



Requirements- Engineering





The SOPHISTS

»Short & clever«

The SOPHIST
RE-PRIMER

The SOPHISTS

SOPHIST GmbH
Vordere Cramergasse 13
90478 Nuremberg
Germany
www.sophist.de

 @ SOPHIST_GmbH
 @SOPHIST.GmbH
 /sophistgmbh
 blog.sophist.de

1st edition 2021

Translated from German by Joachim Kurrer,
SOPHIST GmbH
Copy Editing & Production: David Nawzad, SOPHIST GmbH
Cover Design and Layout: Heike Baumgärtner, Büro Hochweiss

Copyright: SOPHIST GmbH

This work is protected by copyright laws.

All rights, including translation, reprinting and reproduction of the book, or parts thereof, are reserved. This work may not be reproduced, edited or copied in whole or in part, by electronic means or otherwise (photocopy, microfilm or other media) without the written consent of the SOPHIST GmbH.

It should be noted that all software and hardware names as well as brand and product names of the respective companies are normally subject to the registered trademarks and patent protections.

While reasonable care has been exercised in the preparation of this brochure, the author and the SOPHIST GmbH assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.



Requirements- Engineering

The SOPHISTs

»Short & clever«

The SOPHIST
RE-PRIMER

SOPHIST
Trainings



UPDATE

What's new about it?

All SOPHIST training courses are designed in accordance with the latest knowledge. Among other things, our trainers follow the principles of the "...from the Back of the Room" method in order to convey training content in a sustainable way.

An activating learning environment - more movement, less text, more interaction with the participants and surprising exercise concepts - makes learning fun and efficient.

Your benefits?

You benefit not only from the know-how of the method leaders, but also from a didactic implementation that leaves its mark.

Curious now?

Contact us without obligation:

+49 (0) 911 40 900 - 0

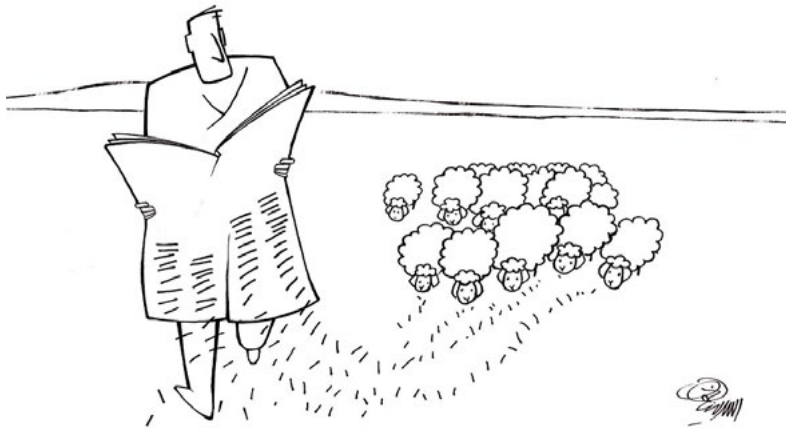
heureka@sophist.de

1.	Introduction and Motivation	8
2.	What is Requirements Engineering?	10
2.1	Classification of Requirements	12
2.2	Quality of Requirements	14
2.3	Sources of Requirements	15
2.4	Main Activities of Requirements Engineering	16
3.	Eliciting Knowledge	19
3.1	Goals, Sources and System Context	19
3.2	Elicitation of Requirements	20
3.3	Identifying Linguistic Effects – The SOPHIST Set of REgulations	22
4.	Deriving Good Requirements	25
4.1	Activities for Analyzing Requirements	25
4.2	Validating and Consolidating Requirements	28
5.	Documenting and Imparting Requirements	32
5.1	Imparting Requirements without Documentation	32
5.2	Imparting Requirements with Documentation	34
6.	Requirements Management	37
6.1	How much Requirements Management Makes Sense?	38
6.2	Versioning and Baselines	39
6.3	Traceability	40
6.4	Change and Release Management	41
7.	Establishment of an Improved Requirements Engineering	43
7.1	Changes within an Organization	43
7.2	The Establishment is a Project!	44
7.3	An Agile Design of Changes	44
7.4	Work Packages of an Establishment	47
8.	Requirements Engineering and Agility	48
9.	Systems Engineering	52
9.1	Definition and Purpose	52
9.2	Embedding of Requirements Engineering	54
10.	Requirements Engineering for Smart Ecosystems and as a Driver of Digital Transformation	56

11.	Business Analysis, Requirements Engineering or Both?	60
12.	Videos within Requirements Engineering.	63
13.	Requirements Engineering for Product Lines and Families	66
14.	Conclusion	68

1. Introduction and Motivation

We SOPHISTS have been working as consultants and trainers in requirements engineering (RE) for more than 20 years. We support customers from many different industries, from pure software development for insurances and banks to the execution of hardware-oriented systems engineering projects in the automotive sector and the planning of building complexes. The focus of our work is the elicitation of requirements, derivation of good system requirements as well as the management of requirements. Furthermore we support in imparting requirements. Depending on the project conditions we e.g., use approaches that are based on the documentation of requirements or storytelling. We also work with other topics, which are closely related to requirements engineering: requirements engineering within a systems engineering approach, creating a foundation for requirements by defining business processes and considering special aspects that come along with requirements that apply to a particular version of the product. Additionally, we pick up on current topics (e.g., Smart Ecosystems, Digital Transformation) and trends (e.g., Videos in RE, Crowd-RE...) in our daily work. Furthermore, we focus on the establishment of requirements engineering as well as agile approaches.



Our life is strongly influenced by IT-systems – sometimes it even depends on them. They make our existence considerably more pleasant, they structure and co-create it. They provide information, support decision making and work processes, as well as automate processes. They also open up opportunities that we have not even dreamed of a few years ago. We live in smart homes, our cars consist of intelligent components, our home appliances are able to communicate with us and our smart phone provides us access to the rest of the world; anytime and anywhere. Industries manufacture their products in a “smart” way, agriculture, energy supply and the health care system, etc. perform better, because systems monitor and optimize the creation of value. To ensure, all of this will stay a dream for humanity and not turn

into a nightmare, it is crucial to make sure these systems do what they are supposed to do, i.e. what we expect them to. This is where requirements engineering comes into play. The complexity of the tasks systems execute is increasing and needs to be invented or identified, analyzed and imparted. The knowledge (the requirements) often needs to be documented and managed. Before that, the business processes that are supported by the system need to be understood and drafted up. The trick is to consider the manifold constraints. Requirements engineering in this context means to find the right approach for the right purpose.

In order to make dull theory a bit more exciting and to make it easier to comprehend the topics addressed in this brochure, we added some examples. They are based on the development of a smart home system (SHS).

Additionally, there are videos and animated graphics covering some selected topics on our homepage. The respective link is embedded in the following icon:



Many thanks to all SOPHISTS, who have contributed to this brochure – the entire “requirements engineering and management” book team, the technical and linguistic reviewers and the discussion partners who have contributed to the contents and the quality of this brochure.

2. What is Requirements Engineering?

The term requirements engineering can be loosely interpreted as follows: dealing with requirements concurrent to engineering principles. Working with requirements should therefore be repeatable, comprehensible and justified. As requirements engineering is usually part of development projects with constraints like time and money, something else matters: It should be executed appropriately. Our goal is to ensure that requirements engineering creates requirements exactly in the quality needed for further development.

These thoughts reflect in the following definition:

Definition of the discipline requirements engineering according to IREB [CPRE20]:

The systematic and disciplined approach to the specification and management of requirements with the goal of:

- understanding the stakeholders' desires and needs and
- minimizing the risk of delivering a system that does not meet these desires and needs.

Unsurprisingly, the term requirement is of pivotal importance in this definition. We SOPHISTs have found a definition of the term requirement that suits us well in practice, and has proven to be very comprehensible, comprehensive and sufficiently specific:

Definition of the term requirement according to SOPHIST:

A requirement is a statement regarding a characteristic or performance of a product, a process or the people involved in the process.

Obviously we interpret the term requirement broader than usual. On the one hand, it is more than the system to be developed demands (the requirement's subject of observation). On the other hand, we want to avoid stipulating a form of representation for requirements. For us, a requirement can be documented or imparted in various ways during a project. Here are some examples:

- classically, textually, possibly using templates,
- as a User Story or Epic,
- as a story using storytelling,
- model-based.



Furthermore, this definition of requirements engineering makes the so-called stakeholder the focus of interest. Usually, the stakeholder is being defined as a role with direct or indirect influence to the requirements. A direct influence in this context means, that a stakeholder provides requirements. (see paragraph 2.3 – “Sources for Requirements”).

The close cooperation between a requirements engineer (also referred to as requirements or systems analyst) and the stakeholders demands certain qualities of the requirements engineer. These qualities exceed the methodical and subject-specific abilities required in the application area of the system to be specified. Usually requirements engineering is concerned with communication between persons who have different goals, educational backgrounds and characteristics. These abilities have proven to be very important.

Figure 2.1 illustrates the abilities a requirements engineer needs according to IREB [CPRE20]. There can be found a distinction between basic competencies (left) and competencies that are highly relevant, especially for determining requirements (right).

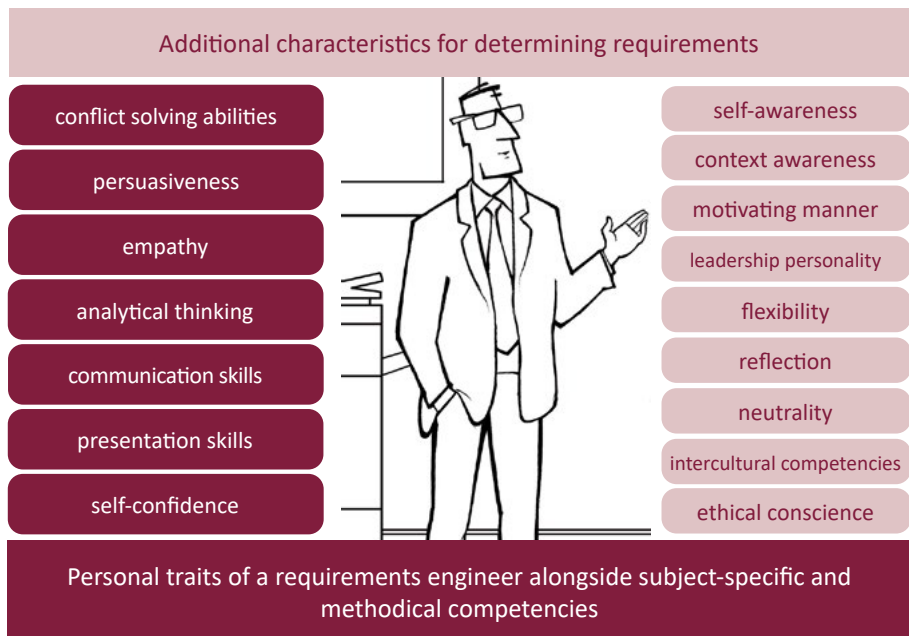


Figure 2.1: Personal traits of a requirements engineer

In the following paragraphs, we will present some basic knowledge that is essential for the work with requirements.

2.1 Classification of Requirements

Generally, there is a variety of options on how to classify requirements. Classifications should always have a certain objective. In this chapter, we will present the types of classifications that help us in our work with requirements and therefore seem valuable. Every requirement can be categorized according to the classifications described in this chapter. The classification allows defining tasks regarding the requirements engineering approach. For instance, sometimes the elicitation can be different for functional requirements and non-functional requirements; sometimes there might be a demand for finding the fitting upper requirement for pre-existing more detailed requirements.

Traditional classification according to types

The following illustration gives an overview of the different types of requirements that are most commonly mentioned in literature.

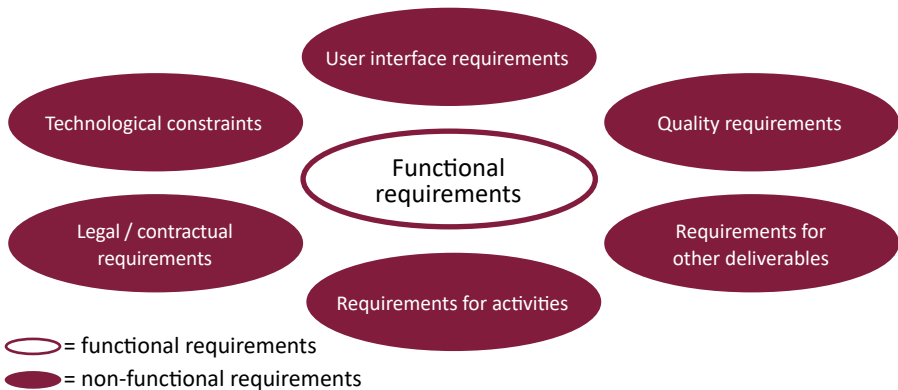


Figure 2.2: Types of requirements

The two most frequently needed types are:

- **Functional Requirements:** which functions does the system provide to a user or neighboring system under certain conditions?
- **Quality-of-service requirements:** they specify the desired quality of the system, mostly related to functions (e.g., the performance of a function or the availability of the entire system).

Classification according to legal obligation

The legal obligation describes the importance a stakeholder ascribes to the individual requirement. The following types can be distinguished:

To obtain a complete set of requirements, all levels of detail for requirements should

be specified in a way that provides enough information for everyone involved. However, this does not necessarily mean that every requirement needs to be broken down to the smallest level of detail.

Legal obligation	English keyword
Obligation	Shall
Wish	Should
Intention	Will

Figure 2.3: Proven keywords for legal obligation

Classification according to responsibility

An important distinction for requirements is the person responsible for them. As we will see later on, one of the purposes of Requirements Engineering is to take all the input for a development project, the source requirements, and derive reliable system requirements. This creates a relationship between these two types of requirements.

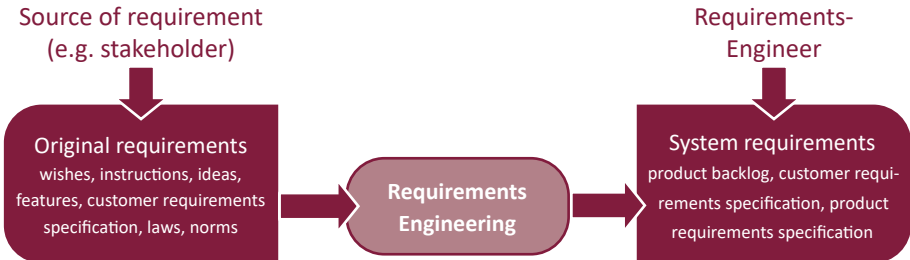


Figure 2.4: Inputs and outputs of a requirements engineer

Classification according to subject matter

The probably most important distinction for the classification of requirements is made based on the subject matter. Does the requirement refer to the considered system in the project, to one of its components or to a business process supported by the system? Is the requirement located outside of the considered system, in its context? Especially in the area of systems engineering, where several levels of the system exist, the requirements shall be allocated distinctly to a subject matter (see paragraph 9 – “Systems engineering”).

The characteristics of a requirement really matter. That’s why we wrote more about deriving good requirements in the chapter about analyzing requirements (see paragraph 4.1 – “Activities for analyzing requirements”).

Classification according to the level of detail

At first, this last type of classification seems relevant only in theory. In practice it can often be found in document landscapes. Requirements can specify other requirements i.e. they indicate a (technical) solution for a requirement. For example it can be required, that a smart-home-system shall unlock the front door as soon as a known person approaches. More details about which characteristics of the person the system shall check would be documented in more refined requirements.

This provides a clear classification of the requirements and therefore assures comprehensively a certain level of completeness of said requirements.

2.2 Quality of Requirements

As mentioned earlier, Requirements Engineering should be executed appropriately. This can be defined by the quality of the requirements that should be achieved for the development.

There are various different approaches for the quality of requirements. All of them have in common, that they define quality by a series of quality criteria. In the following, we have listed the criteria SOPHIST use to measure and assess requirements. In doing so, we differentiate criteria referring to a single requirement from criteria that apply for a whole set of requirements.

Requirements for classic approaches

For development projects executed in a classical approach the requirements need to have a high quality. The reason is that these approaches do not include an integral process step that aims to improve the requirements once they have been initially created. Instead they rely on the optimistic assumption that requirements engineering is finished for a set of requirements, as soon as it has been initially created.

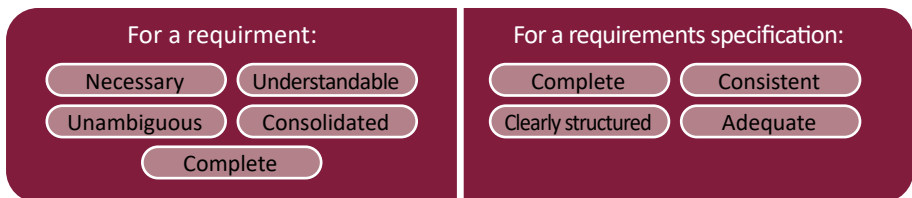


Figure 2.5: Quality criteria for requirements in classic procedures

Requirements for agile approaches

To determine the criteria for a single requirement in an agile development project (normally these will be User Stories) we apply the INVEST principle [Wake03].

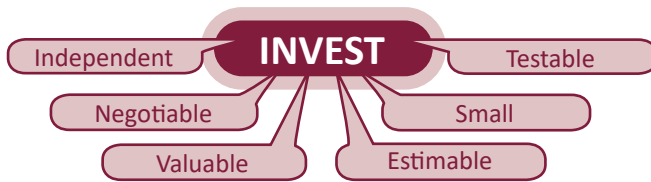


Figure 2.6: The INVEST-principle for a User Story

The criteria for a set of requirements, meaning a backlog, are barely any different from those that are supposed to apply for a requirements specification in the classical context.

2.3 Sources of Requirements

Essentially, there are three different sources of requirements:

- **Stakeholders:** Persons, organizations and institutions that influence the system a directly or indirectly.
- **Documents:** Laws, norms, manuals or other documentation can be used to gather requirements.
- **Systems:** It is often helpful to analyze a preceding system or a competitive product.

Most interesting is the group of stakeholders. They often come from different backgrounds and therefore have variable goals for the considered system with their requirements. In the following, there is an incomplete list of areas that can provide requirements or even be the trigger for a development project.



Commissioning organization

In a relationship between a commissioning and a contracting organization, the commissioning company is presumably the most prominent cause for a development project. It provides the contracting organization with money for the development (and maybe also for following tasks such as production). By doing that, it significantly determines the requirements for the future product.

Management of innovation/ portfolio /product

On the other hand, the areas stated above often are internal causes for development projects. They are responsible for the advancement of pre-existing products to remain competitive in the market. In a project executed for a customer they may, however, also demand additional requirements.

Problem/ change management

Problem management provides change requests to a system that most commonly have been identified in later phases of the life cycle (during production, transport, installation, operation). These can lead to additional or changed requirements. They might also demand a new development project. Changes of requirements during an ongoing development project are supported by change management.

2.4 Main Activities of Requirements Engineering

The activities a Requirements Engineer has to carry out can be roughly allocated into four main activities. These main activities will be discussed in detail later on. In this brochure we can only depict the most important tasks and aspects of the individual main activities. For a more detailed presentation, please refer to [Rupp20].

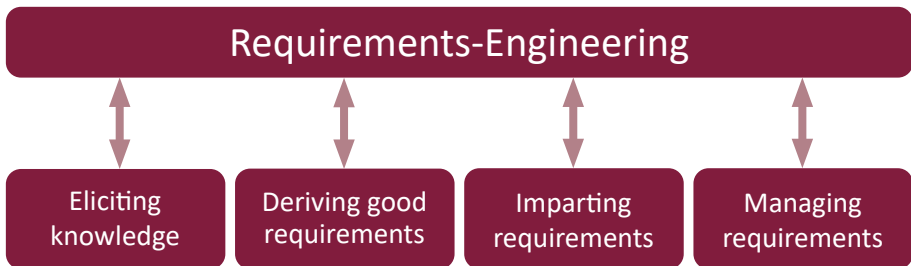


Figure 2.7: The four main activities of requirements engineering

Even though we present the main activities in the order mentioned above, the individual activities will always be repeated over the course of the system development. This could either be caused by an incremental view on the tasks or by the fact that during the performance of an activity the necessity for another activity grows.

Eliciting knowledge

The first main activity “eliciting knowledge” lays the foundation for further requirements engineering. If they don’t already exist, the course of the system development is set by defining visions and goals. Not all sources of requirements, in particular the stakeholders, are always known, so they must be identified as a basis for further activities.

Once the sources of requirements are known, the elicitation of requirements for the system to be developed can begin. Selecting the appropriate elicitation techniques for each specific development is an important success factor.

Deriving good requirements

In order to derive good requirements, first the previously identified demands shall be analyzed in order to get the most comprehensive idea of the requirements for the system to be developed. Depending on the process model, this idea must now be tested. Representatives of different roles in the development can review the requirements:

- From the point of view of the people issuing the requirements, the requirements are checked with regard to the fulfillment of the expectations of the system.
- For test the requirements are checked to see whether corresponding test cases can be derived.
- The development (especially the architecture) will check the feasibility of the requirements.

Depending on the application domain, further checks may be necessary. In the automotive industry, for example, it may be necessary to check compliance with functional safety requirements ([ISO26262]).



Imparting requirements

So far the main activities are covered and a good overview of the needed requirements is created. Still, the requirements shall support somebody's work, so they need to be communicated to the people involved. Here we assume that one either wants to document the requirements or share their knowledge about the requirements in another way (storytelling, videos).

If a collection of documented requirements shall be created, there are two alternatives:

- The system requirements can be documented by using natural language.
- The requirements can be documented model-based.

In our consulting activities a mixed approach between both alternatives always had the highest benefit and the best acceptance.

Managing requirements

As soon as considering the documentation of requirements, a new challenge is waiting; the management of these documented requirements. Some decisions have to be made:



- What kind of information do the requirement collections consist of and how are they structured?
- Who is allowed to perform, which actions in the requirements (assign rights and roles)?
- Which additional information is needed in order to manage the requirements? For example: current state of the requirement, information about variants of the product.
- How shall the traceability be ensured?
- Which versioning concept shall be the foundation for the requirements?
- Which information needs to be provided in the system development and which form shall it have?

We assign these and many other decisions and activities to the management of requirements in order to provide the required information in a comprehensible way at any time.

3. Eliciting Knowledge

Not only is the elicitation of knowledge the first, but one of the most important activities of requirements engineering. It includes the requirements themselves, but also other information such as sources, boundaries and business processes. It takes place in every possible scenario, every process model and for every refinement level of requirements and it is a little different each time. If the right stakeholders aren't identified, the context is set incorrectly or, based on the business processes the right knowledge isn't elicited of, requirements engineering is very likely to fail. This doesn't mean that all of these activities have to be completed at the beginning - elicitation is rather a process that endures the whole requirements engineering process.



A lot has changed in the field of requirements engineering in the past years. Digitalization and concepts like smart cities or smart rural areas have a significant impact on our social context and our human interaction. As a result, IT is making its way into many areas that were barely affected by it previously. Of course, most people already have a washing machine in their house that contains software, but the wishes of the users were often only guessed. Digitalization noticeably

intervenes in areas such as our healthcare system, the delivery of goods, the transportation system and communication. That's why it is important to understand the needs of all the different stakeholders. In our consulting, we always adapt RE methods to the context in which they are applied and to its stakeholders. When digitalization moves into all areas of life, the systems suddenly affect people who hardly had contact to IT and who aren't familiar with their role as stakeholders.

3.1 Goals, Sources and System Context

But first things first: Every project is initiated for a reason or purpose. It needs sources of requirements, and should have a defined framework to distinguish the necessary from the unnecessary.

Goals and goal finding

Setting the goals for the system and a project impacts the success of the development lastingly. Every system development should start with a definition of goals with an extent of about half a page. In practice we made good experiences by using a goal table or a product/project canvas. If goals are not documented, or defined unclearly, there is no foundation for the subsequent steps or activities. As a result, the requirements engineer has no specification of the goals the requirements must meet. Consequently the actual goals may be missed. The process of finding goals strongly depends on constraints. For inventing a new product, visionary thinking and

an open mind to all possible solutions are most important. If an old system shall be replaced, the focus should primarily be on possible improvements and their impact on existing functions.

Sources of requirements

Finding the relevant sources of requirements is crucial. Typical sources are documents, systems and stakeholders of course. A requirements engineer, needs to keep his eyes open for all imaginable sources of information for requirements. If required sources are not available, working with personas [Goodwin09] as fictitious representatives of real people, might be helpful.

System scope and context

The final activity before the actual elicitation of requirements can start is to define what actually belongs to the considered system (scope) and in contrast, what is outside of the system's boundaries (context) and thus can interact with the system. An inaccurate scope or context will lead to incomplete or incorrect and sometimes even to unnecessary requirements.

3.2 Elicitation of Requirements

The goal of the elicitation is to determine requirements that help developing a system that bring as much benefit as possible to the stakeholders. Thereby the requirements need to be adapted to the constraints of the project and be elicited with as little expense as possible. That's why for the elicitation there shall be an efficient balance between the risk reduction and cost explosion. In addition professional elicitation methods fitting to the respected sources are needed. The gathered knowledge is the foundation on which good requirements are derived, documented, imparted, and managed.

Precondition for a good elicitation

It can't be expected that stakeholders present the perfect requirements on a silver platter. Elicitation of requirements is hard work. These are the most important factors for a successful elicitation of requirements:

- Basic communication skills
- knowledge of representational systems of language
- knowing the strengths and weaknesses of each elicitation techniques
- analysis of the relevant constraints for the use of elicitation techniques
- selecting and combining appropriate elicitation techniques
- creating the right atmosphere for the use of elicitation techniques (especially for creativity techniques)

Criteria for the selection of elicitation techniques

Each elicitation technique has its strengths and weaknesses. They are only suitable for the use under certain conditions. Knowing these conditions and using them to select a suitable elicitation technique is decisive for the success of the elicitation of requirements.

- How great is the knowledge of the stakeholder regarding the subject of observation?
- How high is the motivation of the stakeholders to participate within the elicitation?
- Is the stakeholder an active person? Does he like to talk or are is he rather haptically minded?
- How is the local distribution of the stakeholders?
- What time are the stakeholders available?
- Are there any special group dynamics between the stakeholders to be considered?

The small selection of criteria above demonstrates that choosing the appropriate elicitation techniques depends on the experience of the requirements engineer.

Four groups of classical elicitation techniques

In order to elicit knowledge a variety of techniques has been developed, which can be roughly divided into four groups.

■ Survey techniques



(Questionnaire and interview) are the classics among the investigation techniques and are based on asking stakeholders about their wishes and needs in a targeted manner in order to derive requirements from their answers.

■ Observation techniques



(Field Observation, apprenticing, contextual inquiry) are used if stakeholders cannot express their knowledge in language or many implicit wishes are expected.

■ Document-centered techniques



(System archaeology and reuse) have their strengths when there is no knowledge carrier available anymore. In this case the domain logic can only be determined from the system itself and its documentation.

■ Creative techniques



(Brainstorming or brainstorming paradox) are used when innovative ideas are required.

New approaches/frameworks - co-creation-models, CrowdRE and living labs



In addition to the elicitation techniques just explained, there is a whole set of frameworks that combine several investigation techniques.

As a representative, we would like to single out CrowdRE. The attempt is to motivate an anonymous group of people (crowd) to cooperate by providing an access with the lowest possible threshold. There are many approaches on how the crowd can be won and how they contribute. In most cases, electronic means are the standard.

The goal of CrowdRE is also to find so-called unicorns (highly motivated stakeholders with a lot of knowledge) in the crowd of respondents. We will try to encourage them to a further cooperation.

3.3 Identifying Linguistic Effects – The SOPHIST Set of REgulations



In the work with requirements, undesirable linguistic effects can occur especially during the elicitation phase; in direct communication as well as in writing or reading requirements. The SOPHIST Set of REgulations helps to reduce undesirable effects and to create the basis for high-quality requirements. Still we have not yet explained what linguistic effects are and how they arise.

Linguistic effects

Every person perceives their environment differently. The totality of their perceptions forms the person's knowledge, which is further influenced by previous knowledge, social shaping and accumulated experience.

As soon as people communicate their knowledge, transformation processes take place. These processes depend on how the communicating parties assess their situation and interlocutors. Which previous knowledge do they expect from the other person, and how certain of an issue they are themselves? These transformation processes can result in a loss or falsification of information.

However, transformational effects can be detected and resolved – but only if the requirements engineer knows the possible types of transformational effects and their consequences. This is where the SOPHIST framework of REgulations comes into play. Essentially it is based on the meta-model of language and neurolinguistic programming (NLP) [Bandler75] [Bandler94]. Bandler and Grinder distinguish between three "types of transformation": deletion, generalization and distortion.

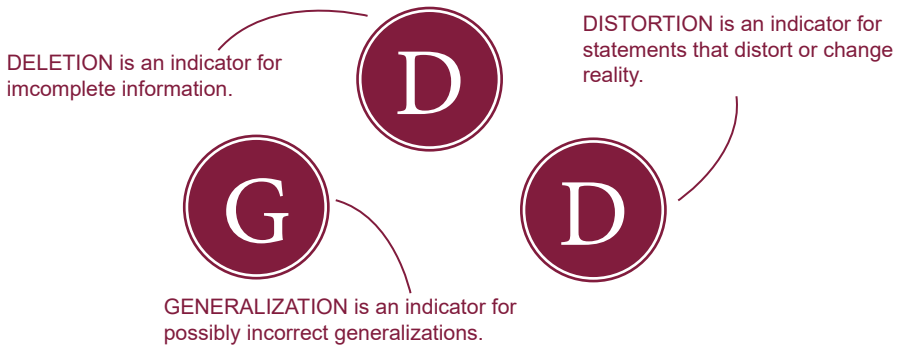


Figure 3.1: Classification of linguistic effects

The SOPHIST Set of REgulations

Fortunately, people proceed in a rule-guided manner when formulating knowledge in natural language. There are tells in written and spoken natural language statements, that help recognizing transformation processes. The SOPHIST Set of REgulations is based on these unconsciously applied rules and enables to find ambiguous and contradictory statements in requirement specifications in a defined and systematic way during system analysis.



Adding process details

It need to be checked if the relevant process details are already sufficiently named in the requirement. Question the verb (or a property word derived from a verb) that expresses the process using the typical question words: When? Whom? Where? How often? etc.

Decide whether the missing information is essential for the implementation. If it is, fill in the missing process details.

We have only provided one of a total of 17 rules to give an idea of these rules. It's not necessary to memorize all 17 rules at the beginning. One can pick out the two or three rules that are most important to him, and listen for the corresponding signal words. Then, if necessary, ask the right questions, to get the deleted information for example.

More information about the usage of the REgulations can be found in the following video:

Signal words	<p>Verbs: save, display, delete, calculate, assemble ...</p> <p>Adjective derived from a verb: saved file, shared document, configured schedule ...</p> <p>Adverb derived from a verb: transmit compressively, print system-controlled</p>
Description	<p>In order to describe a process unambiguously in a requirement, it is necessary that all information that is required for a complete explanation is available. This concerns verbs, or adjectives and adverbs derived from a verb. If there are unanswered questions about the process the missing piece of information must be found and added to the requirement.</p> <p>Specific questions need to be asked such as „By whom or what is the process executed?“, „How often is the process executed?“, „How do you, as a user, execute the process technically?“, „When, or under which constraints, is the process executed?“.</p> <p>Completing the specified process details can lead to time-consuming reformulations of the requirement. Often, however, additional requirements are revealed from the answers of those questions, or the requirements engineer realizes, that the original requirement must be refined and thus replaced by several more detailed requirements.</p> <p>Think also of aspects that are described by properties. They occur either as adjectives or adverbs. In this case, ask questions that determine specific information about the property.</p>
Example	<p>Original requirement: „If the identity check is not correct, the smart home system must display this.“</p> <p>Still unclear: What is being displayed? To who is being displayed? When will be displayed? How long will be displayed? Etc.</p> <p>Improved requirement: After the smart home system verifies the identity entered by the user, and if the identity verification is not correct, the smart home system shall display the error message „Access has been denied“ to the user for three seconds.</p>
Tips & tricks	<p>Process details are statements/information that concerns a verb/adjective/adverb. They can be analyzed by questioning them, but don't necessarily need to be redrafted. Not all process details always need to be included in every requirement. Some process details are already set by a precondition of the use case or are known with a previous requirement and then don't need to be redrafted.</p>

4. Deriving Good Requirements

Once the requirements of the stakeholders (in the following referred to as original requirements) have been gathered, requirements that are good enough to serve as a basis for development and testing must be derived.

In our experience, a common problem with original requirements is that they demand more than what the development object, the system, is capable of. Therefore, during analyzing the original requirements special attention shall be paid to the content that is derived for the system. In the following, we refer to these derived requirements as system requirements.

After the system requirements have been identified, there may be differences to the original requirements. The reason is that the requirements engineer

- interpreted the original requirements,
- made further stipulations,
- adjusted the requirements,
- added missing requirements.



These (and many other) reasons lead to the necessity of having the generated system requirements being reviewed by the stakeholders and of resolving possible discrepancies.

4.1 Activities for Analyzing Requirements

Using the activities presented here, one will generate many system requirements that may have been missing. We assume that each requirement refines another requirement. An exception to this is the most abstract level of requirements. These requirements stand next to each other and form the basis for the refined requirements. The outcome is many trees (mathematically called a "forest"), where the roots are formed by the most abstract requirements and the leaves are represented by the most detailed requirements that are no longer refined.

In a use case-based approach, the roots of the requirement trees represent either

- use cases of the system or
- categories of non-functional requirements that cannot be assigned to functional requirements.

An incomplete example is given in the following image.

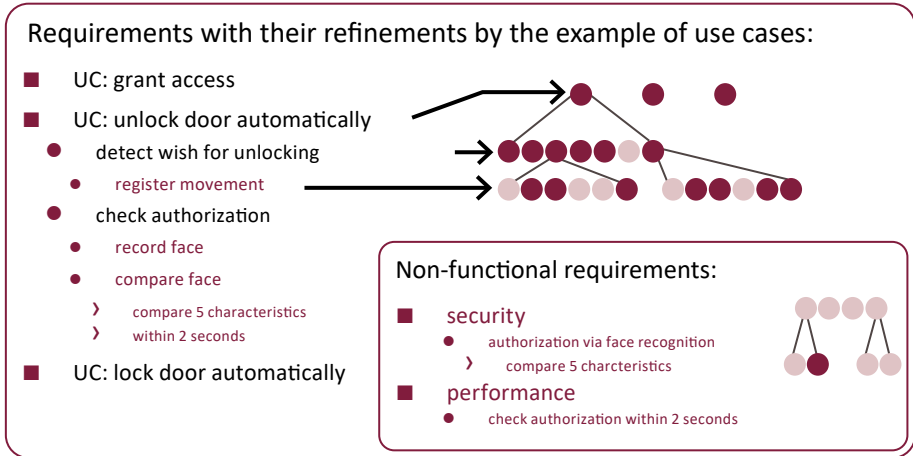


Figure 4.1: Connection between requirements

Each of the activities briefly presented below creates new system requirements from given requirements (original or already created system requirements), or modifies the given requirements.

The first step is to **separate requirements**. If necessary one original requirement shall be decomposed into several requirements in order to examine them separately in the next steps.

The second step, **extract necessary requirements**, can help to make sure, that separated requirements are actually pointed at the considered system. If this is not the case, the part that matters for the system must be identified for further consideration. These first two steps should be applied to all original requirements to create a good starting point for further analysis and to be sure that all original requirements are considered.

The next task is to **abstract the given requirements** until the top level requirements (i.e. the roots of the trees) are found. In the process, new requirements may be encountered. The existing requirements are to be placed into an abstraction/refinement hierarchy. The top level will be completed in the step **add missing requirements**, in which further requirements at a certain level can be defined.

The second last activity is to **refine requirements** and to decide which of the existing nodes in the trees should be further refined.

At the end of the analysis, the found requirements can be improved and checked regarding to their individual quality, e.g., to formulate them unambiguously.

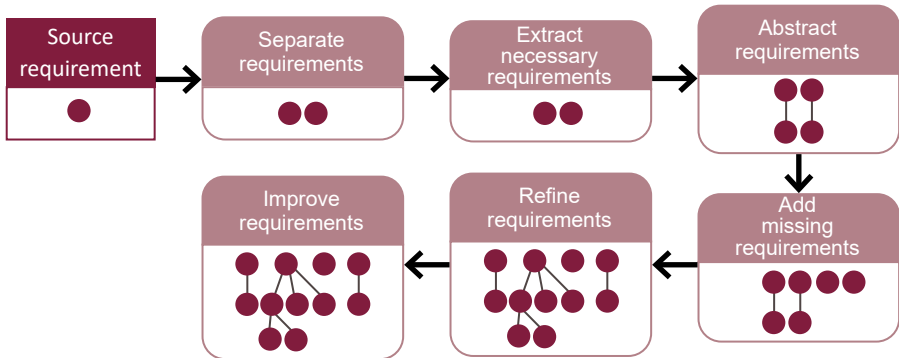


Figure 4.2: The analysis tasks in an overview

The structure created from the activities can become the foundation of the requirements collection of both, natural language and model-based requirements.

There are various techniques that support the activities presented here. Different questions can be asked, that lead to the system requirements. For example, asking “What’s the aim” of a requirement helps to find the more abstract requirements. A structured inspection of given interfaces can help finding missing requirements. In general, many activities are supported by the regulations of the SOPHIST Set of REgulations (see paragraph 3.3 - "Identifying linguistic effects - The SOPHIST Set of REgulations").

In practice, while executing the individual activities, knowledge that goes beyond the information in the original requirements is needed. In order to fill these knowledge gaps, stakeholders or customers need to be asked. Alternatively assumptions can be made, that need to be double-checked afterwards.

An exemplary application of the activities for analyzing requirements can be found in the following video:

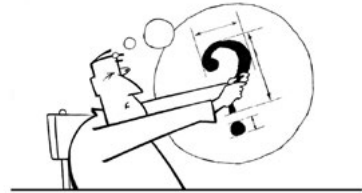
Please note that these activities need to be embedded into the RE process. Among other things it has to be determined when to carry out the analysis for the different parts of the original requirements. Furthermore, it is important to define termination criteria for the individual activities. Otherwise more resources than needed may be used into the analysis of the original requirements.

4.2 Validating and Consolidating Requirements

After the requirements are derived from the original requirements, it is time to ensure that they reach the defined quality criteria by validating and consolidating them and that every stakeholder is happy with the requirements.

Validating requirements

Incorrect requirements impair the development activities. The later a mistake is detected, the more changes have to be made - in architecture descriptions, in test artifacts or maybe even in the source code. When validating requirements, it is advisable to first define the goals of the validation. Later on, the selection of the respective validation techniques depends on them. After preparing and performing the validation, the results can be incorporated. Depending on the extent of the changes, a possible new validation may be appropriate.



Basically, it can be distinguished between automated validations that are performed repeatedly and validations at specific milestones during the development.

Consolidating requirements

In the course of requirements engineering, especially during the validation of requirements, discrepancies can occur at many points. These range from technical misunderstandings to serious personal conflicts that cannot be resolved without any additional help.

The conflicts that we mostly have to deal with in the RE process describe incompatibilities of requirements that are based on conflicting perceptions or competing goals of the stakeholders.

The requirements engineer's task is to identify these conflicts, analyze them, and determine how the conflict can be resolved, together with the stakeholders involved. After a conflict resolution, it is advisable to document the conflict as well as the process of finding the solution, in order to be able to work out a solution for similar conflicts with less effort.



For each of these steps, there are a couple of techniques to choose from, that are based on the constraints of the project (e.g., availability of stakeholders to resolve the conflict). For conflict resolution, there are techniques ranging from building compromise to the pull rank. Especially in the step of identifying conflicts between requirements, a structured and comprehensible documentation of the requirements will help.

SOPHIST

Competence and expertise

par excellence

Method inventor

Speaker

Author of books



Consultant

Coach

Trainer

We offer you:

We support you competently, energetically and expensively both in adaptation of your development processes and methods and in the implementation of your project.

Our customers include many world-renowned companies. The large number of positive opinions and project reports from our satisfied customers speaks for itself.

Take a look at **www.sophist.de/referenzen**

Our services:

- Identify, exploit and introduce potential for improvement taking into account the constraints in your organization
- Elicit, analyze, convey and document requirements and architectures appropriately
- On the way in simple software applications up to complex systems
- Work in agile and adapted way

All of this and many other topics from the world of requirements and systems engineering we offer you in the form of consulting, coaching, training and lectures.

How can we help you?

We would be happy to elaborate a concept with you to provide you with the best possible support for your project.

Contact us without obligation:

+49 (0) 911 40 900 - 0

heureka@sophist.de

5. Documenting and Imparting Requirements

Whether with or without a specification, model-based or natural language - whether as a narrative, backlog, or specification; in order to impart the requirements to others well, the imparting must be planned and various influencing factors must be considered to select the appropriate technique. After all, we usually don't elicit and analyze requirements for ourselves, but for other roles involved in the development process, such as development, test, system architecture, and many more.

Accordingly, an important task for requirements engineers is to impart the requirements to other people in a way that they understand them and no misunderstandings or misinterpretations occur. The requirements engineer should think about how he wants the imparting to take place. For example, he can write down (document) all the requirements and give them to the recipients to read. Or he can talk about the requirements up to a joint playful experience of the requirements. Our project experience shows that successful imparting usually consists of a well thought-out combination of different techniques.

5.1 Imparting Requirements without Documentation

Requirements and user stories can also be communicated without classic requirements documentation. Especially in the agile world, we made positive experiences with techniques that involve less documentation and more communication (e.g., user stories or storytelling). However, requirements documentation based on models or natural language and the options presented below are not mutually exclusive. In fact, one can certainly combine them to take advantage of the strengths of both options.

Storytelling

Who enjoys listening to a monotone lecture if they could enjoy the same content wrapped up in a good story? People have always been fascinated by stories. Since the beginning of time, knowledge has been passed on through stories. It's not surprising that storytelling is also used in requirements engineering. We use different types of stories.

- **Background stories** tell something about the context and the usage of the system - they give important background information.
- **Personality stories** introduce a persona and make them tangible.
- **Conviction stories** transport the initial situation at the very beginning of requirements engineering and motivate why the system is needed.
- **Explanation stories** clarify individual processes and behaviors of the system. From our experience, they are used most frequently in RE.

User Story and Story Mapping

User stories describe desired functionalities resp. properties of a system from the perspective of the person who needs the functionality or property. In the agile world, the imparting of requirements by means of user stories is widespread. In addition to documenting the content of the user story (**who** wants **what** from the system for **which purpose**), a user story represents a communication promise.

Mediation with user stories roughly works according to the following pattern:

- Formulating the requirements in user stories.
- Discussion with the people receiving the requirements, based on the formulated user story
- Joint agreement on the contents of the user story

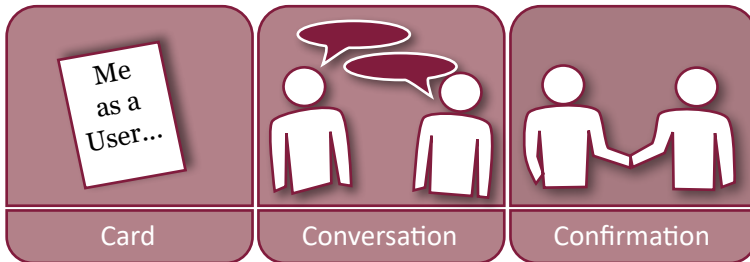


Figure 5.1: 3-C-Model according to Ron Jeffries

User stories generally describe only very small functionalities, so there will be a large number of user stories over the course of a development project. In order to maintain an overview, they can be related to each other with the help of a story mapping.

Prototypes

Prototypes can be used in many areas of system development; they are suitable for eliciting requirements, for validating them, but above all for imparting them. As diverse as their possible applications are, so are their characteristics. We've made good experiences with the following prototypes in system developments:

- Wireframe: a schematic representation of the elements of a user interface.
- Functional prototype: A preliminary implementation of a function regardless of e.g., the performance or presentation of results.
- Mock-up of the user interface: An extension of the wireframes; the design of the user interface is already recognizable.

Pictures

A picture is worth a thousand words... but are a thousand pieces of information relevant for imparting one's knowledge? This is exactly the dilemma one will face when using images to impart knowledge. Many people find it easier to capture ideas and wishes in a picture than to just explain them by using language. Therefore, a picture can be of great help in storytelling. Moreover, research has shown that pictures can be remembered much better [Wolfe10], which is crucial for imparting knowledge. Of course, the use of pictures is limited and not every aspect can be expressed well in a picture. But especially when it comes to spatial arrangements, impressions, i.e. the parts of a system that can be seen and represented visually, pictures help enormously in imparting knowledge.



Talking about image in this context, we mean an informal visual representation, e.g., a drawing of the house on paper to discuss the positioning of the surveillance cameras. It's not a formal or semiformal representation like an architectural model or a sequence of events noted in a UML diagram.

Create joint artifacts

Another way to impart requirements is to jointly create additional artifacts. From our experience, the joint creation of test cases for the requirements is particularly suitable. However, it is also imaginable that other artifacts, like architecture or design documents or operating instructions, are created jointly. Especially if one does systems engineering, it might be useful to create architecture documents (see paragraph 9 - "Systems engineering"). It helps to pick artifacts that have to be created anyway to avoid generating further effort and to be able to use the created artifacts in subsequent process steps.

5.2 Imparting Requirements with Documentation



Let's talk about the most common imparting technique now: documentation. Documentation can be found in a classic commissioning relationship in the form of a specification sheet or requirements specification, but also for communication between departments within an organization. Furthermore, this technique becomes almost inevitable when one wants to make requirements available for reuse.

The representations: Natural language vs. model-based requirements

When talking about documentation, we cannot avoid the terms "natural language" and "model-based". Both types of documentation have advantages and disadvantages.

Natural language documentation is distinctive in that no learning of notation is necessary, since everybody understands language. Natural language is also suitable for documenting all types of requirements. However, care must be taken, because natural language is often ambiguous or misleading.

On the other hand, model-based requirements documentation is well suited for looking at the system in isolation from different perspectives, e.g., the purely structural view of the terminology/ information/ data to be processed, the functional view of workflows/ system processes, or the state-oriented behavioral perspective that illuminates system reactions to events, among other things.

Due to the compact presentation, which is unambiguously understandable for the trained reader, misunderstandings can be avoided. However, the disadvantages are obvious - the corresponding notation must first be learned and understood by all participants.

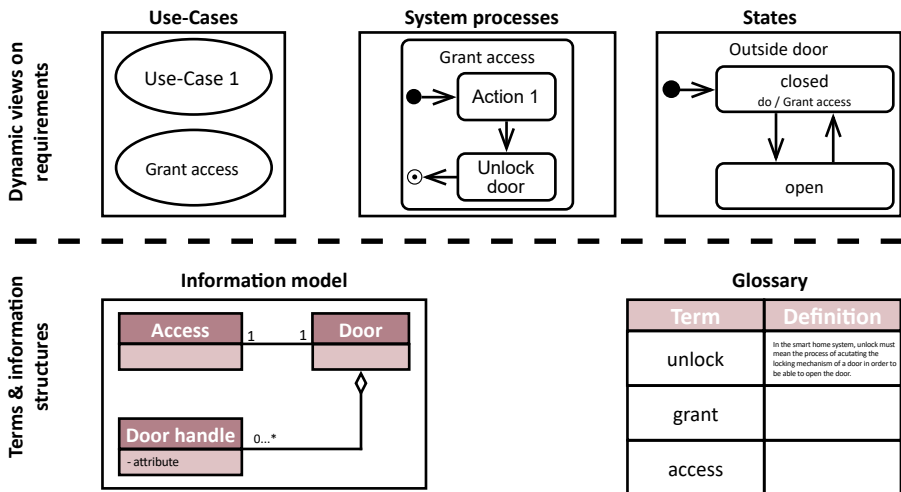


Figure 5.2: Views on requirements

One type of diagram for its own is usually not sufficient for representing complex issues in models. Besides, models are not universally applicable. A common approach is to combine model-based and natural language requirements to take advantage of both forms.

A prime candidate for natural language documentation - the FunctionalMASTER

From the area of the natural language documentation we offer an approach that in our opinion is easy to use. The SOPHIST sentence template, also called the SOPHIST requirements template, is a blueprint that defines the structure of a single requirements sentence. The structure of individual requirements is standardized. So it's possible to determine at first glance, whether important components of a requirement are missing. Especially when specifying in a foreign language, a predefined requirements framework can help to overcome uncertainties.

Using the requirements template is easy to learn and reduces unwanted linguistic effects since the syntax for writing a requirement is already provided. Compared to arbitrarily formulated prose requirements, the quality of the requirements increases significantly after the first application. The SOPHIST requirements template for functional requirements has become an integral part of most requirements engineering processes in companies and is referred to by us as the FunctionalMASTER. Based on our practical experiences, we have further developed the concept and defined other templates in order to cover non-functional requirements and conditions with our template. More information about our new templates can be found in our book Requirements Engineering and Management [Rupp20].

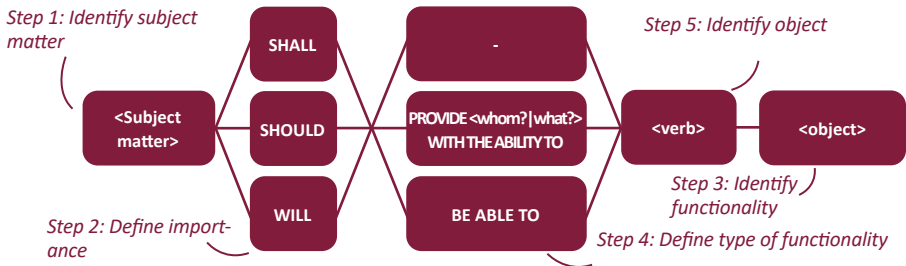


Figure 5.3: Functional MASTER

Our project experience has shown that the first steps in dealing with the templates are not so easy in the beginning. Writing the first requirements according to the template feels clumsy and more effort is required at first. However, the effort mostly comes from the fact that one has to reflect on knowledge elements that end up in the sentence due to the template, which therefore improves the requirement immediately. Moreover, in the beginning one will think about which template to choose: Is the requirement for a system that is supposed to do something on its own? Is the user or an interface involved? If one manages to get through the first few hours, he will find that templates are an effective way to write good requirements quickly and professionally. We would be happy to support with training, workshops or through consulting.

6. Requirements Management

Requirements management includes all the processes that support the main activities of RE and the further use of requirements. In the following chapter, we will give an insight into the world of requirements management (RM) and practical tips regarding the question, to which extent one should perform which activities of requirements engineering.

Why dealing with managing requirements at all?

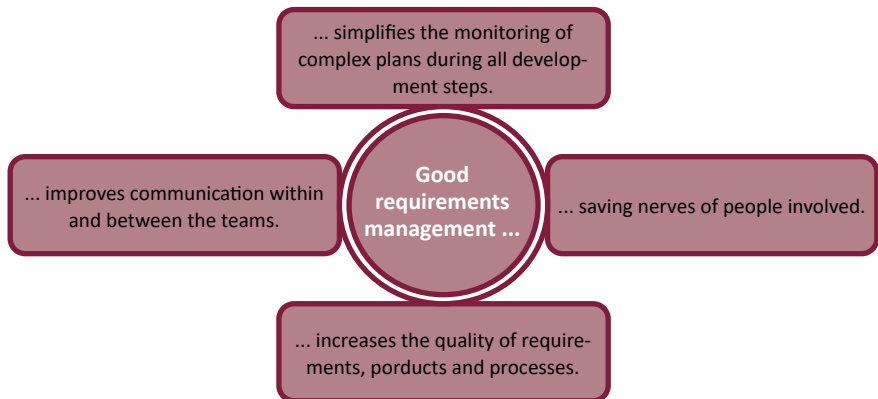


Figure 6.1: Good requirements management pays off

We specified the abstract reasons from figure 6.1 with two prominent representatives.

Requirements are changing

As requirements change frequently over the course of the system development, it is necessary to navigate well within collected requirements. Changes may range from small repairs, such as spelling errors, to complex changes that include extensive revisions of entire sections of a specification. A RE concept should contain a structured approach on how to deal with so-called “change requests”.

Requirements are reused

It should always be kept in mind that requirements are never collected for their own sake, but that stakeholders, such as developers or testers, have to be able to read, understand and work with them. Requirements engineers must therefore ensure that comprehensive information is presented clearly in the requirements collection (requirements specification and/or product backlog).

6.1 How much Requirements Management Makes Sense?

The importance of requirements management within the development process is directly related to the parameters of the project. There is no formula that describes how much resources should be spent for RM, but the following factors can help to estimate the amount of RM needed:

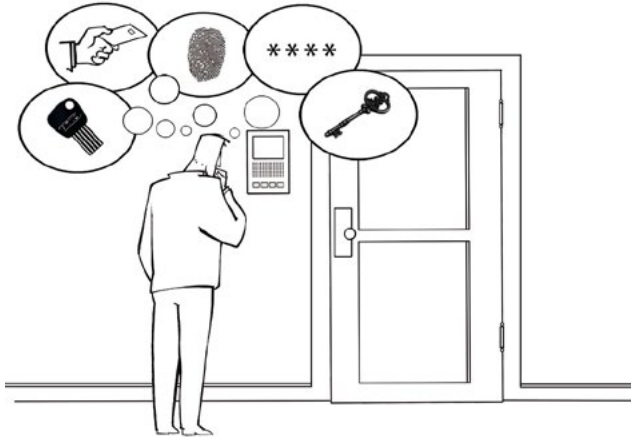
- number/ scope of requirements and further information
- expected lifespan of the product
- frequency of the rate of change
- number of people involved in the process
- availability of the stakeholders
- quality demand on the system
- degree of reuse
- complexity/ complicatedness of the development process
- heterogeneity of stakeholder opinions
- number of releases to be developed
- existing tool environment for requirements management
- approach used in the project
- external requirements (norms, certification process or corporate guidelines)

From our experience, we recommend not to fall into a lethargy of thinking that “It has always been this way” when analyzing external conditions. Try to find out these two things instead:

- Which constraints are unchangeable and which ones can be influenced?
- How resp. by what can external conditions be modified?

In practice, it has been shown that even minor changes to existing constraints can cause a big alleviation or improvement. In the following chapter, we will take a closer look at some of the most important aspects of RM.

6.2 Versioning and Baselines



As requirements change over the course of a project, introducing the versioning of requirements has proven to be a useful tool in order to understand at a subsequent date, how requirements have changed over time.

When creating a new version of a requirement, the first step is to copy the requirement. The old requirement is kept and linked to the new version. The new version is given a new version number. The old requirement is entered into the history of the requirements specification. The new version can now be edited. This procedure ensures that no information gets lost. Many tools developed specifically for RM support versioning. This is one of the reasons for using a professional RM tool.

Moreover, versioning also helps to plan releases and changes of requirements. In doing so, a specific state of the requirements is determined on which one can draw on at any later given point. This selection of requirements is referred to as “configuration”. If a configuration includes all the requirements for a release, we speak of a “baseline” instead of a “configuration”. Each configuration and baseline can be given a unique name for identification purposes.

In order to find out which requirements are part of a configuration or a baseline, a traceability concept is required.

6.3 Traceability

Definition of traceability according to SOPHIST [RUPP20]:

Traceability is the ability to track the connections and dependencies between information that arises at any time during the analysis, development up to the disposal or replacement of a system.

With traceability according to the definition above, it for example can be found out, when a requirement is changed, which other requirements are affected, which requirements are necessary for the development of a system function or which test cases cover these requirements.

Traceability creates a basis for effective and high quality requirements management, because it supports the following aspects:

- Verifiability: Have all the goals, agreed requirements, test cases etc. been implemented? Were all specifications met?
- Identification of dependencies: Which effects does a change of a requirement have on other development artifacts?
- Reuse: Which artifacts from the development process are used in other projects?
- Comprehensibility and overview: How has the system developed and changed? Which expenses must be expected for troubleshooting?

A traceability model can define which traceability is needed, who maintains it at what time and in what way and how these traces are supposed to be used over the course of a project and beyond. Our experience has shown that it should not be underestimated, how much effort it can take to create and maintain traceability regarding the managed information.

6.4 Change and Release Management

Frequent and complex changes to systems need an elaborate procedure in order to cover all processes - be it by collecting change requests, planning of releases or rolling out implemented changes. Suitable methods can be assigned to the discipline change and release management.

The following figure gives a context overview

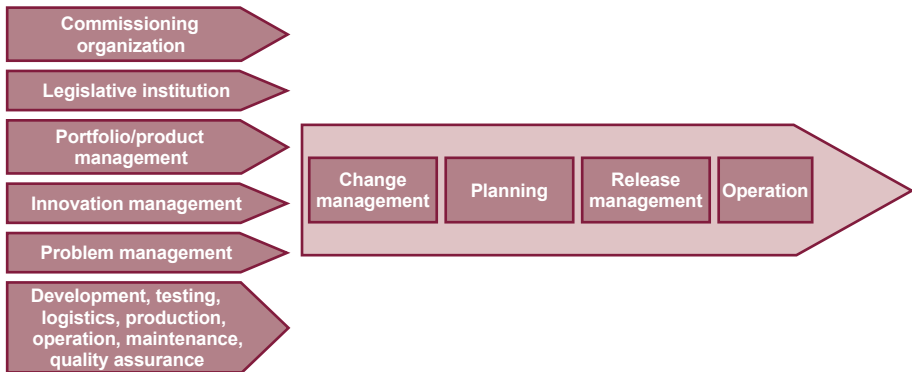


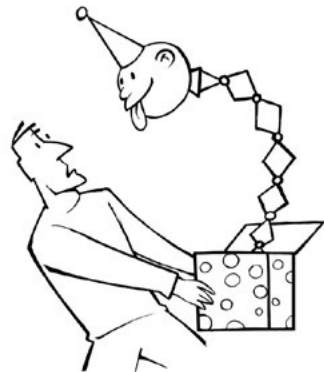
Figure 6.2: Change and release management

On the left side, the potential sources for changes can be seen - explanations to each mentioned source in paragraph 2 - “What is Requirements Engineering” can be found. On the right side, the process of a change can be seen. The process goes from change management, over the implementation, to the transfer of the change, into the operation of the system.

Change management

Change management controls the life cycle of all changes with the aim to introduce the changes into the development process in a controlled way. The tasks of change management include

- performing impact analyses,
- assessing changes,
- prioritizing changes,
- planning changes,
- and communicating the acceptance or refusal of changes.



Release management

As soon as planning, termination and controlling of builds and tests as well as the integration into existing systems are pending, release management comes into play. The release manager has a special role here, controlling that all deadlines are met.

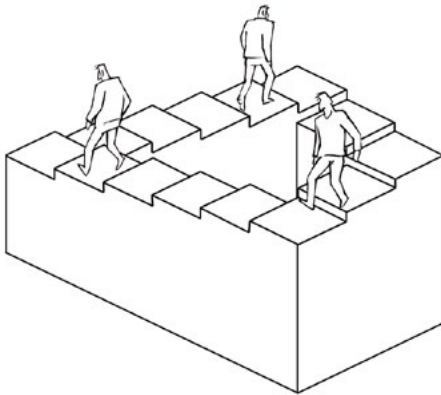
Once the implementation is complete, the new system must be delivered to the customer. The content of the release can vary a lot - for example, a new product version may be created when many innovations have been implemented, or a recall may be started when a critical bug was fixed. It's important to provide a service hotline, training staff, manual authors and all other affected roles with the necessary information in time, e.g., in the form of release notes, to prepare them for the changes.

7. Establishment of an Improved Requirements Engineering

Requirements engineering is part of every development project, but not always in an effective, efficient and satisfying way for everybody involved. If the requirements engineering of a company shall be improved, this is the right place to be. Changes may concern the process, the methods and also the tooling or all at once. Regardless of whether one wants to go from a waterfall-like to an agile requirements engineering or applies the last optimizations to their requirements engineering, the change should always be systematically.

7.1 Changes within an Organization

The British social philosopher Herbert Spencer created the expression “survival of the fittest”. This idea doesn’t only apply to human, but also to companies. Markets change, just as the demands of customers and partners do. Those who fail to keep pace with technological progress will soon lag behind the competition. Especially topics like digitalization can influence markets dramatically. That’s why systems and development processes need to adapt by considering new insights and options. This adaption also influences the requirements engineering.



However, adaptation also means questioning the existing and change always means letting go of something familiar and of the sense of security offered by the known. Consequently, change is always accompanied by fear, concern or doubt.

The need for safety is rooted deeply in human beings. New things that are going to replace the familiar are often perceived as a threat. Our experience shows that especially the fear of failure, the hesitation to learn something new and to make mistakes in the process is

omnipresent. For a successful establishment, in the beginning an awareness of the problem is necessary, a need for action and a motivation to change. If all of this exists, it is time to start looking for an improvement idea and, once it has been tested, to establish it.

7.2 The Establishment is a Project!

Such improvements must be established carefully into an organization. It should be put as much effort into the establishment as into a development project. It doesn't matter if a new procedure, methods or a new tool is to be established not even if they have to be invented and tested first.

We have seen it over and over again; during the analyzing, people are surprised of the guidelines that limit the solution space. Not only are unknown guidelines identified, just as often guidelines are considered that actually don't exist.

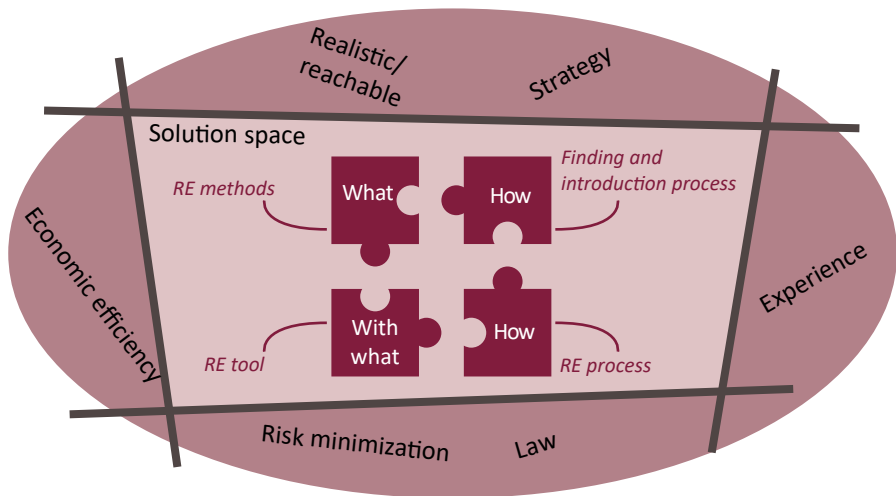


Figure 7.1: Guidelines of a RE implementation

7.3 An Agile Design of Changes

There are different approaches for improving requirements engineering. Our approach, which we often use in our projects and which is suitable for many situations, is based on agile ideas.

We made the experience that our customers' wishes are constantly changing. We can successfully meet this dynamic with agility.

Some of these changes are not content-related, but concern guidelines or new technologies and lead to a different way of working.

However, we can also deal with this dynamic by adapting our agilely developed approach.



Online/Remote Trainings

Your SOPHIST training - almost anywhere in the world

Online/Remote trainings are specifically designed to deliver knowledge and skills via the internet. The unique and carefully thought-out elaboration - in terms of content and didactics - as well as a maximum number of participants of 12 persons ensures a perfect online knowledge transfer.

For **Individuals** and **smaller teams**, our „open trainings“ are perfect. And due to the modular structure, the combination of different focal points and the possibility of individual adaption, online/remote trainings are also perfectly suited for internal company trainings of **complete teams**.

Of course our well-known CPRE certification training courses are also available in this format.

... no matter where

To make this approach work, we start with a few basic assumptions that must lie within the guidelines that have been set.

- The team itself must have authority over its working methods, i.e. it must be allowed to make independent decisions within the set limits.
- Everybody needs to know, aim of trying out new or modified methods is to investigate them and to learn something - and that results may be discarded.

Our experience taught us, that in order to carry out such an approach, there are two aspects that increase the probability of success enormously. First, only teams shall be considered for team selection that above all have a need for action and not only signal willingness for change. Second, there should be at least one "Elvis" in the team, because no one has more imitators; everyone wants to be like him. The success of imitational learning, acceptance and trust in advance facilitates the later spread and introduction of the new procedure. In the following, we will explain the individual steps of the process presented (see figure 7.2).

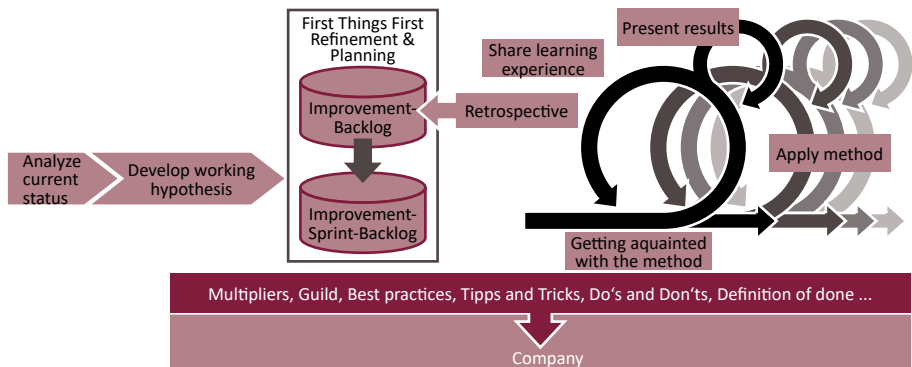


Figure 7.2: Overview over an agile procedure for changes

In the beginning, the current status is analyzed and a working hypothesis is developed. The focus lies on the current working method of the team. The teams examine the requirements engineering methods and tools and, above all, the friction losses that exist within the team and to external partners, e.g., the stakeholders. The resulting ideas for improvement are stored in an improvement backlog. After transferring them to the improvement sprint backlog, the training for the method starts. The team decides whether the entire team or only those participating in the experiment are trained. While the method is being applied, the team gains experience in applying the method under real working conditions. At this point the team can decide, whether they want to try out the innovation in one piece (workshop character) or continuously. During this time, the participants receive any methodical help they request. This can be a further training, a review of the results, coaching, sparring partners etc. The best time to check with the team whether there is a need for help and how the participants of the experiment are progressing is to ask them

in the daily meeting. To evaluate the experience gained, the results are presented to the stakeholders and their feedback is requested. The team sees how easy or how hard it was to apply the method and it hears from the experimenters when they share their learning experience. This allows the team to reflect in the retrospective on the process and results. The measures derived from the retrospective fill the improvement backlog again.

The team that has found an improvement can talk about it and their experiences to other teams. By doing it, they take the rollout partially into their own hands.

If there is not only one team or topic to be improved, this agile approach can be scaled with a usual framework such as SAFe [SAFe].

7.4 Work Packages of an Establishment

To ensure the success of implementations, it is worthwhile to plan and work out the most important steps using the following concepts.

- Marketing concept: Plan on who shall be excited about the idea.
- Concept for imparting knowledge: Structure the systematic build-up of knowledge.
- Piloting concept: Find the criteria for when and what will be tested by which pilot project.
- Migration concept: Think through how to deal with the existing (e.g., existing specifications).

What matters at all these concepts is that the execution of the measures is monitored and metrics are used to measure their success. For example in case of a marketing concept, it is worth defining the goal of how many people shall be reached with the information, in advance. In regular intervals it shall be checked how many of the targeted people have already been convinced of the new ideas. This is the only way to know whether the executed measures have been successful.

8. Requirements Engineering and Agility

The word “agility” is on everyone's lips these days and currently it is impossible to imagine the area of system development without it. Therefore, we must ask ourselves the question: What about requirements engineering in agile system developments?

First of all, we can clearly say from our experience, that requirements engineering is also carried out in agile approaches. Many things are called differently and, above all, are done at different times. However, in an agile context, there are often additional tasks for requirements engineers that are usually handled by other roles in classic system developments.

There are various agile approaches or frameworks. The best-known framework is scrum. Typical for all agile approaches is that the product is developed step by step. In many iterations, small increments of the product are developed, which results in a big picture. After each iteration there is supposed to be an outcome, which can be shown to the stakeholders so they can give feedback. Obviously the feedback given must be integrated into the requirements of future iterations.

At the beginning it is better not to think too much about very detailed requirements that will be implemented at a later date. There will be several changes due to feedbacks. That's why in agile contexts we go with the approach of just-in-time requirements engineering. This means we work with the requirements that are about to be realized.

This creates new challenges. For example, there are more dependencies between requirements to think about. It shall be avoided to demand something in one iteration that conflicts with requirements from previous iterations. To deal with this challenge, we pay attention to a clever splitting of the requirements in order to have no dependencies if possible (keyword “independent” from the INVEST principle, see paragraph 2.2 - "Quality of requirements"). We also use techniques from the classical world of requirements (e.g., models) in order to make the connections visible.

Roles of a requirements engineer in an agile context

In an agile environment, the term requirements engineer often doesn't exist. However, the job still has to be done. Even agile development cannot go without eliciting the wishes of the stakeholders. Their knowledge needs to be processed and imparted for the development team. In agility, it's usually the job of the product owner. However, the product owner can also be supported by other specialists, who are called proxy product owners, business analysts or sometimes even requirements engineers. Obviously product owners have more tasks than the typical requirements engineering activities. They also handle issues like release plannings, prioritizations, and decisions about the scope of the product to be built. These are typically not classic requirements engineering tasks. So obviously, there is some additional work for requirements engineers in agile contexts.



Brochures



www.sophist.de/wissen-for-free

Requirements engineering activities in agile contexts

In this brochure, we have already discussed the main activities of requirements engineering. These are eliciting knowledge, deriving good requirements, imparting requirements, and managing requirements. We will take a closer look to requirements management later and will start with the first three main activities.

Eliciting knowledge: The equivalent in the agile world can be found easily. For requirements engineers or product owners the work isn't any different to the one in the classic world. Product owners also think about goals, stakeholders, or system context boundaries.



However, agility brings new, great techniques (such as vision box, news from the future), which can also be used in classical development projects. Product owners, however often face one special challenge. They can define a system and context boundary, but the boundaries will be very "unstable" over a long period of time, because the development team deals with the requirements only step by step.

Deriving Good Requirements: In agile projects good requirements need to be derived from stakeholder statements, too. However, the requirements often look different because of different working methods. Usually the requirements are not written down in every detail.

Instead, we describe the requirements in a more rudimentary way in order to talk about them. Rough user stories are more likely to be created. We use quality criteria to tell how good requirements are. In agile contexts we use different criteria than in classical ones. We often take the INVEST principle as the quality goal of our requirements (see paragraph 2.2 - "Quality of Requirements").

Especially in this activity, cutting of user stories is an important topic. Success often depends a lot on proper cutting. There are a several cutting techniques and criteria that have worked well for us.

Imparting Requirements: In this main activity there are many differences between agile and non-agile development. In the agile world, requirements are imparted primarily through conversations in refinement meetings. So there are no longer the typical documents in which one can read the requirements. Instead there is a collection of requirements in a product backlog. These requirements can be described in different ways and communicated using special techniques. There are better ways to communicate them than a disorganized discussion. We often use techniques such as storytelling, creating test cases in a group, or videos to impart the requirements.

The requirements collection in agile contexts

In classic approaches usually a requirements specification is created (e.g., a specification sheet). At the end of the requirements analysis, this contains all requirements in the desired quality and in the required level of refinement. In agile contexts, we use a product backlog for the collected requirements. All the requirements known at a certain time can be found there. But - not all requirements for a product can be found there. And also not all requirements fulfill the assigned quality criteria and the required level of refinement. That only applies to the requirements (or better product backlog items), which are realized in a soon iteration. In addition to these characteristics, there is one more major difference to a requirements specification.

While the requirements in a requirements specification are organized and sorted by technical correlations, in a product backlog they are sorted by priority. This means that the requirements at the top of the product backlog currently have the highest priority. It is quite possible that these do not have any technical correlations.

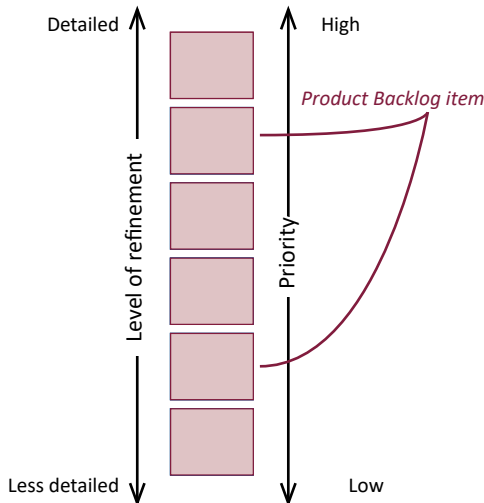
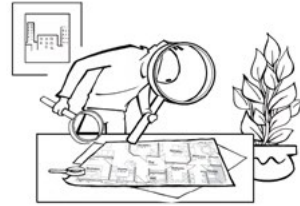


Figure 8.1: The product backlog

This makes the work on the content of the product backlog items more difficult. However, the problem can be handled by using various techniques. We successfully use different types of story maps or work with other forms of documentation in addition to the product backlog. There are many options for making requirements engineering profitable in the agile world.

9. Systems Engineering

Within a systems engineering approach an organized requirements engineering becomes decisively more important. Not only because the system analysis at system level lays the foundation for the overall development, but also because working with requirements is reflected at all levels of the considered system.



Before we present these statements in more detail, there are some key terms that relate to systems engineering to define.

9.1 Definition and Purpose

The first definition addresses the term systems engineering.

Definition systems engineering according to INCOSE [INCOSE20]:

Systems engineering is an interdisciplinary approach and mean to realize successful systems. It focuses on the definition of customer needs and required functionality early in the development cycle. As well as on the documentation of requirements, subsequent design synthesis and system verification considering the complete problem.

Let's take a closer look at a few terms in this definition.

The "complete problem" refers to the consideration of the entire life cycle of the system. Therefore, we don't only consider requirements that address the use of the system. From the very beginning of development the needs of production, transportation, storage, and installation all the way to the disposal of the system must be included.

Obviously another central term in the definition is the system.

Definition system according to SOPHIST [RUPP20]:

A system consists of several parts. The visible behavior and properties result from the interaction of these parts.

Another important activity of system engineering is also derived from this: The decomposition of the system into its components. In the following, we will refer to this as "system architecture", in which requirements will play a role again.

Many of the systems we encounter in our projects have two other characteristics:

- Because of their complexity their decomposition is considered over several levels. A part of the system is considered as a subsystem and again decomposed into its parts by another architectural step.
- The components of the system come from different subsections, such as software, electronics, mechanics, etc.

As a result the development of the system (or a part of it) is embedded into a larger system. Thus, the requirements for a system at any level of decomposition come from the system architecture of the system level above.

There is one last term to look at from the definition above. System engineering is supposed to deliver a “successful” system. For us, a system is successful if the following goals have been reached:

- The project is "in time & budget", which means that the estimations for development expenses and development time were met.
- The estimated production costs have been met as well as the quality in the production of the system can be guaranteed.
- The system possesses the required properties. These may differ from the initial stakeholder requirements because, among other things, the system above may change.

The first point is addressed in the figure below. There, the development time and the development expense are shown with and without the use of systems engineering methods.

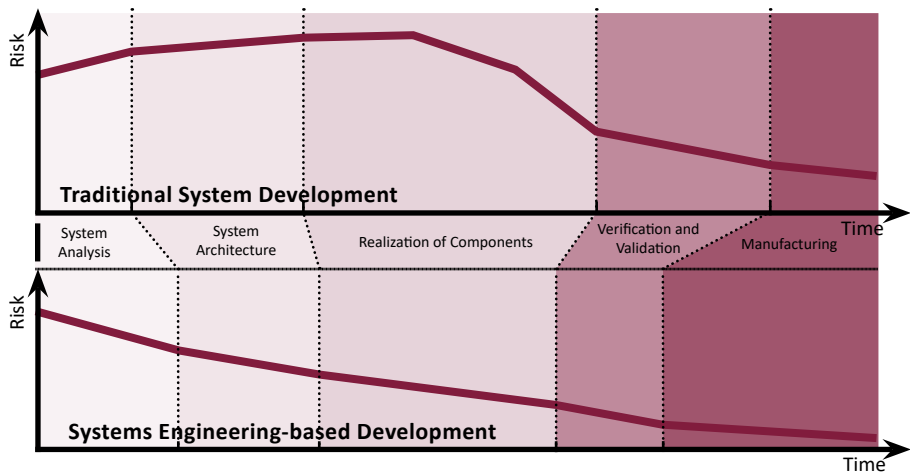


Figure 9.1.: Costs and risks before and after introducing systems engineering

By putting more effort into the system analysis, it is probable to overall save time and thus money while maintaining the same quality. The risk is minimized early and not only during the realization phase.

9.2 Embedding of Requirements Engineering

Figure 9.1 shows that in an organized system development, a high quality system analysis should be performed. As indicated in the previous chapter, a system may consist of several subsystems for which we suggest a similar procedure.

This idea is best represented in an illustration.

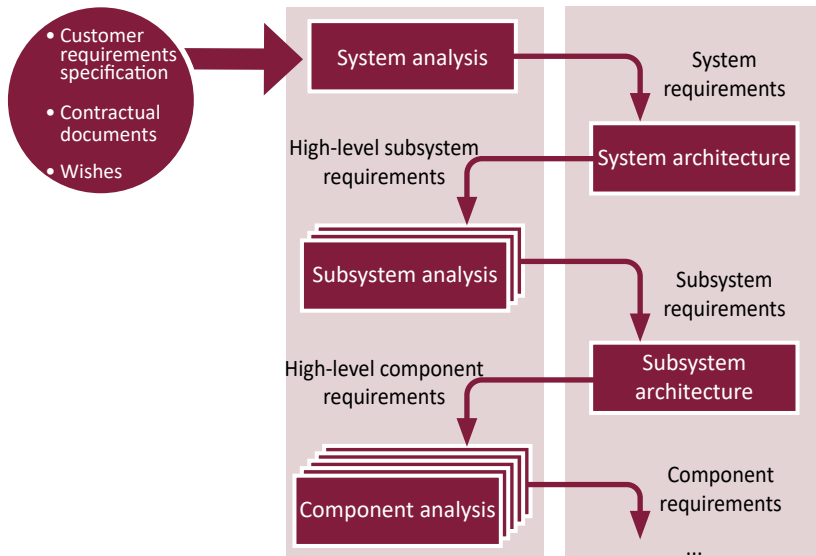


Figure 9.2: Iteration of analysis and architecture on several levels

From the input artifacts the system analysis generates the system requirements to be developed. The system requirements are the input for a system architecture step including two important sub steps for this context:

- The subsystems of the system are identified.
- Requirements for the subsystems are derived from the system requirements.

The requirements for a subsystem show how the subsystem is involved in the realization of the system requirements. So the high-level requirements can be considered as requirements of a specification sheet for the subsystem.

When realizing the subsystem, the first thing to do, is to analyze the requirements addressed to the subsystem in order to obtain resilient subsystem requirements.

In a subsystem architecture they can then be distributed to the respective components.

There are some special cases that make an exception but in general this sequence of analysis and architecture steps continues downwards until

- a component has been found that is developed outside of the responsible system organization unit (e.g., by a supplier) and/or
- a component has been found that can be assigned completely to a domain (mechanics, software ...).

This procedure ensures a complete consideration of all requirements at the different system levels.

Many of the approaches, methods and techniques we presented so far, can be used at any system level. However, there are some other techniques for the development of complex, technical systems, which for example follow from laws and standards.

There is the FMEA (Failure Mode and Effects Analysis [Werdich12]) to find weaknesses in the realization. Requirements are derived from the found risks and will be incorporated into the development at the considered system level.

A FuSa (Functional Safety, [ISO 26262]) can give new input to the analysis process on any system analysis with a HaRA (Hazard and Risk Analysis) and a functional and a technical safety concept.

Especially in safety-critical areas, there is more than an analysis process. This affects, for example, the traceability of the requirements to the inputs, the realization and the tests and checks that must be performed for the requirements.

For defining a requirements engineering procedure for systems like this, the following drivers result. If possible, all of them should be fulfilled;

- Procedural standards and norms that partly complement and contradict each other
- Quality of requirements for further development or assignments adapted to the needs of a supplier
- Time and effort, given by the project

These drivers, together with the "cautious" change that is motivated in paragraph 7, don't make it easy to define the appropriate measures for an upcoming systems engineering project. We've made the experience that the point of view from an external expert can help with the methods and gives assurance for the next steps.

10. Requirements Engineering for Smart Ecosystems and as a Driver of Digital Transformation

Prof. Dr. Key Pousttchi from the Chair of Business Informatics and Digitalization at the University of Potsdam defines digital transformation as follows:

Definition digital transformation according to Pousttchi [Pousttchi17]:

The term digital transformation describes significant changes in everyday life, the economy and society due to use of digital technologies and techniques as well as their effects.

The effects are far reaching. They affect almost all areas of our daily lives. Business models and value chains are changing drastically. Communication media and thus our way of communication are changing. Ultimately, this changes the work environment as well as our lifestyles. These new systems, but also the newly affected stakeholders, have a significant impact on the working environment of requirements engineers.

The example of the smart home system that is used in this brochure is a smart ecosystem. Many individual systems (video surveillance, heating, blinds control, access control, etc.) communicate and interact with each other. The smart home system itself provides the communication platform for its containing systems. These ecosystems can be randomly scaled. The smart home system can also be seen as a component of a higher-level ecosystem, a so-called “Smart Rural Area”. Systems like that are usually not developed by one company, but are created due to the interaction of different systems in the same context or surroundings. The differentiating factor between “normal” and smart ecosystems is that in smart systems, additional systems can be added to or removed from the ecosystem at any time.

Impact of smart ecosystems on requirements engineering

A special challenge, which we had to deal with in a lot of our projects, is setting the system and context boundaries for a system within a smart ecosystem. The difficulties are that there are more potential interaction partners and thus interfaces. At the time of development and implementation the interacting systems cannot yet be determined unambiguously and completely.

This problem is not entirely new. The challenge also exists in service-oriented systems. The keyword is: interface contracts or quality-of-service agreements or service-level agreements. They ensure that the retrieving system has a predefined (data) quality and thus the right to use the service.

Another effect is that the systems within a smart ecosystem complement each other

and are capable of providing functions as a compound that the individual systems cannot realize separately. One possible approach is not to limit oneself to the own subject of observation, but to look at the bigger picture. This can be done by analyzing the upper abstraction levels that are above the concrete subject matter. When developing a control unit within a vehicle for example, it may be helpful to also take a look at the entire vehicle in order to find dependencies and interaction of the individual parts (see also paragraph 9 - "systems engineering"). We recommend that the business process level is included in the analysis. In the process other possible deployment scenarios or application possibilities for the considered system can be identified.

Possible approaches for the analysis of smart ecosystems

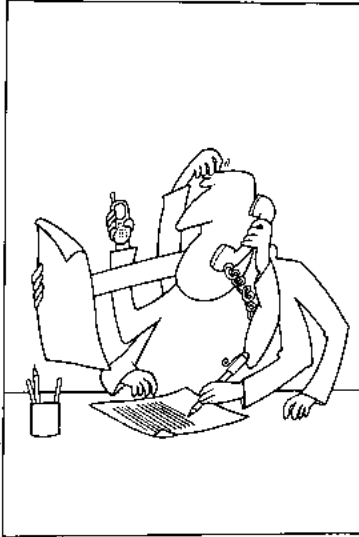
One possible approach to improved requirements engineering in a smart ecosystem is to embed the work with requirements into an agile development. The reasons are that smart ecosystems are highly complex systems with many uncertainties at the time of development. Also a large number of changes in circumstances and constraints can occur within a very short time. However, it is possible to establish, to live and to optimize an agile approach in a greater context. This is an exciting challenge for the requirements engineer that requires a lot of experience.

Another approach is model-based systems engineering. Model-based systems engineering is suitable for smart ecosystems, because it divides the amount of information and data by decomposing the whole system and looking at the different views of the considered system. By doing that the whole system becomes understandable and developable.

Solid knowledge of requirements engineering and modeling is essential, as well as experience in dealing with complex systems.

The use of artificial intelligence (AI) will influence requirements engineering, too. The specification of an AI system differs slightly from a conventional system. The significant difference is the subject of observation. In a classic system the focus lies on the functionality, the system behavior and the technical realization. In an AI system the focus is on the specification of the training process and its quality of the data. This approach moves the complexity out of the specification into the AI system. Only the goal and the criteria for productive use are specified without digging to deep into the functionality.

Digital transformation in requirements engineering



For the **elicitation** of knowledge, a challenge is that due to new technologies, knowledge is available in many different places. Thus the process of selecting sources for requirements is increasingly gaining priority. We can highly recommend two tools for the work with stakeholders who are difficult or impossible to access: personas and empathy maps. Besides, new technologies increase the working methods of a requirements engineer. For eliciting requirements we recommend the use of CrowdRE methods, living labs, design thinking or other co-creation approaches. Therefore, it's pleasant to have at least some short term help from experts with experience in these techniques.

The business world is changing; mobile work is spreading and becoming the norm. The spatial separation makes it difficult to use elicitation techniques that require the physical presence of the participants. Special tools for virtual meetings can help, as well as using elicitation techniques that are not affected by this problem. Due to digitalization, many manually executed processes are being automated. Therefore enough time and effort should be invested into the business process analysis to be able to adequately digitalize undocumented manual processes.

The digital transformation affects the main activity **creating good requirements** in a way that assistance systems can support in creating good requirements. For example, there are text recognition systems to alert quality deficiencies or image recognition software that converts diagram sketches into syntactically correct diagrams.

The increasing digital communication affects the **imparting** of requirements. In particular it will change the collaboration between the requirements engineer and his stakeholders. In the past, product/sprint backlogs, roadmaps or Kanban boards hung on walls with sticky notes. Nowadays, these artifacts are digitalized because not everyone involved is on site. Meetings that are based on working with haptic objects become more difficult to carry out. The use of virtual or augmented reality can help.

If the stakeholders are spread out and direct collaboration for improving requirements is not possible, creative, tool-based ways of working will save the day. Heavy weighted, extensive documents probably won't stay this way much longer. The change effort would be hardly manageable due to a regular and continuous delivery and rapid technological progress - unless the models are managed by means of professional systems engineering and a good tool. By now we often resort to videos for documentation and imparting. This creates a new exciting REpresentation in requirements engineering with a lot of potential.

In the **management** of requirements, change management is gaining relevance. The whole society is confronted with more technologies and uses them actively. So the requirements engineer should keep close contact to his stakeholders and expect a whole lot of change requests.

11. Business Analysis, Requirements Engineering or Both?

In many companies there is no such role or job called requirements engineer. Instead there are requirements manager, business consultant, business analyst, etc. Depending on the company the difference between these job descriptions are more or less specific. In literature, at least two terms are clearly distinguished: requirements engineering and business analysis.

We have already explained the term requirements engineering in chapter 2. Business analysis is regarding to the International Institute of Business Analysis (IIBA) defined in the following way:

Definition business analysis according to the International Institute of Business Analysis [IIBA]:

Business analysis is an activity of enabling change in an organization by defining business needs and recommending solutions that give value to stakeholders. [BABOK®v3]

This statement show an obvious similarity between business analysis and requirements engineering: "defining business needs [...] that give value to the stakeholders". That sounds a lot like requirements, doesn't it? I.e. in short: requirements engineering is embedded in business analysis. But what exactly does it mean?

Business analysis is a very broad discipline that may include many individual activities. In principle, business analysts need to understand a company's strategies, goals, business processes, etc. and combine them with market requirements to develop products/improvements that meet the company's business goals. It can be divided into the following four different areas.



Planning and controlling work steps

One of the most frequent activities we perform in business analysis is planning and controlling all business analysis activities. In our experience often a simple PDCA cycle is implemented. Planning (P) includes familiar things such as stakeholder analysis, setting up requirements management and planning the individual work steps. After that the work steps are to be done (D) and the result is documented. Now it can be checked (C) whether the steps have brought the desired result and whether it was effective. Finally, if necessary, one will act (A) and derive adjustments to the work steps in order to compensate for any possible deficits.

SOPHIST Self-productions

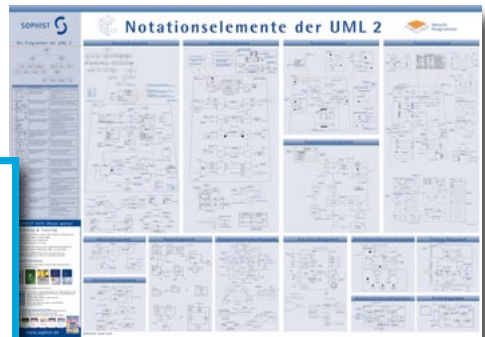
A different kind of knowledge carriers!

Freebie!

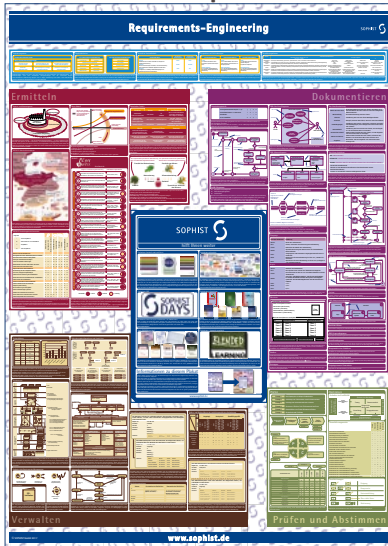


Posters

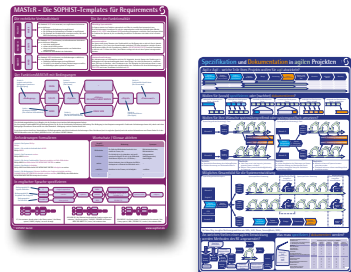
The SOPHIST UML-poster



The SOPHIST RE-poster...



... and its „cores“



Creating a business case

In order to achieve improvement or develop a new product, the first step is to find out what is required. Therefore, we identify potentials with our customers, derive business requirements, collect alternative solutions and evaluate them. It can be done the traditional way by means of feasibility studies, economic feasibility studies, etc. However, our project experience shows that the use of agile techniques is rising in this area: product canvas with minimum viable product (MVP), vision statements, product box, etc., which are presented in the course of an elevator pitch.

Requirements engineering

Is it done? Has the management been convinced of the product idea/business case? Then requirements engineering begins. Never mind if it's a classical or agile process model. Detailed information about the selected solution alternative needs to be determined, requirements derived, documented, prioritized and imparted to the development team.

The best way to do this can be found in our brochures, books, on our website or in personal conversations.

Establishing developed solution

Finally, the most important part: to accompany the establishment of the solutions into the targeted environment. Especially if new software has been developed to optimize the business processes, or even if the solution replaces staff, proceeding with special caution is recommended. It would be easy to just meet the technical requirements for integrating software into the production environment but that's not enough. The greatest threat lies in the risk that the stakeholders will reject the solution due to existential fear or the fear of change. The key lies in a well-planned, organizational change process (see also paragraph 7 - "Establishment of an improved requirements engineering") that engages all people affected/involved.

So, business analysis and requirements engineering are closely intertwined. If help is needed, we are ready and happy to consult!

12. Videos within Requirements Engineering

In the context of requirements engineering activities many requirements will be created that should be documented or imparted in different ways. These collections of requirements can get very complex. As a result the connection between the requirements can be lost notwithstanding model-based documentation techniques or story maps are used.



Videos can help. Among other things they can show many small requirements and their connection in a comprehensible way. Videos can also help in the elicitation phase; the original problem or the context of the new system can be shown.

Thanks to modern technology it's no problem to create and send such videos. There are apps for smart devices to create and edit videos, which are cheap or free and easy to use. Of course, the planning, the creation and the editing of the videos are not completely free. However, the time and therefore costs spent are compensated by fewer costs for imparting the requirements. Especially, if there are a lot of stakeholders and consumers. To keep the cost of the videos as low as possible, we have briefly summarized the most important thoughts and principles for their creation and use.

The PILZ of the video

Before starting to film a video, it needs to be decided carefully which content and purpose the video shall have. We have defined a structured way that we call PILZ (German for mushroom). In PILZ, four different dimensions are addressed, which determine significantly the content and the form of the video (the script).

- Phase ("Phase"): In which phase of the requirements engineering process should the video be used. For example, should it support the elicitation or the imparting of the requirements?
- Content ("Inhalt"): Shall the video impart a rather static or dynamic content? E.g., the presentation of the entrance area of the house in which a part of the new smart home system shall be embedded (static) or the currently typical sequence when entering the house without the new system (dynamic).
- Solution reference ("Lösungsbezug"): How much of a possible (subject-specific or technical) solution shall be given? It is possible to show the entering of a PIN code at a keypad (technical solution) or this action can be hidden behind an inserted card with the label "Authentication" (solution-neutral).

- Time reference (“Zeitbezug”): Does the video represent a target state or the current state? In other words: Does the video show a goal in an abstract way, or does it show the currently given state?

For the different characteristics in the four dimensions, we recommend actions to support the creation of the script and thus ensure the quality of the video. For example, to show a static content, it should be switched between an overview and detail shots, showing the details for a certain amount of time without changing the camera position. In [Rupp20] we have given the complete list of characteristics of the dimensions including the respective recommended actions.

Embedding into a collection of requirements

Another part of our approach is to describe how videos can be linked to a collection of requirements. Videos in requirements engineering usually have a user’s perspective, so they can be homogeneously integrated into document structures that use use cases as the top hierarchical level.

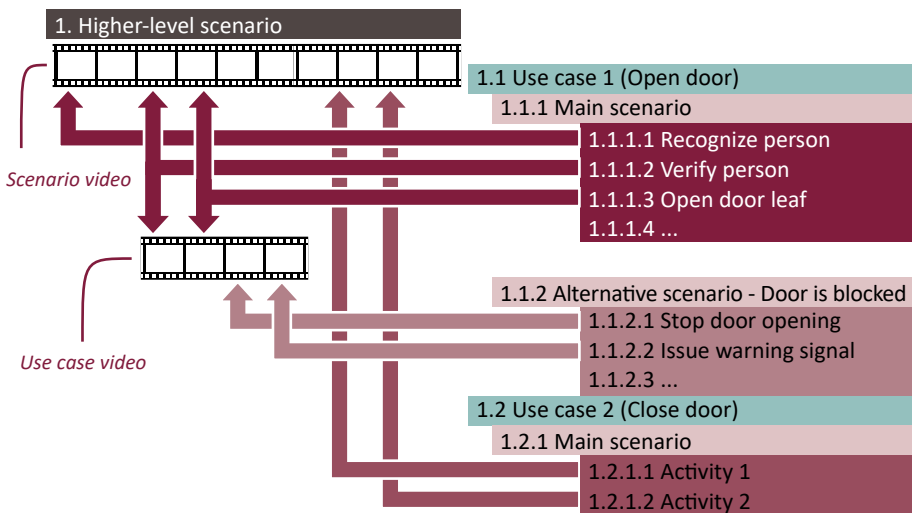


Figure 12.1: Levels of detail for videos

With this approach, it can be decided which content shall be presented at which abstraction level in different videos. Figure 12.1 shows different videos at the use case level, that visualize different parts of the possible sequences. These use case videos can now also be cut together to visualize a higher-level sequence, so that requirements collection can be extended easily at different abstraction levels.

Shooting videos in a workshop

The creative act of creating videos requires an intensive analysis with the content that shall be presented. In our experience, the creation of a video can be done very well in a workshop. During the planning and actual shooting of the video, negative aspects in a described process can be found. Details must be defined clearly, and existing requirements must be double checked to see if they e.g., don't already contain (unintended) parts of a solution. When creating the script, the way of working with the requirements is special. It helps, if several people are collaborating. The actual shooting is always a group event that will be remembered for a long time by all leading and supporting actors.

In addition to PILZ, we have some other tools ready to conduct video workshops, e.g., storyboards and mood boards. More details can be found on our website at www.sophist.de/re7/kapitel27.

13. Requirements Engineering for Product Lines and Families

We as customers are lucky! We can always and everywhere choose from many alternatives - or even better - create our own individual product. To make this possible, we are supported, for example by configurators. In many areas, the ideal is: the more individual a product, the better.

However, a growing variety and individuality lead to complications in the product development. In order to reduce the development effort as much as possible; the analysis of reuse options and product similarities is important. The goal is to fulfill as many different customer requirements as possible with the least possible development effort. Nowadays most products aren't new, but are developed further in an evolutionary manner. Almost nobody launches a single product, but usually a product family or product line, in which the products differ more or less. In the spirit of "shift left": The earlier this fact is taken into account in the development process, the higher the savings potential in the entire development process – best, if already considered in portfolio management when defining the product strategy.

The feature model

We assume that there are several products that are similar in terms of their properties and functions, but also differ in some way. The properties or functions are called characteristics or features. Later they can be used to select or distinguish between variants. For documentation purposes, the feature model has proven to be a useful notation.

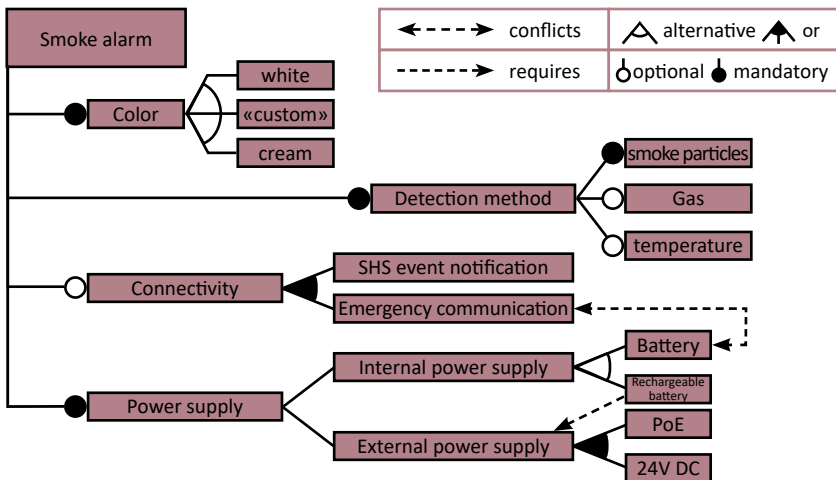


Figure 13.1: Feature model of smoke alarm

If the product is supposed to be described "as always", to each requirement the matching feature from the feature model can be assigned. The features can be linked to requirements or simply be seen as an attribute of the requirements used for the mapping.

Smoke alarm Use Cases; requirements	Source/Derivation from Feature
Detect danger	Always
- Detect smoke particles	Smoke particles
- Detect gas	Gas
- Detect temperature	Temperature
Signalize event	Always
- Signal danger	Always
- Signal event	SHS event notification
Use voice connection	Emergency Communication
Charge battery	Chargeable battery
Establish connection to control center	SHS incident signaling
Turn on system	Always
Turn off system	Always

Figure 13.2: Mapping requirement features

Based on the feature model and its rules, it is easy to determine different variants and, of course, the minimum viable product (MVP) or a platform (which is common to all products).

Obviously the goal is not only to reuse the requirements, but also other development artifacts such as architecture, test cases or even test results, if possible.

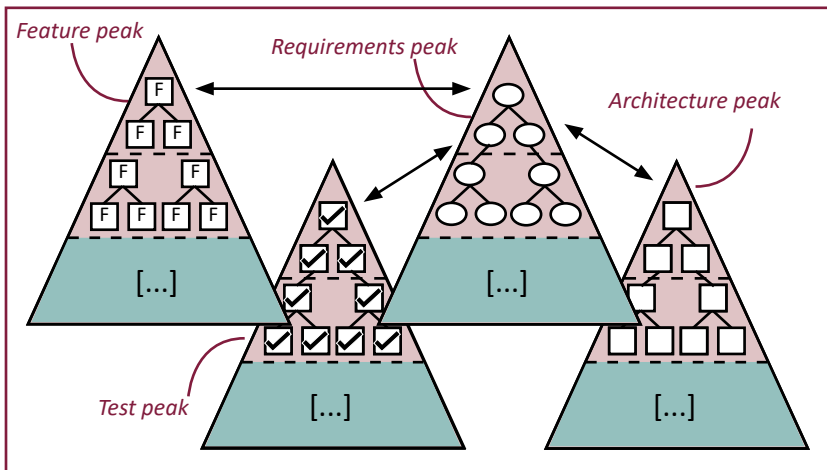


Figure 13.3: Peak Model

14. Conclusion

Requirements and user stories are the heart of successful system development. They can decide over success and failure, as they are the precondition for contracts, correct implementation and development, as well as tests and acceptance criteria. This applies to agile development just as much as to classical development.

Requirements engineering is a demanding field of activities that opens up many possibilities. For instance, professional requirements engineering increases the chances for the success of projects in IT environments and others. An appropriate analysis can ensure that

- the system to be developed is described appropriately to the procedure.
- the fulfillment of goals is ensured by the associated requirements or stories.
- only agreed and tested requirements that meet the goal, are implemented.

Even if the expenses for a requirements engineering that is adapted to the approach seems high - a lot of money is saved if the development goes into the right direction and match the customer's wishes right from the beginning. Requirements engineering is worth its costs!

This brochure is supposed to give an impression of what belongs to the basic work of a requirements engineer. Of course, we could only present a small part of a complex job. There are more information in our publications, on our website and of course in consulting conversations and trainings.

So if you want to learn from the pioneers, professionals and real practitioners, SOPHIST is the right place for you. Equipped with years of experience in a variety of contexts, we are happy to take care of you, your projects and your co-workers. We are always in line with your needs, because our customers are the centre of all of our activities. We have no standard approach. We adapt processes, techniques and methods individually to your needs. Together we can work out the approach that works best for you. As the ancient Greek Sophists said – To us the human being is the measure of all things.

We are looking forward to getting in touch with you.

Yours, the SOPHISTS

Bibliography

- [Babok®v3] International Institute of Business analysis: BABOK®v3: Leitfaden zum Business Analysis Body of Knowledge, 3rd Edition, Wettenberg 2017
- [Bandler75] Bandler, R.; Grinder, J.: The Structure of Magic II. Science and Behaviour Books, Palo Alto/CA, 1975.
- [Bandler94] Bandler, R., Grinder, J.: Metasprache und Psychotherapie, Die Struktur der Magie I. 8th Edition. Junfermann, Paderborn, 1994.
- [CPRE20] Pohl, K.; Rupp, C.: Basiswissen Requirements Engineering. Aus- und Weiterbildung zum Certified Professional for Requirements Engineering – Foundation Level nach IREB-Standard, 4th Edition, Heidelberg, 2020.
- [Goodwin09]: Goodwin, K.: Designing For The Digital Age. 1st Edition, Indianapolis, 2009.
- [INCOSE20] INCOSE: Guide to the Systems Engineering Body of Knowledge. Introduction to Systems Engineering (2019), https://www.sebokwiki.org/wiki/Introduction_to_Systems_Engineering (as of 01/09/2020)
- [ISO26262] ISO-Norm 26262: Road vehicles. Functional safety, 2nd Edition, 2018.
- [Pousttchi17] Pousttchi, K.; GITO: Digitale Transformation (2017), <http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/technologien-methoden/Informatik--Grundlagen/digitalisierung/digitale-transformation> (as of 04/01/2020)
- [Rupp20] Rupp, Chris: Requirements-Engineering und –Management v, Aus der Praxis von klassisch bis agil. 7th Edition. Hanser, München, 2020.
- [SAFe] Scaled Agile Framework: Scaled Agile Framework (o. J.), <https://www.scaledagileframework.com/> (as of 04/01/2020)
- [Wake03] Wake, B.: INVEST in Good Stories, and SMART Tasks (2003), <https://xp123.com/articles/invest-in-good-stories-and-smart-tasks> (as of 07/25/2019)
- [Werdich12] Martin Werdich: FMEA – Einführung und Moderation. 2nd Edition. Vieweg & Teubner 2012,
- [Wolfe10] Wolfe, P.: Brain Matters Translating Research into Classroom Practice. 2nd Edition. Alexandria, 2010.



The basics of Requirements Engineering

When developing a system - be it software or a building - the most important prerequisite for success is that all those who are involved know exactly what is to be developed. And this already begins with the rough targets and ends with highly detailed instructions for the implementation.

The work of requirements engineers deals precisely with this topic. Their job involves:

- elicitation,
- documentation,
- validation and consolidation, as well as
- management

of system requirements.

With this brochure, we want to give you an overview of the activities belonging to these roles and, hence, what we, the SOPHIST's, have specialized in for now over 20 years.