Requirements Engineering

Samenvatting

Manuel Mol .

Samenvatting

This document provides a foundational introduction to Requirements Engineering concepts. It covers the basics of requirements, the requirements engineering process, and the different types of requirements. It also discusses the importance of requirements engineering and the challenges that come with it.

Contents:

Samenvatting

Published Jun 11, 2024

Introduction:

The 3 levels of Requirements Artifacts:

1. **Goals:** What the system should achieve
2. **Scenarios:** Describes a concrete example of satisfying or failing to satisfy a goal
3. **Requirements:** Define the data, functions, behavior, quality, and constraints, often imply solution

Scenarios:

Scenarios are the glue between abstract models and reality. They can be abstracted (video) or concrete (sequence diagram). Abstraction can also be achieved by using different english levels.

Information in scenarios:

Some typical information in scenarios:

1. **Actor:** The entity that interacts with the system
2. **Roles**: The roles of the actors
3. **Goal:** The reason for the interaction
4. **Precondition:** The state of the system before the interaction
5. **Postcondition:** The state of the system after the interaction
6. **Location**: Where the interaction takes place
7. **Req. resources:** The resources needed for the interaction

8 scenario types:

1. **State scenarios:**
    1. **Current-state scenarios:** Describe the current situation
    2. **Desired-state scenarios:** Describe the desired situation
2. **Type & Instance:**
    1. **Type scenario:** describes the general process
    2. **Instance scenario:** Describes a specific instance of a type scenario, added when there is need for more clarification.
3. **Alternatives:**
    1. **Main scenario:** sequence of interactions to satisfy (set of) goal(s)
    2. **Alternative scenario:** sequence of interactions, instead of main scenario, to satisfy the same (set of) goal(s) → “Plan B”
4. **Exception scenario:** Describes what happens when something goes wrong. As a consequence one or more of the initial goals will not be satisfied.
5. **What if? scenarios:** Describes what happens when the environment changes. Documents alternative realizations.
    1. **Descriptive scenario:** Describes process or workflow
    2. **Explanatory scenario:** Provides background information and rationales
6. **Misuse scenarios:** Describes how the system can be misused
7. **Negative scenarios:** Goal is not satisfied. Should be explicitly not allowed.
8. **System-internal, interaction and context scenarios:**
    1. **System-internal:** focus on interactions within system
    2. **Interaction:** focus on interactions between system and actors
    3. **Context:** focus on interactions between system and environment

Documenting scenarios:

Narrative scenarios
Describe the scenario in a narrative way, as a sequence of events.

Diagrams

1. **Sequence diagrams:** Show the sequence of interactions between actors and the system
2. **Activity diagrams:** Show the flow of activities in a scenario.
3. **Use case diagrams:** Suited for documenting relationships between use cases

Business problems:

Scoping the business problem:

What is the purpose of the project, what is the scope of the work and what are the goals of the project? This should take into account the needs of the stakeholders. It should also include:

- **Constraints**: restrictions which help determine the scope of the work. They can be solutions constraints or project constraints.
- **Terminology:** define from the start to enhance clarity and understanding. make sure it's shared terminology
- **Estimated costs:** measured is always better than guess
- **Risks**: identify, assess and manage risks.
- **To go or not to go:** SWOT and other models can help here

The scope, goals and stakeholders of the business problem are interconnected.

Goals:

What do you want to achieve? Make sure your goals are SMART:

- **Specific:** Clearly defined
- **Measurable:** Can be measured
- **Achievable:** Can be achieved
- **Relevant:** Relevant to the stakeholders
- **Time-bound:** Achievable within a certain time frame

Goal orientation is important because it helps to ensure that the system meets the needs of the stakeholders. It also helps to reduce the risk of project failure. It provides focus and gives justification for the requirements.

Goals can be set on different levels:

- **AND-decomposition:** All subgoals must be achieved
- **OR-decomposition:** At least one subgoal must be achieved

Goal dependencies:

- **Requires dependency:**
  - Satisfaction of Goal X is prerequisite for satisfaction of Goal Y
- **Support dependency**
  - Satisfaction of Goal X contributes to satisfaction of Goal Y
- **Obstruction dependency:**
  - Satisfaction of Goal X hinders satisfaction of Goal Y
- **Conflict dependency:**
  - Satisfaction of Goal X excludes satisfaction of Goal Y
- **Goal equivalence:**
  - Satisfaction of Goal X leads to satisfaction of Goal Y

How to document goals:

1. be concise
2. bedrijvende vorm (actief), voorbeeld: "Het systeem moet…"
3. Precise documentation
4. Decompose goals
5. Explain added value
6. Document why the goal is introduced
7. Avoid unnecessary restrictions

Goals initiate the definition of the scenarios. The definition of scenarios leads to refinement of the goals. Goals can also be used to classify scenarios.

Scope:

Your scope shouldn't focus exclusively on the system, but should also include humans.

Stakeholders:

Examples:

- Sponsor
- Customers
- Users
- Management

BUC (business use case):

A BUC is a collection of related scenarios that describes how a system or product should respond to a specific business event.

BUC requirements:

- **A "natural" partition of the work**, making an obvious and logical contribution.
- **Isolated**, with minimal connectivity to other parts of the work.
- Clearly **defined in scope**, with rules for defining that scope.
- **Recognisable to stakeholders**, using names and terms they understand.
- **Readily determined to exist**, not something that is unclear or debatable.
- **Supported by one or two known experts** for that part of the work.

Buc structure:

- **Context**
- **One main scenario**
- **One or more alternative scenarios**
- **One or more exception scenarios**

Scope:

The scope of each BUC includes anything that you are allowed to change, as well as anything you need to understand to determine what can or should be changed.

Business events
are things that happen outside the system that the system needs to respond to. They can be triggered by people, other systems, or even time passing. When a business event happens, the system should respond by **initiating the correct BUC**

Motivatie voor requirements engineering:

Systems mainly driven by software.

N.I.V.E.A.
Niet Invullen Voor Een Ander.

Requirements engineering is important because it helps to ensure that the system meets the needs of the stakeholders, and it helps to reduce the risk of project failure. It reduces costs because logical errors can be detected early in the process.

RE can be embedded in the business:
1. **Processes:** Marketing, sales, production, maintenance
2. **Developmental processes:** Project management, design, quality assurance.

Involving these stakeholders reduces the risk of project failure.

RE vs SE:

Software is build according to specifications. These specifications come from requirements.

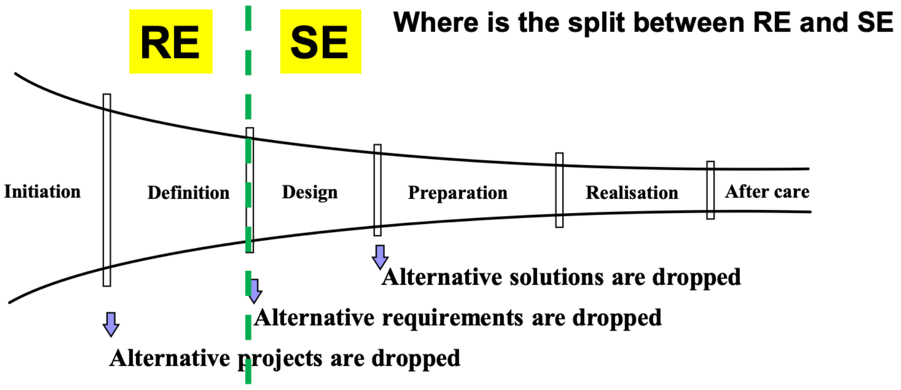
RE
SE
Where is the split between RE and SE
Initiation
Definition
Design
Preparation
Realisation
After care
Alternative solutions are dropped
Alternative requirements are dropped
Alternative projects are dropped

Figure 1: Fuik van de softwareontwikkeling

Fundamental truths:

1. Requirements are about understanding the real business problem (or opportunity) and providing a solution for it.
2. We build software with optimal value for the stakeholders.
3. Understand the need to build the right software
4. Building software does not necessarily solve the whole business problem
5. Requirements have to become known to the builders
6. Customers do not always give 'the right' answer.
7. Requirements do not come about by chance
8. Iterative approaches (like Agile/Scrum) do NOT make requirements [engineering] redundant.
9. There is no 'silver bullet'.
10. Requirements should be measurable and testable.
11. You will change the way the user thinks.

Requirements engineering process:

RE is an iterative process! It's about what you want to build, not yet about how you want to build it.

3 dimensions of RE

1. **Content:** from vague to complete
2. **Agreement:** from conflicting to agreed
3. **Documentation:** from informal to formal

Requirements engineering is about understanding the real business problem (or opportunity) and providing a solution for it. Requirements process is iterative

The Volere requirements process:

The Volere requirements process is a process for gathering, documenting, and managing requirements. It consists of 9 steps:

1. **Start of project**: define business problem and scope, identify stakeholders and goals. produce cost estimate and decide whether to proceed.
2. **Requirements elicitation:** gather requirements from stakeholders, document requirements, and validate them. Avoid *old wine in new bottles*.
3. **Writing and documenting requirements:** write requirements in a structured way, unambiguous (using a rationale), testable using **fit criterion**. The process of writing requirements might be even more important.
4. **Quality Gateway:** testing each requirement for completeness, correctness, measurability, absence of ambiguity and several other attributes, before allowing the requirement to be passed to the developers.
5. **Reusing requirements:** like quality requirements
6. **Reviewing the requirements:** nothing missing, consistent, conflicts resolved, etc.
7. **Iterate:** iterate on the requirements until they are complete and correct.
8. **Retrospective:** what went well, what went wrong, what can be improved.
9. **Evolution of requirements:** requirements change over time, so they need to be managed.

The Volere template is a template about *What* to write about. The *snow card* is a guide to *how* to write about it.

The future how (solution):

There are many factors to decide what the best solution is, such as: cost, constraints, PUC (Product Use Case), innovation, organizational goals, etc.

Once you understand the complete BUC, you can establish how much of it wil be solved by the product. This is the *future how*. Keep in mind the **real origin** of the business event. Also consider the users and don't forget NIVEA.

Also keep in mind the technology and systems already in place.

Product Use Case
sets out functionality of the product. Good tool to consolidate the views of the stakeholders.

Sequential strategy:

In a sequential strategy, the requirements are first gathered and then the software is developed. This is a traditional approach.

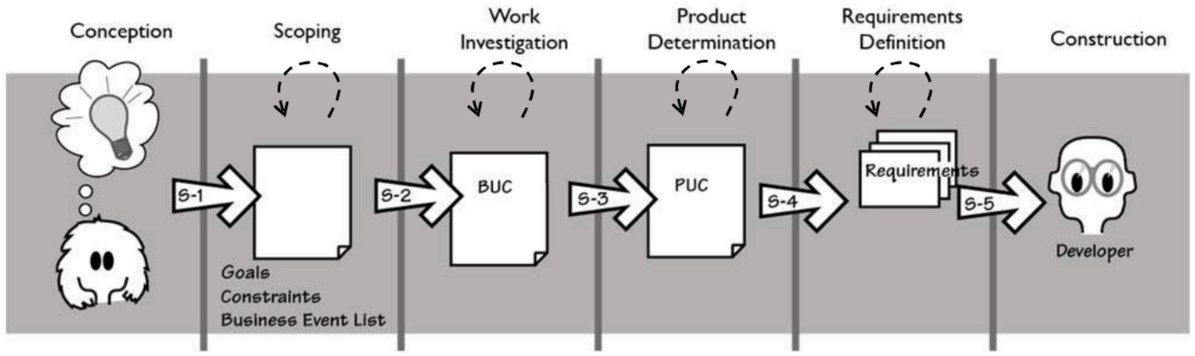

Figure 2: Sequential strategy

1. **Conception** → **Scoping:** establish goals, sponsors, identify key stakeholders, etc.
2. **Scoping -> Investigation:** identify the work, all stakeholders, detailed constraints, etc.
3. **Investigation** → **product Determination:** understanding business carried out by each BUC.
4. **Product Determination** → **Atomic requirements:** Specification for each PUC is understood and agreed.
5. **Atomic requirements** → **Construction:** Detailed requirements are understood and agreed.

Iterative strategy:

Basically the same as the sequential strategy, but the steps are repeated. This is a more modern approach. Construction is done in small increments. BUC's are prioritized and the most important ones are developed first.

Breakout:

The point at which the requirement knowledge is sufficient to move to the next activity (software development).

In a **sequential strategy**, the breakout is at the end of the requirements phase.

Quality Gateway:

The quality gateway is a point in the requirements process where the requirements are tested for completeness, correctness, measurability, absence of ambiguity, and several other attributes, before allowing the requirement to be passed to the developers. The requirement must pass a set of tests:

- **Within scope:** does the requirement fall within the scope of the project?
- **Relevant:** is the requirement relevant to the stakeholders/project?
- **Complete:** no missing attributes?
- **Fit criterion:** is the requirement testable?
- **Consistent:** does the requirement conflict with other requirements?
- **Viable:** can the requirement be implemented?
- **Requirement or Solution:** is the requirement a requirement or a solution?
- **Gold plating:** is the requirement more than what is needed?
- **Creep:** is the effort worth the benefit?
- **Traceable:** can the requirement be traced back to a goal?

Quality control is important, because it prevents **error propagation** and **cost escalation** down the line. Principles of quality control:

- Involve the right stakeholders:
- Separate defect detection from defect correction
- Multiple views
- Use appropriate documentation formats
- Create development artifacts (e.g. prototypes, test, architecture)
- Repeat quality control

Some techniques for quality control are **inspection** and **prototyping**. Others are creating **checklists** and **artifacts**

Reusing requirements:

Requirements can often be reused across projects with similar purpose. Existing stakeholders can also be a source of reused requirements. Some sources of requirements include colleagues, experiences, existing specs, domain models and books.

Requirement patterns
are used as a guide when choosing requirements. Can improve accuracy and completeness of requirements. There are patterns for domains, and across domains.

Communicating requirements:

Use a template to communicate requirements. The Volere template is a good example. It is a template about *What* to write about. The *snow card* is a guide to *how* to write about it. It's important to document. The requirements artifacts should be complete, traceable, correct, unambiguous, comprehensible, consistent, rated, verifiable, upt-to-date and atomic. Use simple language.

Requirements elicitation:

Requirements elicitation should be focust on diversity of requirement sources. It focuses on **goals**, **scenarios** and the tree requirement types: **functional**, **constraints** and **quality requirements**.

Objectives of requirements elicitation:

- identify relevant requirement sources
- elicit existing requirements
- develop new requirements

Elicitation is first diverging and then converging.

Activities:

- **Identify relevant requirement sources:** stakeholders, documentation, existing systems, etc. Filter on relevance.
- **Elicit existing requirements:** interviews, questionnaires, workshops, etc.
- **Develop new requirements:** brainstorming, prototyping, etc.

The Brown Cow Model

The Brown Cow Model. This shows four views of the work, each of which provides the business analyst and the stakeholders with information that is useful at different stages of the requirements discovery process. The four views are:

1. **How (solution) now:** What is the current situation?
2. **What (essence) now:** BUCs, essence of the work
3. **Future what (essence):** Future state situation
4. **Future how(solution):** Comprehensive future state situation

Techniques:

- **Interviews:** structured, semi-structured, unstructured
  - **Preparation:** Define a goal, invite people, speak participants language, align interviews incase of multiple interviewers
  - **Execution:** Explain goal, ask questions, focus on subject, sum up and *thank participants* (positive feedback)
  - **Followup:** Summarize, send summary, ask for feedback
  - **Critical success factors:** communication skills, no leading questions, active listening, etc.
- **Questionnaires:** open, closed, Likert scale
  - **Critical success factors:** clear questions, clear instructions, stakeholders need to be motivated, etc.
- **Workshops:** brainstorming, prototyping, etc.
  - **Critical success factors:** participants need to understand goal, invite the right people, motivate, avoid groupthink, etc.
- **Observation:** observing the stakeholders in their natural environment, either watching or participating.
  - **Critical success factors:** willingness to cooperate, objectivity, etc.
- **Perspective based reading:** reading the requirements from different perspectives
  - **Critical success factors:** selecting the right perspectives.

Assistance techniques:
- **Mind mapping:** visual representation of ideas
- **Prototyping:** creating a model of the system
- **Brainstorming:** generating ideas
- **Checklist**
- **KJ method:** grouping ideas

Understanding the real problem:

Understanding the real problem is important because it helps to ensure that the system meets the needs of the stakeholders. It also helps to reduce the risk of project failure. It provides focus and gives justification for the requirements.

What is a requirement?

> A condition or capability needed by a user to solve a problem or achieve an objective that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document
>
> — IEEE

Types of requirements:

1. **Functional requirements:** What the system should do
2. **Constraints:** What the system should not do, constraints on the functional requirements that restrict the way in which the product or system should be developed
3. **Non-functional (Quality) requirements:** How the system should do it, quality attributes of the system, such as *performance, usability, reliability*, etc. They are important primarily to users or developers. They should not be hidden inside the functional requirements. They must have a **business justification**

Fit criteria:

Fit criteria are used to determine whether a requirement has been met. They are used to test the requirements. They should be clear, unambiguous, and measurable. You need to understand the **rationale** to understand the fit criteria.

Fit criteria can be derived in a few different ways:
- **Scale measurement:** using a scale to measure the requirement
- **Quality requirements:** max 1 incident per 1000 hours, always available, etc.
- **Study:** study existing practices and new practices and see if there is a change
- **What is considered failure:** what is considered a failure of the requirement
- **Completed**
- **Create tests**

Forms of fit criteria:
- Text and numbers
- Graphics: diagrams, sketches, etc.

Completeness:

Determine whether requirements are missing, prioritize requirements, and determine whether requirements are consistent and without conflicts.

Steps:

1. **Review specification**
2. **Inspections → Fagan inspections:** a formal review process that involves a team of people who review the requirements document and identify any errors or omissions.
3. **Find missing & conflicting requirements**
4. **Prioritize requirements**

Data driven RE:

take profit of the existence of large amounts of data in the form of feedback to guide requirement engineers in their decisions about what requirements to include in subsequent system releases.

Feedback can be analyzed using NLP and ML. Feedback can be gathered implicitly thru usage data or explicitly thru surveys. Context is important.

FAME
Framework for Adaptive Process Modeling and Execution. It is a framework that allows for the modeling and execution of adaptive processes. It is based on the idea that processes can be modeled as a set of activities that are executed in a specific order. The framework allows for the modeling of processes that are adaptive, meaning that they can change based on the context in which they are executed.

Crowdbased RE: is an umbrella term for automated or semi-automated approaches to gather and analyse information from a crowd to derive validated user requirements.

Decision making:

Two key dimensions of decision making:

- Visualize data in an actionable manner
- Arrange decisions in the form of a software release plan

Challenges:

- Integration with Data-oriented Analysis Cycles.
- Integration with Other Software Engineering Approaches.
- User Motivation and Trust.
- Context-Driven Feedback Gathering.
- Analysis of Implicit Feedback.
- Use of Domain Knowledge.
- Adoption by Companies

Lessons learned:

Data-driven RE is promising but not free. It is different for every company, requires expertise and needs to be implemented in a incremental way. It also requires full transparency.

Scrum & Agile:

Product management vs project management:

- **Product management:** continuous, success when customer profitability, managed on customer adding value
- **Project management:** temporary, success when project is on time, on budget, on scope, managed on time, money and scope.

VUCA

Volatile, Uncertain, Complex, Ambiguous:

1. **Volatile:** The nature and dynamics of change, and the nature and speed of change forces and change catalysts.
2. **Uncertain:** The lack of predictability, the prospects for surprise, and the sense of awareness and understanding of issues and events.
3. **Complex:** The multiplex of forces, the confounding of issues, no cause-and-effect chain and confusion that surrounds organization.
4. **Ambiguous:** The haziness of reality, the potential for misreads, and the mixed meanings of conditions; cause-and-effect confusion.

The answer to VUCA is VUCA: Vision, Understanding, Courage, Adaptability.

Agile:

A focus on the 3 V's: Value, Vision, Validation. The agile mindset is about delivering value to the customer. Agile frameworks embody this mindset into practical tools and techniques.

Scrum:

Scrum Artifacts:

- **Product Backlog:** a prioritized list of all the work that needs to be done on the project.
  - Feature requests
  - Bug fixes
  - Quality requirements
- **Sprint Backlog:** a list of tasks that need to be completed during the sprint. selected on:
  - Value (ROI, customer satisfaction)
  - Risk (dependencies, market trends, feature)
  - Size (smaller)
- **Increment:** the sum of all the product backlog items completed during a sprint.

DOVE

Description, Order, Value, Effort. This is a way to prioritize the product backlog.

Scrum events:

- **The sprint:** a time-boxed period of development, typically 2-4 weeks.
- **Sprint planning:** a meeting at the beginning of the sprint where the team decides what work they will complete during the sprint.
- **Daily stand-up:** a short meeting where the team discusses what they did yesterday, what they will do today, and any obstacles they are facing.

- **Sprint review:** a meeting at the end of the sprint where the team demonstrates the work they completed during the sprint.
- **Sprint Retrospective:** a meeting at the end of the sprint where the team discusses what went well, what could be improved, and what actions they will take to improve.

Roles:

- **Product Owner:** responsible for the product backlog and for maximizing the value of the product. Closer to the team than a product manager.
- **Scrum Master:** responsible for ensuring that the team is following the scrum process and removing any obstacles that are preventing the team from being successful.
- **Development Team:** responsible for delivering the product increment.

User story: As a [role], I want [goal], so that [reason]. A good user story should can be written by listening. You can use the snow card to write user stories.

RE management:

As a RE manager, you manage requirements engineering artifacts, system context (changes in environment, like new tech) and the requirements engineering process (choose approach).

Traceability
is the ability to trace requirements from their origin to their implementation. It is important because it validates, avoids gold plating, and helps to manage change. It also makes sure someone is accountable for the requirement.

RE prioritization:

Resources are limited, you need to prioritize. You can use the MoSCoW method: Must have, Should have, Could have, Won't have. The should be classified into different priority classes for each of the five RE activities:

- **Elicitation**
- **Documentation**
- **Agreement**
- **Validation**
- **Management**

Preparation activities:

1. **Determine which stakeholders to involve:** based on project goals
2. **Select the artifacts which are subject to prioritization:** goals first
3. **Define the prioritization criteria:** importance, urgency, risk, cost, etc.
4. **Select the appropriate prioritization technique.:** ranking, top 10, MoSCoW, Kano model, etc.

Configuration management:

CM applied over the life cycle of a system provides visibility and control of its performance, functional, and physical attributes. CM verifies that a system performs as intended, and is identified and documented in sufficient detail to support its projected life cycle. It's about the versions of the requirement artifacts and Consistency between them.

Negotiation:

Negotiation is about finding a solution that satisfies all parties. It's about finding a win-win solution. It's not conflict resolution. Negotiation is aimed at agreement between stakeholders, so all stakeholders should be involved.

1. Identify
2. Analyse
3. Resolve
4. Document