

Functional and Non-Functional Requirements in Software Engineering

1. Introduction

Requirements are the needs and expectations of users and other stakeholders from a software system. They describe what the system should do and how it should behave.

"Functional requirements tell us what the system should do, while Non-functional requirements tell us how well it should do it."

2. Functional Requirements

Functional requirements define the specific functions or services that the system must provide. They are directly related to the features and operations of the system.

Key Points

- Describe the behaviour and features of the system.
- Often written from the user's perspective.
- Usually tested through specific test cases.
- Answer the question: "What will the system do?"

Examples of Functional Requirements

- The user should be able to register/login to the system.
- The system should allow users to add, update and delete data.
- The system should generate a report in PDF format.
- The user should be able to search for products.
- The system should send a confirmation email after registration.

3. Non-Functional Requirements

Non-functional requirements define the quality attributes and constraints of the system. They describe how well the system should perform rather than what specific functions it should provide.

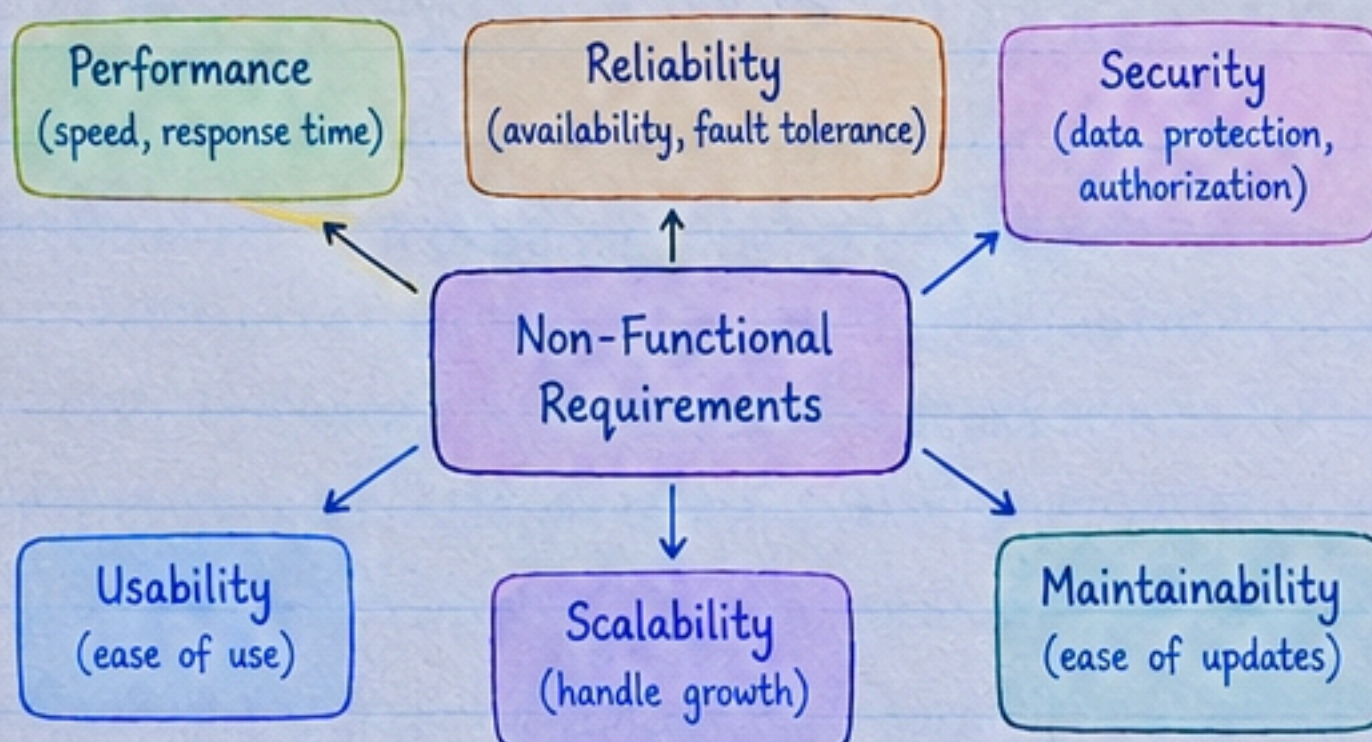
Key Points

- Describe the performance, reliability, security, etc.
- Often related to the overall system behaviour.
- May be harder to test than functional requirements.
- Answer the question: "How well will the system do it?"

Examples of Non-Functional Requirements

- The system should respond within 2 seconds.
- The system should be available 99.9% of the time.
- The data should be encrypted.
- The system should be scalable to handle 10,000 users.
- The application should be easy to use (good usability).
- The system should be compatible with modern browsers.
- The software should follow industry security standards (e.g., GDPR).

4. Types of Non-Functional Requirements



5. Conclusion

Both functional and non-functional requirements are important for the success of a software project.

- ✓ Functional requirements tell us what to build.
- ✓ Non-functional requirements tell us how well to build it.

Together, they ensure the system is useful, reliable, secure and user-friendly.

Software Requirements Document (SRD)

in Software Engineering

1. Introduction

A Software Requirements Document (SRD) is a formal document that describes what the software system should do. It captures the needs of the stakeholders and serves as a contract between the customer and the development team.

"The Software Requirements Document is a written representation of the system requirements, serving as a basis for design, development, testing and acceptance."

2. Objectives

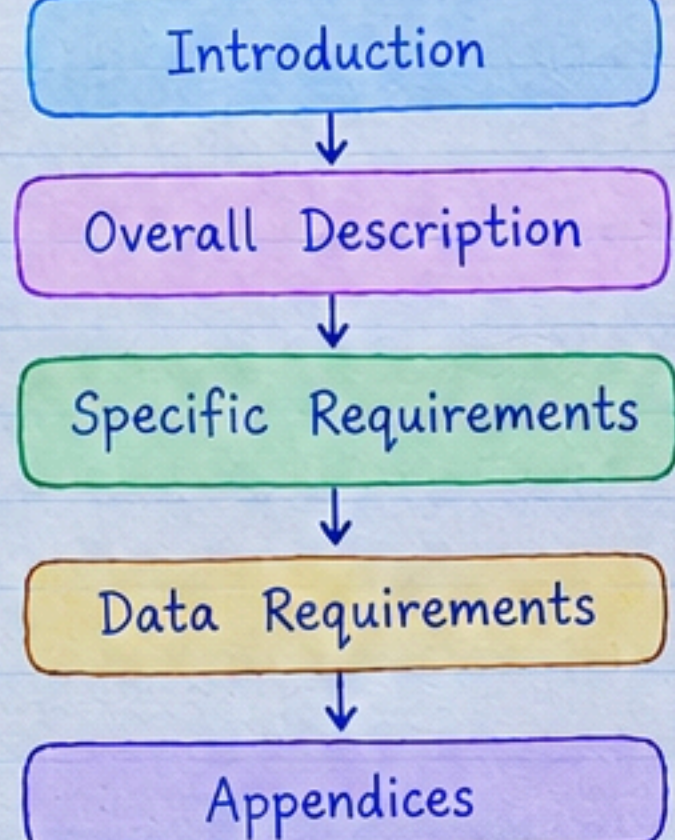
- Understand and document the needs of stakeholders.
- Provide a clear and unambiguous description of the system.
- Serve as a reference throughout the software development life cycle.
- Reduce risks and avoid misunderstandings.
- Help in planning, estimation and resource allocation.

3. Contents of an SRD

A typical SRD includes the following sections:

- ① **Introduction** – Purpose, scope, definitions, acronyms.
- ② **Overall Description** – Product perspective, features, system context, user classes.
- ③ **Specific Requirements** – Functional requirements, non-functional requirements, interfaces, constraints.
- ④ **Data Requirements** – Data dictionary, data formats.
- ⑤ **Appendices** – Supporting information (diagrams, forms, examples, etc.).

Structure of an SRD



4. Types of Requirements

The SRD usually contains two types of requirements:

① Functional Requirements

- Describe what the system should do.
- Related to the features and services provided to the user.
- Example: Login, search, generate report.

② Non-Functional Requirements

- Describe how well the system should perform.
- Related to quality attributes (performance, security, usability, reliability, etc.).
- Example: Response time < 2 seconds, data encryption, 24x7 availability.

5. Importance

- Helps in clear communication between stakeholders and developers.
- Minimizes changes and rework.
- Improves project planning and estimation.
- Ensures the final product meets user expectations.
- Acts as a reference for testing and maintenance.

6. Conclusion

The Software Requirements Document is a crucial part of the software development process. It ensures that the right product is built, with the right features and quality, to meet the needs of the users.

★ Remember

- SRD = What to build (not how to build).
- It is prepared after gathering and analyzing requirements.
- It is a living document (may evolve).

① Requirements Specification

1. Introduction

Requirements specification is the process of documenting the requirements gathered from stakeholders in a clear, complete, consistent and unambiguous way. It acts as a contract between the customer and the development team.

2. Purpose

- Provides a clear understanding of what needs to be built.
- Serves as a basis for design, development and testing.
- Reduces ambiguity and avoids misunderstandings.
- Helps in managing changes.

3. Key Elements / Contents

- Functional requirements – what the system should do.
- Non-functional requirements – how the system should perform (e.g., performance, security, usability).
- Constraints – limitations (e.g., budget, technology, standards).
- Assumptions – factors considered as true.
- Models/Diagrams – use case diagrams, data flow diagrams, etc.

4. Example

"The system should allow users to register, login, search for products and make payments. It should respond within 2 seconds and be available 99% of the time."

A good requirements specification is clear, complete, consistent and unambiguous.

② Requirements Engineering Processes

Requirements engineering is the overall process of discovering, analysing, documenting, validating and managing requirements throughout the software development life cycle.

1. Key Processes

① Inception / Elicitation

Understand the problem, identify stakeholders, and gather initial requirements.

② Analysis

Study the requirements, resolve conflicts, prioritize and refine them.

③ Specification

Document the requirements in a formal SRS.

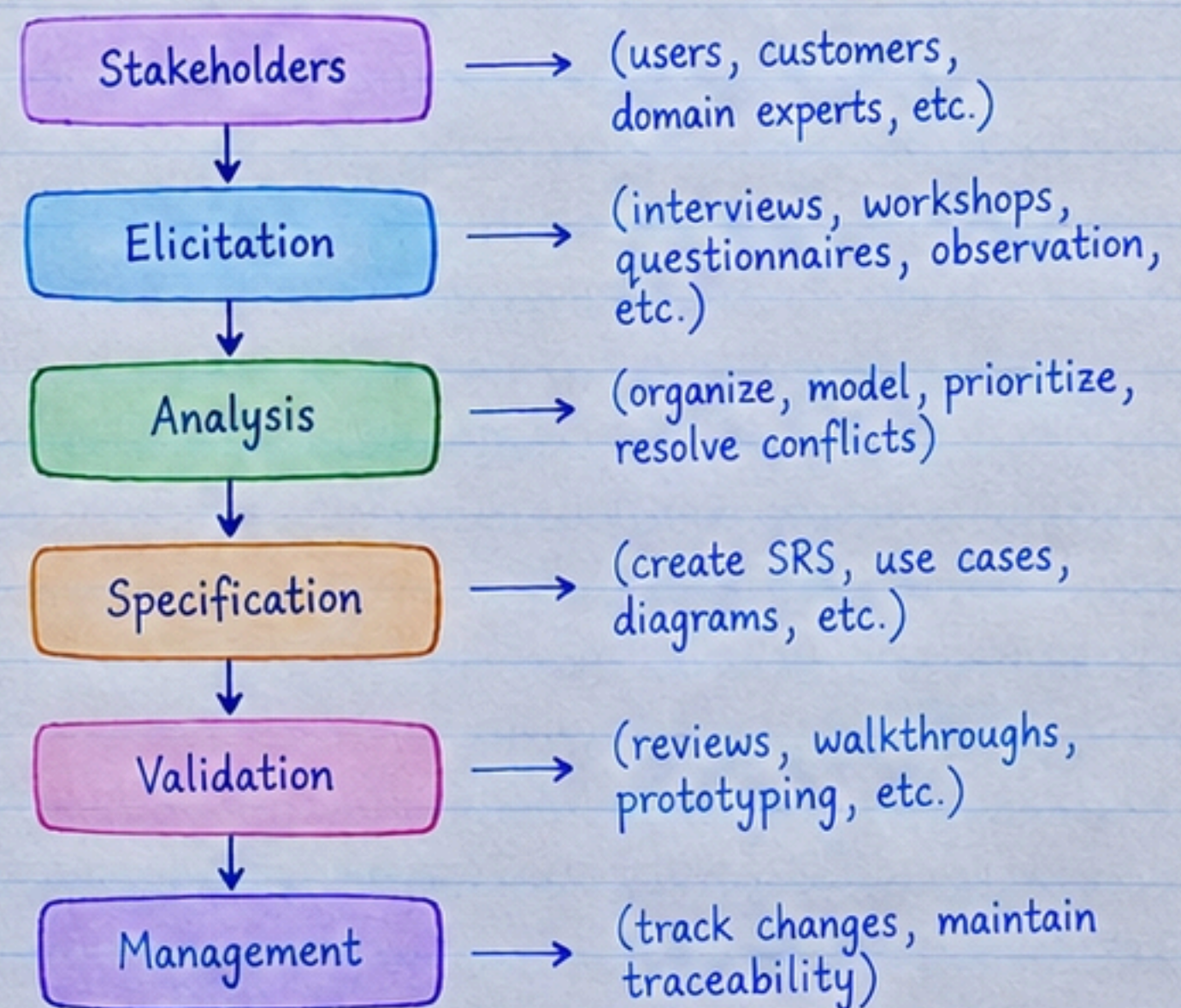
④ Validation

Check the requirements with stakeholders for correctness, completeness and feasibility.

⑤ Management

Handle changes, maintain traceability and ensure ongoing consistency.

2. Flow of Requirements Engineering Process



3. Outcome

- Software Requirements Specification (SRS)
- Traceability Matrix
- Well-defined and agreed-upon requirements

① Requirements Validation in Software Engineering

1. Introduction

Requirements validation is the process of checking that the specified requirements are correct, complete, consistent and feasible. It ensures that what is written in the requirements document actually meets the needs of the stakeholders and can be implemented in the system.

2. Objectives

- Ensure requirements are correct and complete.
- Check consistency with business goals and system design.
- Identify and remove ambiguities and errors.
- Ensure feasibility (technical, operational and cost).
- Gain stakeholder agreement (especially user).

3. Techniques for Validation

- **Reviews** – peers and experts review the requirements.
- **Walkthroughs** – step-by-step discussion with team and stakeholders.
- **Inspections** – formal process with defined roles, checklists and reporting.
- **Prototyping** – build a quick model to validate requirements.
- **Test case derivation** – derive test cases from requirements and check for correctness.

Validation answers the question:
"Are we building the right thing?"

4. Outcome

- Validated and approved requirements document.
- List of issues/changes (if any).
- Increased confidence among stakeholders.

② Requirements Management in Software Engineering

1. Introduction

Requirements management is the process of tracking, controlling and maintaining the requirements throughout the software development life cycle. It ensures that changes are handled properly and all stakeholders are informed.

2. Objectives

- Maintain a single, consistent and up-to-date version of requirements.
- Track changes and their impact.
- Ensure traceability (link requirements to design, code and test cases).
- Manage priorities and handle conflicts.
- Keep stakeholders informed.

3. Activities in Requirements Management

- ① **Change Management** – evaluate and approve changes.
- ② **Version Control** – maintain different versions of requirements.
- ③ **Traceability** – link requirements to other artifacts.
- ④ **Prioritization** – set priorities based on business value.
- ⑤ **Communication** – keep all stakeholders updated.
- ⑥ **Status Tracking** – monitor progress and resolve issues.

4. Tools

- Jira
- Confluence
- DOORS
- Rally
- Azure DevOps

5. Outcome

- Well-managed requirements.
- Reduced risk of scope creep.
- Better communication and collaboration.
- Higher quality software.

Requirements management answers the question:
"Are we managing the right thing in the right way?"