

Unit-3

Requirements Engineering

Deepak Palavalasa
Assistant Professor
AI&DS Department

Problem Recognition

Problem recognition is the process of identifying and clearly understanding a problem in a system before designing or implementing a solution.

Why it Matters

- To understand the actual problem before solving it
- To avoid wrong or incomplete solutions
- To save time, cost, and effort

What is Requirement?

Is getting a coffee
every morning a
requirement for you?

Perhaps having a
laptop that works is a
requirement for you?

What about...Is it a
requirement that
you are paid every
month?



What is Requirement?

A requirement is a clear and specific statement that defines a need or expectation that a system or product must satisfy.

It describes what the system should do or the conditions it must meet to be considered complete and successful.

What is Requirement Engineering?

The process to gather the software requirements from client, analyze and document them is known as requirement engineering.

The goal of requirement engineering is to develop and maintain a sophisticated and descriptive System Requirements Specification document.

Requirement Engineering?

It is a four-step process, which includes

1. Feasibility Study
2. Requirement Gathering
3. Software Requirement Specification
4. Software Requirement Validation

Feasibility Study

A **Feasibility Study** is a detailed study done **before starting a software project** to find out whether the project is **possible, worth developing** within the available **time, money, and technology**.

It helps management decide **whether to start the project or not**.

Feasibility Study

Types of Feasibility Study



Feasibility Study

Main Types:

1. Technical Feasibility

Checks if required **technology and skills** are available.

2. Economic Feasibility

Checks if the project is **worth the cost**.

3. Operational Feasibility

Checks if users will **accept and use** the system.

4. Schedule Feasibility

Checks if the project can be completed **on time**.

Requirement Gathering

Requirement Gathering is the process of **collecting, understanding, and documenting** what the users need from a software system **before development starts.**

Requirement Gathering

Why Requirement Gathering is Important

- Avoids misunderstandings
- Reduces rework and cost
- Ensures user satisfaction
- Helps in proper system design

Requirement Gathering

Types of Requirements in Software Engineering

1. Functional Requirements

These describe **what the system should do**.

2. Non-Functional Requirements

These describe **how the system should work**.

3. User Requirements

These are requirements written in **simple language** for users.

4. System Requirements

These are detailed and technical requirements used by developers.

What is a Stakeholder?

Stakeholders are the people or groups who get affected either **positively** or **negatively** — by a company's decisions, performance, or business situation.

If someone gains or loses because of the company, they are a stakeholder.

Examples of Stakeholder?

Employees:— their jobs and working conditions depend on the organization.

Customers/clients:— they use the product or service.

Investors/shareholders:— they put money into the organization and want returns.

Suppliers:— they provide goods or services.

Communities:- may be affected by company operations.

Government/regulators:— set rules and standards.

Requirement Engineering Tasks

Requirements Engineering is the systematic process of identifying, analyzing, documenting, validating, and managing the requirements of a software system.

Requirement Engineering Tasks

Seven distinct tasks

1. Inception
2. Elicitation
3. Elaboration
4. Negotiation
5. Specification
6. Validation
7. Requirements Management

Requirement Engineering Tasks

These tasks ensure that the **software accurately captures, documents, and manages** the client's needs and expectations.

The tasks include:

Inception

Identifying the problem, stakeholders, business goals, and system scope.

It answers *why* the system is needed and *who* is involved.

Requirement Engineering Tasks

Elicitation

Gathering requirements from stakeholders through interviews, workshops, questionnaires, observation, etc. It focuses on understanding *what* users need.

Elaboration

Refining and expanding the gathered requirements into detailed, structured forms (use cases, user stories, models). Ambiguities are clarified here.

Requirement Engineering Tasks

Negotiation

Resolving conflicts among stakeholders' requirements and prioritizing them based on constraints like **cost, time, and feasibility**.

Specification

Documenting the agreed requirements formally and clearly, usually in a **Software Requirements Specification (SRS)** or equivalent artifact.

Requirement Engineering Tasks

Validation

Ensuring the requirements are **correct, complete, consistent, feasible**, and aligned with stakeholder needs, often through reviews, walkthroughs, and prototypes.

Requirements Management

Handling changes to requirements throughout the project lifecycle, maintaining traceability, version control, and impact analysis.

Requirement Engineering Tasks

Example:

For an online shopping app, tasks include collecting features like **login**, **product search**, **cart** management, and analyzing overlaps with existing systems.

documenting them in SRS; validating with client approval, and updating requirements if the client adds new features.

Software Requirement Specification - SRS

An SRS is a detailed document that explains what a software system should do and how it should behave. It describes all the requirements, both *functional* and *non-functional*, before the software is developed.

In simple words:

SRS = A blueprint or contract between the client and the developers about what the software will do.

An Example of SRS

Document Title
Author(s)
Affiliation
Address
Date
Document Version

1. Introduction
 - 1.1. Purpose of this document
 - 1.2. Scope of this document
 - 1.3. overview
2. General Description
3. Functional Requirements
 - 3.1. Description
 - 3.2. Criticality
 - 3.3. Technical issues
 - 3.4. Cost and Schedule
 - 3.5. Risks
 - 3.6. Dependencies with other requirements
 - 3.7. Any other appropriate.
4. Interface Requirements
 - 4.1. User Interface
 - 4.1.1. GUI
 - 4.1.2. CLI
 - 4.1.3. API
 - 4.2. Hardware interfaces
 - 4.3. Communications interfaces
 - 4.4. Software interfaces.
5. Performance Requirements
6. Design Constraints
7. Other Non-Functionality Attributes
 - 7.1. Security
 - 7.2. Binary compatibility
 - 7.3. Reliability
 - 7.4. Maintainability
 - 7.5. Portability
 - 7.6. Extensibility
 - 7.7. Reusability
 - 7.8. Application compatibility
 - 7.9. Resource utilization
 - 7.9.1. Serviceability
8. Operational Scenarios
9. Preliminary Schedule
10. Preliminary Budget
11. Appendices

An Example of SRS

The SRS serves as a reference for developers, testers, and stakeholders. It includes functional requirements (what the system does) and non-functional requirements (how the system behaves, like performance, security).

Example:

- Functional:** The system allows users to log in, view marks, and download assignments.
- Non-Functional:** The system should respond to requests within 2 seconds and be accessible on mobile devices.

Use Cases

A Use Case describes how a user interacts with the system to achieve a specific goal.

Use cases help understand the functional requirements from the user's side. Each use case defines:

Actor: Who is using the system.

Steps/Scenario: How the interaction happens.

Goal: What the user achieves.

Use cases - Actors

- **An Actor is outside or external the system. It can be a:**
 1. Human
 2. Peripheral Device (Hardware)
 3. External System or Subsystem
 4. Time or time-based event
- **Represented by the following figure:**



Use cases - Relationships

- Represent communication between actor and use case
- Depicted by line or double-headed arrow line
- Also called association relationship



A UML Use Case symbol, represented by an oval containing the text "Make Appointment".



Use Cases

Example:

Use case: **“Withdraw Money from ATM”**

Actor: Bank customer

Steps: Insert card → Enter PIN → Select amount → Cash dispensed
→ End session

Goal: Customer successfully withdraws money

Functional Specification

Functional specification is a detailed description of **all the functions and features** that the system must perform.

Explanation:

It focuses on “**what the system should do**”. It guides developers in implementing the required functions.

Example:

For an e-commerce website:

The system should allow users to browse products, add to cart, make payments, track orders, and generate invoices.

Requirements Validation

Requirements Validation is the process of ensuring that the documented requirements are **correct, complete, consistent, and meet user expectations.**

Example:

For an online booking system, validation ensures all requirements such as **login, seat selection, payment, and ticket generation** work as intended and meet client expectations.

Requirements Validation (Contd.)

Requirement checking can be done in following manner:

1. Validity Checks
2. Consistency Checks
3. Completeness Checks

Validity Checks

Validity checks ensure that each requirement truly reflects the actual needs of stakeholders and aligns with the business goals. A requirement is *valid* only if it solves the right problem.

What validity checks look for:

- Does this requirement satisfy a real user/business need?
- Is the requirement relevant to the project scope?
- Did the correct stakeholders approve this requirement?

Consistency Checks

Consistency checks ensure that no requirement contradicts another requirement. All terminology, constraints, and rules must be applied uniformly.

What consistency checks look for:

- Do any two requirements conflict?
- Is terminology used in a consistent way?
- Are rules aligned across the document?

Completeness Checks

Completeness checks usually focus **ONLY** on **missing information, missing scenarios, or incomplete requirements**.

However, in requirement validation as a whole (not just completeness), we have additional checks such as:

- **Realism (Feasibility) Checks**
- **Verifiability (Testability) Checks**

Completeness Checks

- **Realism checks**

1. Using knowledge of existing technology, the wants should be checked to make sure that they can actually be implemented.
2. These checks should also appreciate the budget and schedule for the system development.

Completeness Checks

- **Verifiability**

3. To reduce the potential for dispute between the customer and the contractor

4. System requirements should even be written in order that you ought to be ready to write a group of tests that will demonstrate that the delivered system meets each specified requirement.

Requirement Validation Techniques

Requirements Reviews

1. both the customer and contractor staff should be involved in reviews.
2. Reviews could also be formal (with completed documents) or informal.
3. Good communication should happen between developers, customers and users. Such communication helps to resolve problems at an early stage.

Requirement Validation Techniques

Prototyping:-

The requirements can be examined through executable model of system

Test Case Generation:-

1. If the tests for the wants are devised as a part of the validation process, this often reveals requirements problems.
2. If a test is difficult or impossible to style , this usually means the wants are going to be difficult to implement and will be reconsidered.
3. Developing tests from the user requirements before any code is written is an integral a part of extreme programming

× ○ DIGITAL LEARNING CONTENT



Parul[®] University



www.paruluniversity.ac.in