Reviews

Types of Reviews:

- **Continuous Review**: Ongoing quality checks during documentation
- **Phase-End Review**: Conducted before sign-off and baseline

Key Principles:

- Include all stakeholders
- Record and resolve every issue
- Use structured review templates/checklists
- Finalize via formal change control post-baseline

Requirements Sign Off

The act of sign-off should be treated as a milestone in the project where the requirements are “baselined” and that future changes to the requirements can be made through a well defined, controlled change management process.

- Meaningless Formality – Many customers think of this as a meaningless
- some customer representatives are afraid to sign-off, as they believe that developers will stonewall any future changes with statements like “you should have told us before we signed off”

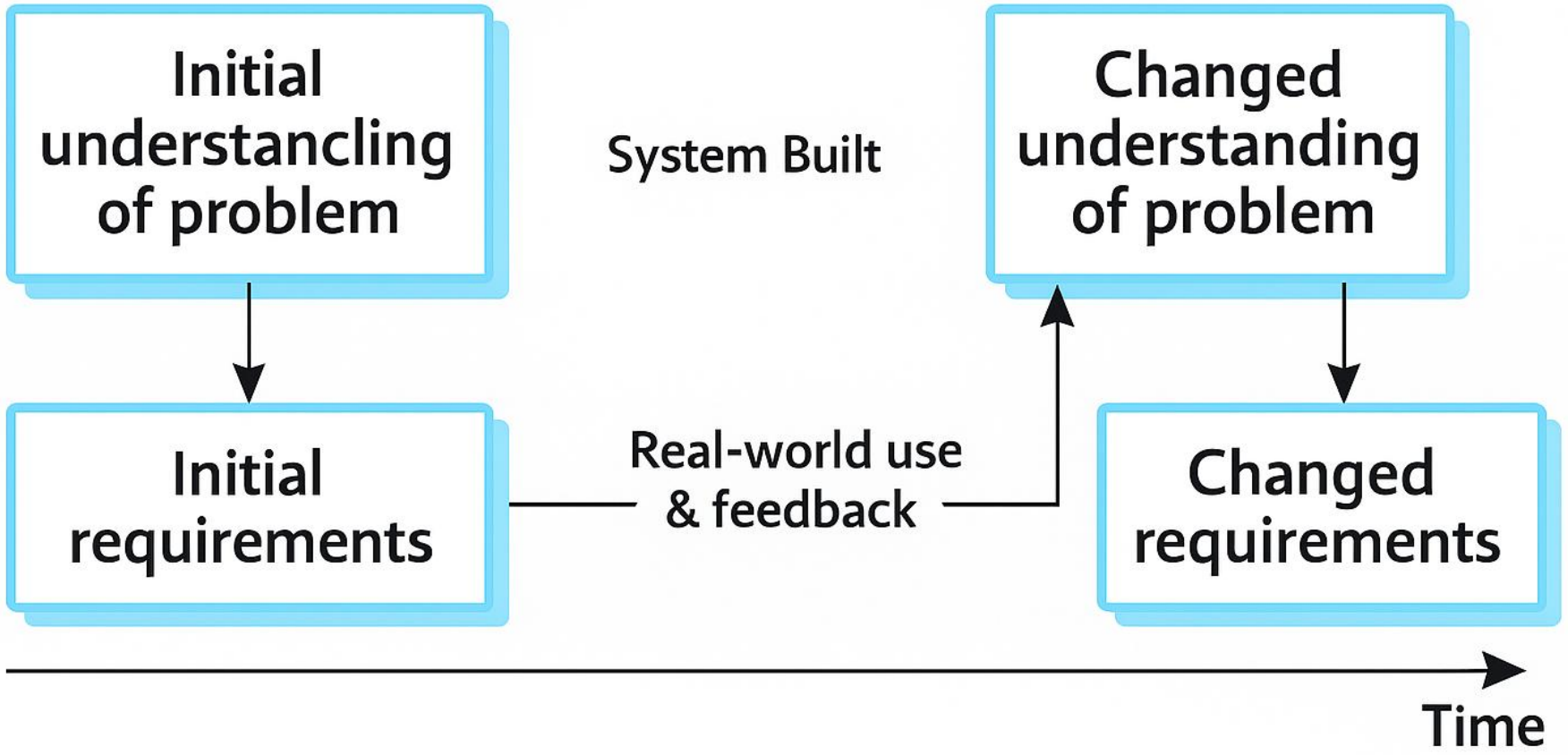
Requirements management planning/decisions

- *Requirements identification:* Each requirement must be uniquely identified so that it can be cross-referenced with other requirements.
- *A change management process:* This is the set of activities that assess the impact and cost of changes.
- *Traceability policies:* These policies define the relationships between each requirement and between the requirements and the system design that should be recorded.
- *Tool support:* Tools that may be used range from specialist requirements management systems to spreadsheets and simple database systems.

Why Requirements Change?

- 📉 **Evolving Business Needs**: Business goals, regulations, and processes change over time—so must the system.
- 👨‍💼 **Stakeholder Diversity**: Users and funders often differ. Their needs, expectations, and priorities can conflict.
- 🌐 **Technology Shifts**: New platforms, integration needs, and constraints emerge after initial deployment.
- 🔄 **Understanding Deepens Over Time**: What we think we need at the start often differs from what's needed in practice.

Requirements Evolution



Effective requirements management helps teams adapt gracefully—without losing control.

Requirements Management (RM)

RM ensures requirements are agreed upon (baselined) and any changes are tracked and controlled systematically.

<u>It is a continuous process that:</u>

- Responds to emerging needs during development.
- Establishes formal mechanisms to evaluate and approve changes.
- Communicates effectively with all stakeholders.

Two Pillars of RM:

- 🔁 Change Management
- 🔗 Requirements Traceability

Two Pillars/Aspects of RM

Aspect	Goal
1. Change Management	Manage, review, and authorize changes
2. Requirements Traceability	Ensure every requirement is linked throughout lifecycle

Definition Requirements Management

Requirements Management is the disciplined process of documenting, tracking, and controlling changes to system requirements throughout the project lifecycle.

Key Features:

- Starts after *requirements baseline* is established.
- Continues throughout development and maintenance.
- Ensures all stakeholders are aligned and informed.

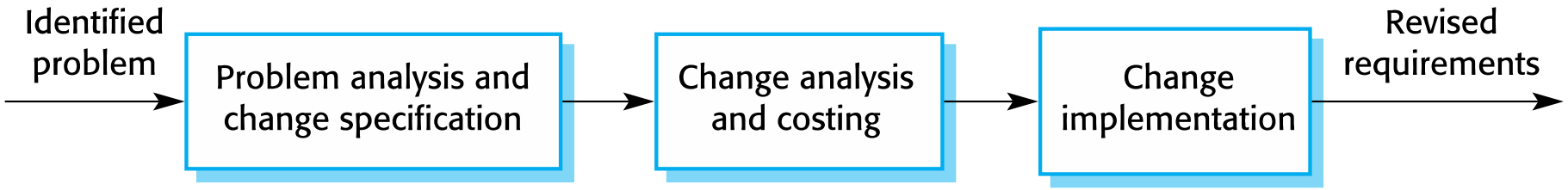
Key RM Activities

1. Requirements Identification
   - Assign a unique ID for each requirement.
2. Change Control
   - Formal procedure for proposing, evaluating, and approving changes.
3. Traceability
   - Linking each requirement to design, implementation, and tests.
4. Impact Analysis
   - Evaluating the cost, risk, and benefit of each proposed change.
5. Tool Support
   - Use of tools like Jira, Jama, Azure DevOps, or IBM DOORS.

RM Planning & Decisions

Activity	Description
Change Management Process	Define how requests are evaluated and approved
Traceability Policies	Specify how requirements link to downstream artifacts
Tool Integration	Select appropriate tools (with AI if needed)
Version Control	Maintain history and baselines of changes

Requirements change management process



- Start with a *problem or request for change*.
- Proceed with:

1. 🧩 **Analysis**: Understand the change.
2. 💰 **Costing**: Assess impact and effort.
3. 🔧 **Implementation**: Make the change.

- Result: 🔁 **Updated Requirements**

Principles of Change Management

- Changes to requirements are inevitable and must be handled in a structured way.
- The process applies to any baselined artifact: SRS, design docs, code, etc.
- Key enablers:
- 📄 **Change Request Form (CRF)**
- 👷 **Change Control Board (CCB)**

"Walking on water and developing software from a frozen spec are equally easy — if both are frozen."

Change Management Lifecycle

1. **Initiation & Submission**
   - Change is proposed via the **CRF**.
   - Description must include rationale, scope, urgency, and initiator's viewpoint.

2. **Impact Analysis & Decision**

- CRF reviewed by config. manager + team (including client reps).
- **CCB** meets to decide:
  - Approve, Reject, Partially approve, Defer, Request further analysis

Change Management Lifecycle …..

3. **Implementation & Closure**

If approved:

Update the Implementation Plan.

Execute the change with proper quality controls.

Verify all changes and **close** the CRF.

Modern Tools Supporting RM with AI Capabilities:

Tool	Capabilities
Jama Connect	End-to-end traceability, impact analysis, real-time collaboration
Helix RM	Baselining, change tracking, customizable workflows
ReqView	Lightweight tool with traceability matrix, change history
Azure DevOps	Requirements linking, backlog management, agile boards
AI-assisted tools (e.g., ChatGPT, Copilot)	Support in drafting, rewording, validating requirements against standards

Configuration Manager's Role in Requirements Change Management

- The **Configuration Manager (CM)** is the custodian of all *configuration items* (CIs) once they are baselined.
- No changes to baselined items (e.g., SRS, design docs, test plans) are allowed unless the **Change Request Form (CRF)** is reviewed and formally **approved by the Change Control Board (CCB)**.
- The CM ensures:
  - 🔒 Change integrity – only authorized, approved changes are applied.
  - 🔗 Traceability – maintains links between changes and affected documents or components.
  - 📚 Version control – all modified items are tracked with appropriate versioning and status updates.

What is Requirement Traceability?

- Requirement traceability is the ability to describe and follow the life of a requirement, in both forward and backward directions—from its origin through development, testing, and deployment.

Key Purpose:

- Ensures every requirement is implemented and verified.
- Enables alignment with business goals.
- Supports change impact analysis and audits.

Types of Traceability

Type	Description
Forward Traceability	From requirements → design, implementation, and test cases. Helps ensure implementation covers all requirements.
Backward Traceability	From design/test cases → original requirements. Helps validate the origin and necessity of features.
Bidirectional Traceability	Combines forward and backward. Ensures comprehensive coverage and traceability throughout lifecycle.

Importance of Requirement Traceability

- Validates that deliverables meet business goals.
- Detects missing or extra features (scope creep).
- Improves impact analysis during changes.
- Helps in regulatory compliance and audits.
- Supports effective project management (scope, cost, time, risk).

Traceability in Practice

What It Enables:

- 🔍 Identifies overlooked requirements in design/test.
- 🚫 Detects unnecessary features without linked requirements.
- 🔁 Tracks requirement evolution and dependencies.
- 🔧 Supports better change management and regression testing.

Requirement Identifiers	Reqs Tested	REQ1 UC 1.1	REQ1 UC 1.2	REQ1 UC 1.3	REQ1 UC 2.1	REQ1 UC 2.2	REQ1 UC 2.3.1	REQ1 UC 2.3.2	REQ1 UC 2.3.3	REQ1 UC 2.4	REQ1 UC 3.1	REQ1 UC 3.2	REQ1 TECH 1.1	REQ1 TECH 1.2	REQ1 TECH 1.3
Test Cases	321	3	2	3	1	1	1	1	1	1	2	3	1	1	1
Tested Implicitly	77														
1.1.1	1	x													
1.1.2	2		x	x											
1.1.3	2	x											x		
1.1.4	1			x											
1.1.5	2	x												x	
1.1.6	1		x												
1.1.7	1			x											
1.2.1	2				x		x								
1.2.2	2					x		x							
1.2.3	2								x	x					
1.3.1	1										x				
1.3.2	1										x				
1.3.3	1											x			
1.3.4	1											x			
1.3.5	1											x			
etc....															
5.6.2	1														x

Tools Supporting Traceability

Tool	Key Features
Jama Connect, IBM DOORS	Enterprise-level requirement & traceability support
Jira + Xray/Zephyr	Agile traceability through epics, stories, and test links
ReqView, Helix RM	Lightweight and compliant tools
Spreadsheets (basic)	Used in small or early-stage projects

Best Practices in Traceability

- Assign unique IDs to each requirement.
- Maintain traceability links throughout lifecycle.
- Regularly update traceability matrix.
- Automate with tools when possible.
- Use consistent terminology and structured SRS.

Tools for Requirements Engineering

Tool	Description
Jama Connect	A collaborative requirements management platform offering real-time reviews, change tracking, and end-to-end traceability. https://www.jamasoftware.com
IBM DOORS Next	A scalable solution for capturing, tracing, analyzing, and managing changes to requirements in large systems. https://www.ibm.com/products/engineering-requirements-management
Helix RM (Perforce)	Provides version control and traceability between requirements, design, and test cases in safety-critical systems. https://www.perforce.com/products/helix-rm

Tool	Description
ReqView	A lightweight desktop tool that enables structured SRS creation with hierarchical and traceable requirements. https://www.reqview.com
Modern Requirements4 DevOps	Extends Azure DevOps with visual requirements modeling, smart documentation, and review tools. https://www.modernrequirements.com
Jira + Xray/Zephyr	Popular Agile work tracking tool combined with test management plugins for managing evolving requirements. https://www.atlassian.com/software/jira