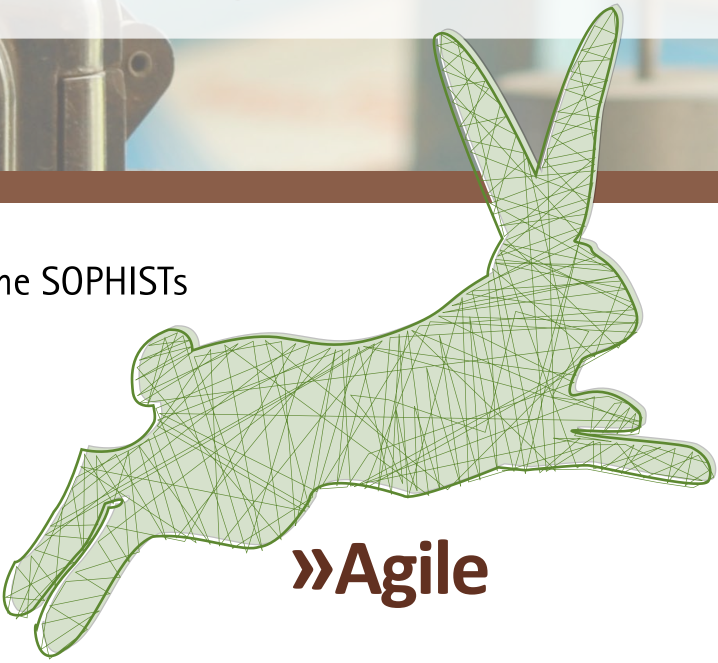
AGILITY



The SOPHISTs

»Agile Requirements Engineering«

SOPHIST

The SOPHISTs

SOPHIST GmbH
Vordere Cramergasse 13
90478 Nürnberg
Deutschland
www.sophist.de

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

1St edition 2022

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

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 evironment - more movement, less text, more interaction with the paricipants 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?

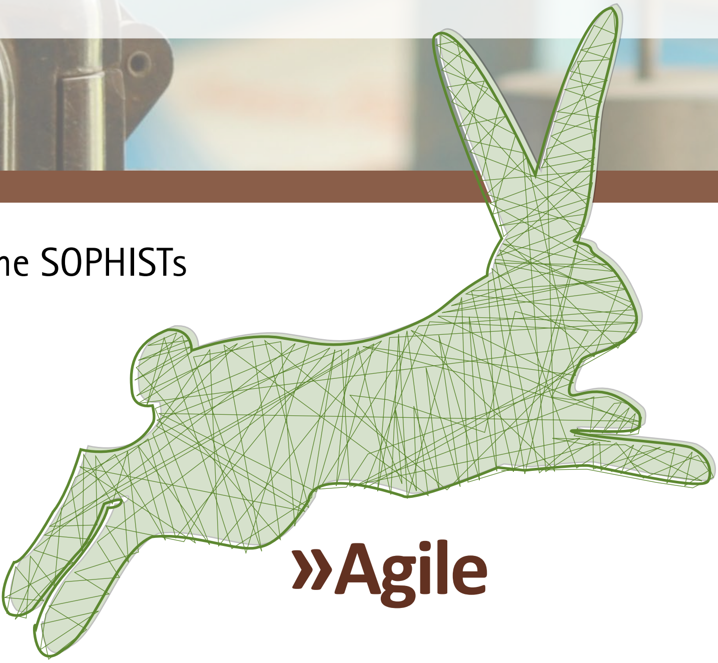
Contact us without obligation:

+49 (0) 911 40 900 - 0

heureka@sophist.de

AGILITY

The SOPHISTs



»Agile Requirements Engineering«

SOPHIST

www.sophist.de

1. Introduction

1.1 What is Requirements Engineering?

Do you have a wish that you cannot fulfill? Do you have a product vision or a goal but you do not know how you can achieve it? Let's say hypothetical, that you want a blue car with as many horsepower as it is possible and say to your development team that they shall build that car. However, when the final product is presented to you, disillusionment follows. Your development team delivers a blue car with lots of horsepower. However it is a truck and not a sports car as you had imagined. There are difficulties in communicating, analyzing, and conveying requirements and because of that, requirements engineering comes into play. In the past, requirements engineering was thought to be simple or superfluous: the requirements engineer conducts interviews with stakeholders, writes down the resulting requirements, and composes them into a requirements specification.

However, this is only partially true. The requirements engineer is responsible for more than just that. He/she serves as a connecting point between the stakeholders and the developers. The following tasks are among his/her four main activities:

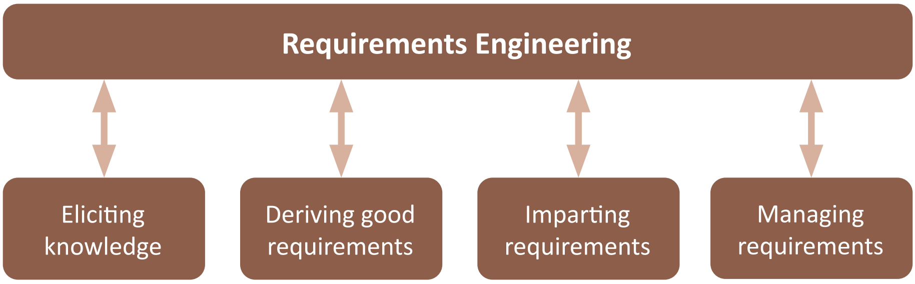


Figure 1.1: The four main activities of Requirements Engineering

Eliciting knowledge from the stakeholders is his/her most important activity. However, it is not as simple as it sounds, as knowledge is divided into three categories: the conscious, the subconscious and the unconscious knowledge. In the example above, you were only aware that the car should have a lot of horsepower and be blue. These characteristics are also called performance factors. However, subconsciously you wanted to develop a sports car, which is a basic factor to you. You may also subconsciously be excited about automatic drive function in your new car. This is called the excitement factor. In order for the requirements engineer to determine these factors, he/she makes use of a variety of elicitation techniques to do so, such as conducting interviews, reusing past products, or brainstorming.

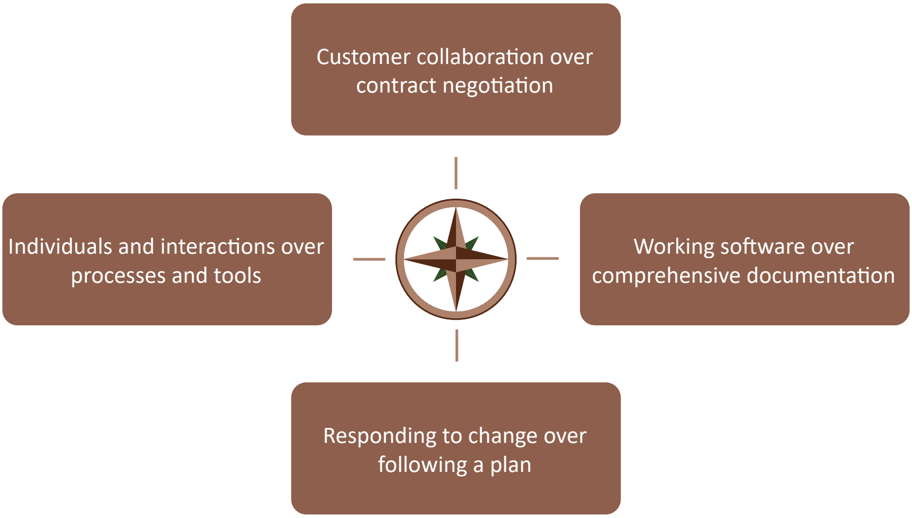
In order for the requirements engineer to deliver his/her elicited knowledge to the development teams, he/she must **derive good requirements** which can be understood and implemented by development teams. It is essential that the requirements are unambiguous, complete, necessary, agreed among stakeholders and understandable; otherwise the final product may not meet your wishes. There are certain quality criteria for this, such as the INVEST principle. At SOPHIST, we use e.g. our SOPHIST set of REgulations or our MASTeR templates to create requirements with a good quality.

The preparation of the requirements is important because the requirements engineer has to **impart the requirements** to further persons. Therefore, he/she uses various imparting techniques according to the situation. Examples for these techniques can be storytelling or videos. Usually, it is not only the content of requirements that play a role in imparting them, but also, for example, the complexity or the scope of the subject matter. This main activity also includes the documentation of requirements, classically e.g. via requirements specifications, or in agile frameworks using the product backlog.

Finally, the requirements engineer must **manage the requirements**. This activity especially serves the traceability and modifiability of requirements. It may be that requirements need to be adapted or changed because you want a red car after all, or because you have discovered that you do not want to install the engine with the most horsepower after all. Versioning and traces can be used to keep a record of the changes and dependencies of the requirements. Above all, this main activity serves to improve communication within and between development teams in a project, to increase the quality of requirements and thus the quality of the product and processes, and to simplify the monitoring of complex projects during all development steps. And this reduces the stress for all involved parties.

1.2 What is Agility?

What exactly distinguishes agile methods from heavyweight methods, such as the waterfall model? The four basic ideas of agility are defined in the „Manifesto for agile Software Development“, also called the agile manifesto:


Customer collaboration over contract negotiation
Individuals and interactions over processes and tools
Working software over comprehensive documentation
Responding to change over following a plan

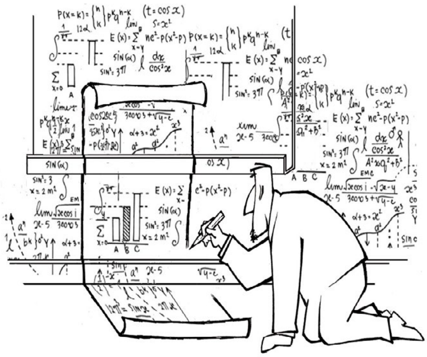
Here, the cooperation with the customer is more important than contract negotiations. Of course, contract negotiations are also important, because the cooperation should also be defined (required resources, expectations, goals, etc.). But a positive and constant collaboration brings more advantages than a detailed contract.

Furthermore, working software is more important than comprehensive documentation. Again, this does not mean that documentation should be disregarded, but instead that, time should be invested in working software rather than in extensive documentation. It is of no use to you as a customer if your software does not work, but it has been documented in detail.

Individuals and interactions are more important than processes and tools. Again, processes and tools should not be completely ignored, but what we want is to ensure communication between and within the different teams. Even if your development team is equipped with the best tools, a good end product will only be achieved with a good communication between all your team members and the responsible people, such as the product owner.

Reacting to change is more important than following a plan. Having a plan is not bad. However, the goal of agility is to react spontaneously to, for example, market changes that interest you and to shape the product according to your wishes. And doing so is the only way to keep your product up to date and true to what you want.

The reasons why products are developed in an agile manner are numerous. Let‘s briefly discuss the most important ones:



- The iterative approach in sprints allows for early and regular feedback from stakeholders. This prevents the development from going in the wrong direction.
- The iterative-incremental approach makes complex systems more manageable.
- The cooperation in team increases the employee’s motivation and ensures that everyone can identify with the product.
- The intentional minimization of rules and specifications ensures an efficient way of working.

Among the best-known methods in the agile context are:

- Scrum
- Kanban
- Crystal family

Since in agile working environments are many development teams involved, it is necessary to work within a framework that facilitates the communication and synchronization of dependencies between the teams. The most important and well-known frameworks in the agile world are:

- SAFe
- LeSS
- Nexus
- Scrum@Scale

1.3 RE in Agility

Since agile is a broad area with various approaches and frameworks, we will look at requirements engineering in scrum as an example.

In order to be able to understand the requirements engineering in this framework, we will first describe roughly the setup of scrum.

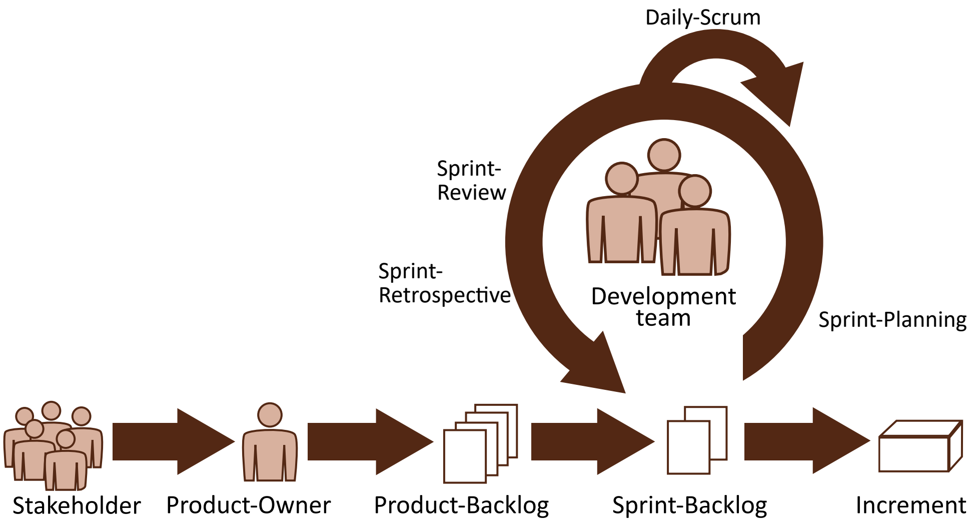


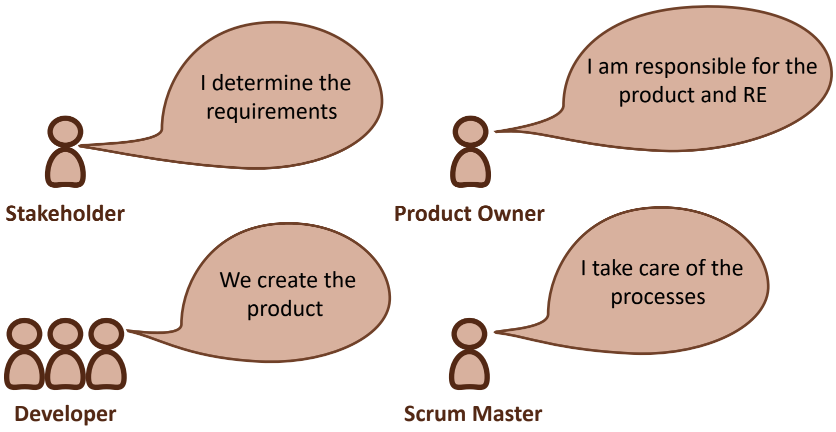
Figure 1.2: Procedure of development according to Scrum

In scrum, the stakeholders share a product idea or vision to the product owner. The product owner records the associated requirements in the product backlog and discusses them with the development team. At the start of a sprint (a period between 2-4 weeks), the developers take as much content from the product backlog as they can implement in that sprint. In other words, they define the sprint backlog. In the sprint, the selected content is implemented according to the agreements between the product owner and the developers. Here, the scrum master supports the development team with concerns regarding agile working methods and processes. The goal of the development team is to create a product increment in each sprint. A product increment is a part of the future end product that could potentially be delivered. This means it is something that works and has the required quality. Thus, the product will grow from sprint to sprint.

Roles and Responsibilities

- In order for a project to be successful, the various roles of the people involved must be clarified in advance. A scrum team includes the roles of the product owner, the development team and the scrum master.
- The product owner is the requirements engineer in the agile world, but with more responsibilities than a requirements engineer in classic framework. He/she is the interface between the stakeholders and the development team. With the stakeholders, he/she determines the vision or the product goal, agrees this with them and records it using product backlog items. He/she also manages the product backlog by clearly and comprehensibly defining, prioritizing and sorting product backlog items. In doing so, he/she should also consult with the development team and is entitled to hand over parts of his/her tasks to the development team. However, he/she is responsible for maximizing the value of the product, measured against the product vision or product goals. For this reason, he/she determines the order in which backlog items are implemented by sorting the backlog items.
- The development team consists of 3 to 9 members who are responsible for the realization of the product increment. In the end, it consists of several experts in various disciplines that are relevant to the implementation, who work on an equal level to implement the sprint backlog items and incorporate them into the final product. The development team is self-organized and carries the responsibility for the implemented product. Frequently, requirements engineers can also be assigned to development teams to support the product owner in his/her activities.

- The scrum master is the one who is responsible for the implementation and understanding of scrum. He/she is the person in charge of implementing scrum in the organization. This means that he/she is also responsible for ensuring that the members of the development team and the product owner adhere to the rules they have agreed upon and that all stakeholders gain an understanding about agile working methods. He/she also gives advice to the development team and the product owner on the methods and techniques that are used in the everyday work.
- Despite all this, the most important role is the role of the stakeholder. This is often taken on by individuals, companies or organizations. The stakeholders determine a vision or a product goal and are the contact person for the product owner in this respect.


I determine the requirements
Stakeholder
I am responsible for the product and RE
Product Owner
We create the product
Developer
I take care of the processes
Scrum Master

Events

There are a few events established within the scrum framework. These events are described below:

- The **sprint** is a period of only a few weeks and at the end of each sprint a part of the product (a product increment) should be implemented. In this way, the product continues to grow from sprint to sprint.
- The **sprint planning** takes place at the beginning of each sprint and is used by the scrum team to determine what should be implemented in the sprint. In addition, the developers will also plan how the implementation will take place in the sprint.
- The **review** takes place at the end of each sprint and is primarily used to review the sprint results together with the stakeholders and at the same time to gather feedback from them.
- The **daily** takes place every day during the sprint and it is used for the developers to briefly agree on what work is to be done that day.
- The **retrospective** rounds up a sprint and lets the team members review the sprint in order to find approaches that can be used to improve their work.

In this brochure we would like to show you how requirements engineering looks like in scrum, what happens in the events and what else needs to be done outside the events regarding requirements engineering.

Artifacts

Scrum defines three artifacts

- The **product backlog** is the place where we will store our requirements. The backlog is an ordered list of all the things that are to be incorporated into the product (see chapter 5.1)
- The **sprint backlog** contains all the backlog items that are to be implemented in the current sprint. It is filled by the developers with item from the product backlog at the beginning of the sprint during the sprint planning.
- The **(product) increment** is the result of a sprint. The increment is a (small) part of the product which is developed in one sprint, so that the final product will be developed during multiple sprints.

It may also be useful to use other artifacts. These three are at least the minimum of artifacts that are required.

In chapter 7 we have explored the idea of further artifacts for requirements engineering in scrum.

2. Goals/Vision, Stakeholder, Delimitation of the System

In order to elicit requirements for the system to be developed, you, as a requirements engineer, should build a foundation. This foundation consists of defining the goals/ vision, of finding the right stakeholders and determining the context and boundaries of the system to be developed.

2.1 Goals/Vision

Goals are initial rough requirements that specify the direction in which development needs to be going. The term vision is often used in connection with goals. Vision is a rough and long-term definition of goals. It is important that the goals/vision are documented and that all participants have a common understanding of these.

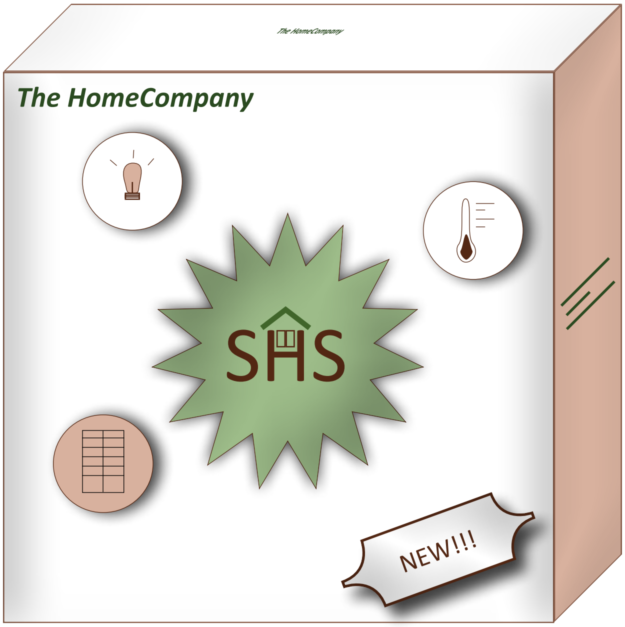


Figure 2.1: Example of a Vision Box

Because without clearly defined goals/vision, you - as a requirements engineer may find yourself in the situation where you have no guidance on which goals/vision must be met by the requirements and may be specifying past the actual goals/vision.

And without a shared understanding of the goals/vision, it is possible that individual stakeholders may end up dissatisfied because the system developed does not meet their expectations.

You can avoid dissatisfaction by working together with all stakeholders in order to identify and document the goals/vision.

To elicit these, you can hold a workshop where you can create, together with the stakeholders, a vision box [insert reference], formulate the goals/vision using methods like: PAM (**P**urpose, **A**dvantage, **M**easure), News from the future or using a canvas.

To describe high-quality goals/vision, the acronym SMART is also often used. The individual letters stand for Specific, Measurable, Attractive, Achievable, Time-bound and can help you formulate good goals.

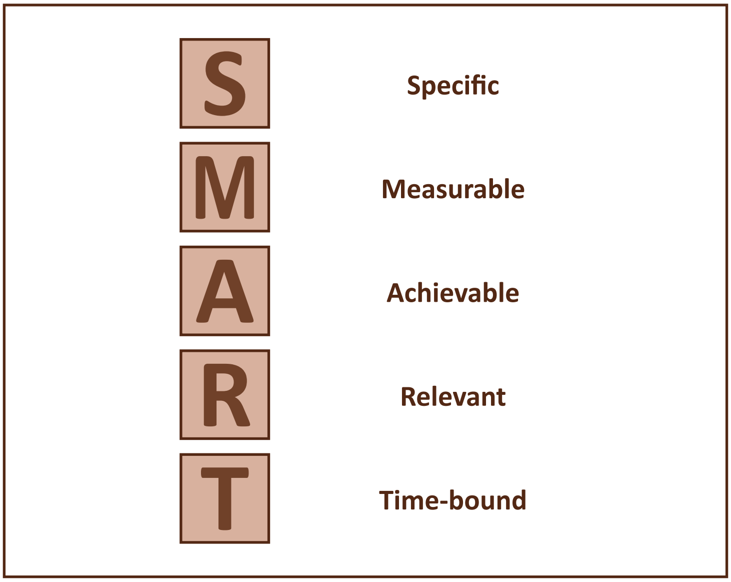


Figure 2.2: SMART

2.2 Stakeholder

In addition to defining the goals, it is also important that you find all relevant stakeholders. As we have already mentioned (in Chapter 2 Roles and Responsibilities), stakeholders are of great importance.

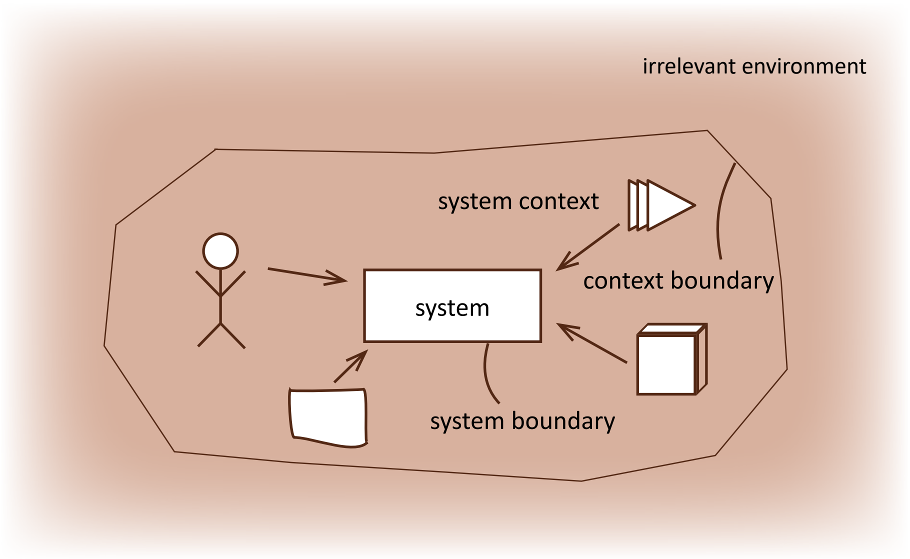
To find all relevant stakeholders, we use stakeholder checklists. Once you have identified a stakeholder, you should systematically document the relevant information about that stakeholder. A simple way is to keep a form of table. Keep in mind that stakeholders can and should be added on an ongoing basis and that the stakeholder list must be maintained and updated accordingly.

Contrary to classical approaches, you - as a requirements engineer have to involve the stakeholders in a different way in agile approaches, since they are to participate continuously in the development of the product. This leads to additional tasks for the requirements engineer. For example, the requirements engineer must present the product increments to the users in order to obtain feedback, or he/she must encourage the stakeholders to participate on an ongoing basis.

2.3 System Context

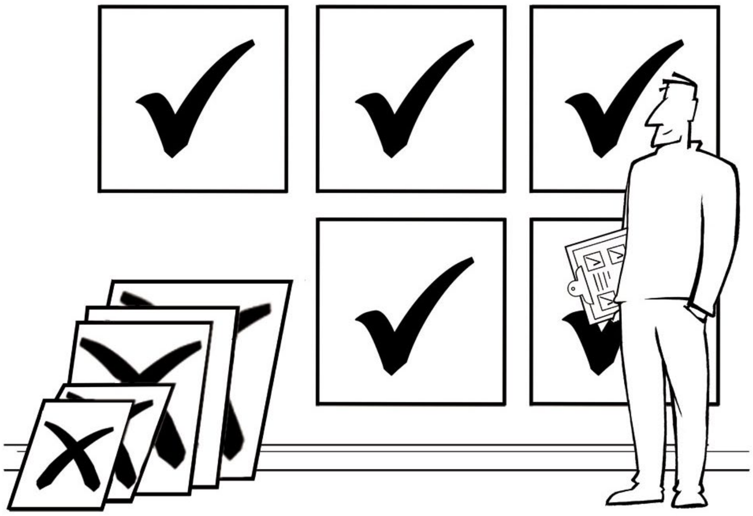
The basis for the elicitation of requirements should almost be complete now. After you have identified and documented the goals/vision and the relevant stakeholders, all that is missing is the context delimitation.

You should determine here the rough scope of the system, the context of the system and the associated system boundaries to separate it from its environment.



The system boundary is determined by the fact that no requirements will be captured for the parts that are outside the system boundary. But for these parts, you will still need to delimit the system context from the area that plays no role in the requirements. The boundary between these two areas is called the context boundary.

To determine the system context, you must identify all objects that have a connection to your planned system and therefore influence the requirements for the system under development.



As with the stakeholder list, you must take into consideration that the system boundary is not yet definitive. So you and the developers will start to develop the system, although the final system boundary and context boundary has not yet been established.

Especially in agile frameworks, we have the phenomenon that the system boundary and thus the context boundary keep changing in the course of the development. It is therefore of great importance to always keep this in mind.

Online Trainings

Your SOPHIST training - almost anywhere in the world

Online 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 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

3. From Requirements to the Product

The Life of a Backlog Item

Just as it is done within non-agile frameworks, we also need to identify, analyze and communicate requirements, although we find ourselves within an agile framework. Of course, there is also administrative work, but this is not the focus at this point.

Let‘s take a look first at requirements elicitation, the activity that can, from a logical point of view, be considered the starting point. The big difference in the agile framework in comparison with classic frameworks is the point in time when we elicit requirements. Especially detailed requirements will be considered and analyzed just before they are to be implemented and everything that will be implemented in the distant future may only be recorded as a rough requirement. This means that when we work out detailed requirements with the stakeholders, other requirements have already been implemented (except in the first iterations) and thus the system parts that already exist can, or should, be of great help in determining the new requirements.

The elicited requirements are gathered in the product backlog as backlog items (e.g., user stories). It can be helpful to group the backlog items according to topics. This is usually done by assigning detailed backlog items to roughly detailed backlog items. An example of this is the assignment of user stories to epics.

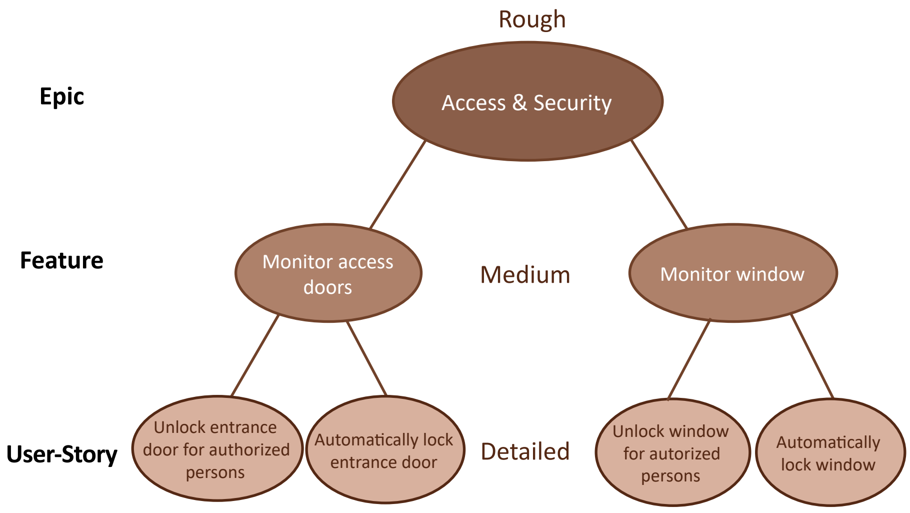

Rough
Epic
Access & Security
Feature
Monitor access doors
Medium
Monitor window
User-Story
Unlock entrance door for authorized persons
Automatically lock entrance door
Detailed
Unlock window for autorized persons
Automatically lock window

Figure 3.1: Example of User Stories for an Epic

The backlog items collected in the product backlog must be prioritized on a regular basis. This is our responsibility as product owners. However, this should not prevent us from seeking advice and other opinions. If the priority should reflect the importance of a backlog item for the user, then we should refrain from deciding ourselves what is important for the user. Unless, of course, we ourselves are the future users of the product. You can find more on prioritizing backlog items in chapter 6.4. IIn our product backlog we have to sort the backlog items according to their priority. This means that the items with the highest priority will be at the top of the backlog.

Now, after the prioritization took place, we know which requirements (backlog items) have the highest priority and we will implement these items according to their priority. But before the implementation can start, we still have to do a few things. We have to get the backlog items into a shape in which the entire scrum team agrees that the backlog item is ready for implementation. In this situation we say that the backlog item is "ready".

The definition of ready (DoR) helps us to decide when an item has reached this state. The DoR contains all criteria for a ready backlog item and is agreed upon by the entire scrum team. Common and also useful contents of the definition of ready are::

- The backlog item fulfills the INVEST principle (see chapter 6.1)
- The backlog item is understood by the parties involved (see chapter 6.2)
- The backlog item contains acceptance criteria (see chapter 5.2)
- ...

To meet these criteria, we need to decompose the backlog items (see chapter 6.1) and talk to stakeholders about them in order to gain a more detailed knowledge about it. We need to do everything we can to gain a shared understanding regarding the contents of a backlog item and determine when, in our eyes, the backlog item is ready to be implemented. So, we analyze the backlog items in detail and make them more concrete. However, we should not take it too far, because we do not want to document all requirements down to the smallest detail in the backlog items, for a reasonable amount of freedom should be left for the developers.

We must do this always for the backlog items that are to be implemented in the near future. Therefore, this will be an ongoing activity. Because whenever backlog items are implemented, new backlog items move to the top of our product backlog.

An important point within agile development is to create a shared understanding. Therefore we will talk on a regular basis with the developers (and also together with stakeholders) about the backlog items. Refinement meetings can be scheduled to talk about the backlog items that will need to be implemented in the near future. The time between the meeting and the actual scheduling of a sprint should not be too long; otherwise everything that was discussed will have been forgotten by then. You can read about what needs to be taken into consideration in such a meeting in chapter 6.2.

When the backlog items have reached the status ready, they can be moved into the sprint backlog by the developers during one of the next sprint planning sessions. By doing so, they will be removed from the product backlog and thus room will be made for the next backlog items, which shall go through the same steps. All the backlog items that the developers think they can implement in the sprint end up in the sprint backlog. But the developers don‘t just choose random items from the backlog; they follow the order of the backlog items in the product backlog. Therefore, it is very important that to have properly ordered the backlog.

At the end of the sprint, the results, i.e. the product increment created in the sprint, are presented to the stakeholders in a review meeting. Here, the stakeholders can see how the product was developed further within the sprint and will have the opportunity to give feedback and name new requirements. This is usually the point at which the lifecycle of a backlog item ends. If the backlog item is needed again later, it can be archived. Otherwise it shall be deleted.

The last act in a sprint is the retrospective. This allows us to reflect on the previous working methods and to develop ideas regarding potential improvements. The retrospective focuses on the cooperation within the team and not on the product. The retrospective is very important because agile work strives for continuous improvement.

4. Eliciting Requirements

Requirements elicitation is an ongoing activity. It is carried out until the end of product development. Particularly at the beginning of the development, many requirements are initially only coarsely elicited. As soon as the implementation of the requirements approaches, the detailed requirements are to be elicited.

The specific characteristic of the agile approach is that we can and should use the product increments that have already been developed to help us elicit further requirements. This allows the stakeholders to experience what has been achieved with the previous requirements and to come up with new requirements in a playful way. Some quality requirements in particular (performance, usability, etc.) can be made explicit more easily in this way.

By allowing stakeholders to provide feedback on existing product increments, to express new requirements and to be able to change requirements throughout the product development, they will be given a sense of being an active part in the developing the product.

In the agile world, we are also seeing an increasing number of approaches that involve the stakeholders more closely in the elicitation of requirements. Working out a story map [Rupp 20] together with the stakeholders can bring undiscovered requirements to light. Or you can go through a user journey together with the users. There are also approaches that combine different techniques. Well known are "Design Thinking" and "Living Labs" [Rupp 20]. If certain stakeholder groups are not known or the group is too large and it is impossible to be able to talk to individuals, then personas and crowd-RE offer promising approaches that can help you in eliciting the requirements of these stakeholders. [Rupp 20]

5. Documentation of the Requirements

5.1 The Product Backlog

Whereas in the non-agile frameworks one typically works with requirements documentation (e.g., a requirements specification), in the agile framework, most of the requirements are found in the product backlog. Thus, the product backlog is the central place where requirements are documented. However, not everything that is to be found and documented in the product backlog automatically has to be a requirement. In the product backlog, you will find everything that, according to the current status, will be included in the finished product at some point. It is important to emphasize the „current status“, because just because something is in the backlog today does not mean that it will still be there tomorrow.

In principle, the following properties are attributed to the product backlog.

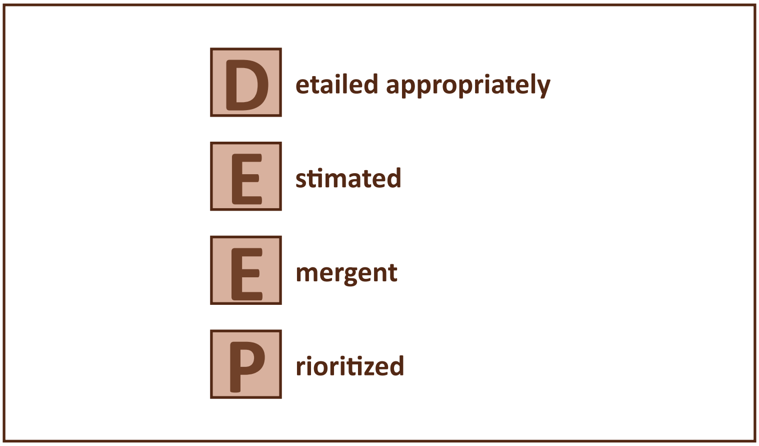


Figure 5.1: DEEP

The product backlog Items are **detailed** according to the current needs. This means that the items that are to be implemented in the near future have the level of detail required for the implementation. Other backlog items are allowed to remain less detailed for the time being. This differs from the classic requirements specification, where the level of detail of the requirements does not depend on the time of realization.

Moreover, the backlog items should be **estimated**. This enables the product owner to plan ahead even for a long period of time. For more information about estimation see chapter 6.3 and for more Information about planning see chapter 6.4.

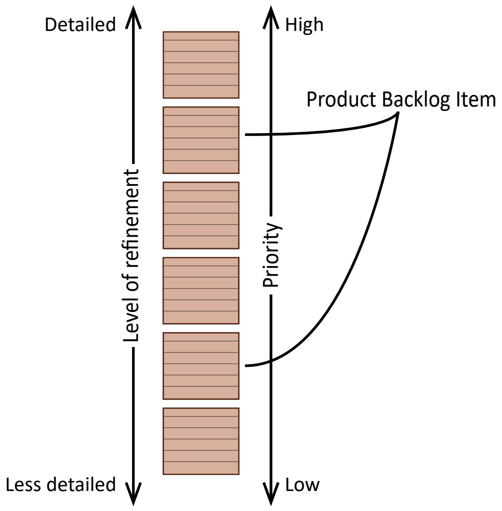


Figure 5.2: Product Backlog

GIF www.sophist.de/bage/k5a1

An important property of the backlog is its **emergence**. This means that the backlog is subject to constant change. The order of the backlog items or the backlog items themselves can change at any time. Items can disappear from the backlog or new ones can be added at any time. Hence, according to the definition, a product backlog can never be complete. In our opinion, this is the key difference to a requirements specification, which should be complete at a given point in time.

The product backlog should also be **prioritized**. This means that the product owner must be aware of the value of the individual product backlog items in order to sort these correspondingly in the backlog. This is because the product backlog is implemented iteration by iteration from top to bottom. Therefore, the items with the highest value should be at the top so that they can be implemented first.

5.2 From Product Backlog Items to User Stories

As described in the previous section, in the product backlog we will find the product backlog items (PBI).

Everything the PBIs can be is like a colorful flower bouquet and goes from epics, user stories, stories over bugs, incidents to, well quite simply, product backlog items. The term user story or simply story is often used because the backlog items are described much like a short story of a future system user. There is a sentence template for this purpose, which is structured as follows:

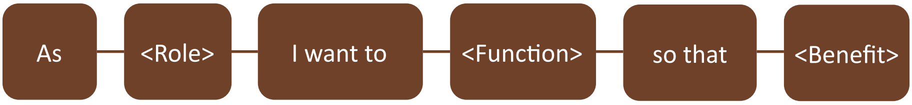


Figure 5.3: User Story Template

This template ultimately answers three “W”-questions:

- **Who** wants to accomplish something?
- **What** they want to accomplish?
- **Why** they want to accomplish that thing?

Especially the latter part causes several problems for many aspiring product owners when it comes to phrasing it. However, this statement is definitely highly desirable, since we have to think about the „why“ of a function. After all, we invest time and money to implement this function in a single iteration. But this question is not only important for the raison d‘être of the PBI. The answer to the “what for” or “why” question also serves another important purpose. It is extremely valuable for the developers to understand what goal the user is aiming for with the function. Thus, this kind of phrasing enables a goal-oriented realization of the PBI by the development team.

A PBI usually also includes acceptance criteria. These are defined by the product owner and describe the criteria according to which he/she will accept the product increment at the end of the sprint. Acceptance criteria are formulated more concretely than a user story and supplement or detail the contents of a story accordingly. Therefore, the acceptance criteria should be formulated according to specific templates.

No matter what terms are chosen for the PBIs, an important task of the product owner is to prioritize the backlog items and assign a value to each backlog item. The product owner must create as much value as possible and establish a good balance between the resources used (the work of the scrum team) and the value of the result.

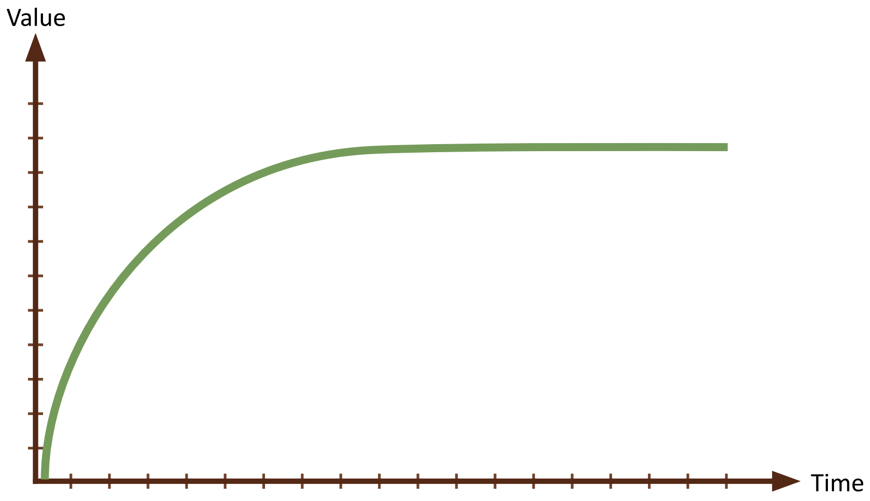


Figure 5.4: Value graph

As we can see in the graph, an important goal of the product owner is to create as much value as possible in the early stages of the product‘s development. Logically, PBIs with lower value then remain, which causes the curve to flatten out. At the latest, when the curve no longer has a slope, one should consider whether further development should be stopped. However, this curve is rather ideal-typical than corresponding to reality. Through prioritization, however, a product owner should approach this curve as closely as possible.

SOPHIST

Competence and expertise

par excellence

Method inventor

Speaker

Author of books

P. Hruschka Ch. Rupp G. Starke
Agility kompakt
Tipps für erfolgreiche Systementwicklung
2. Auflage
Spektrum

chris RUPP stefan QUEINS & die SOPHISTen
UML 2 glasklar
PRAXISWISSEN FÜR DIE UML-MODELLIERUNG
4. Auflage
HANSER
EXTRA: Mit kostenlosem E-Book

Agile Softwareentwicklung für Embedded Real-Time Systems mit der UML
HANSER

chris RUPP & die SOPHISTen
7. Auflage
REQUIREMENTS-ENGINEERING und -MANAGEMENT
Das Handbuch für Anforderungen in jeder Situation
EXTRA: E-Book inside
HANSER
SOPHIST

SOPHIST GROUP / Chris Rupp
Systemanalyse kompakt
2. Auflage

Klaus Pohl • Chris Rupp
Basiswissen Requirements Engineering
Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering Foundation Level
dpunkt.verlag
IREB

Consultant Coach Trainer

We offer you:

We support you competently, energetically and expendiently both in adapation 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 oragnization
- Elicit, analyze, convey and document requirements and architectures appropiately
- 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

6. Deriving Good Requirements and Communicate Them

6.1 Decomposition of Backlog Items

In a product backlog we can distinguish product backlog items (PBI) in different levels of detail. One common way is to divide the backlog items into three different levels of granularity:

- Epics are generally **rough descriptions**, i.e. they are very large and vague. Example: As a house resident, I want the building to be monitored to prevent unauthorized access.
- Features are **medium-detailed**, meaning they are medium sized and slightly more detailed than epics, but still too large to implement in a sprint. Example: As house resident, I would like to have an automated control of the windows to prevent unauthorized access.
- User stories are **fine-grained**, i.e. they are small and detailed. Example: As house resident, I would like the windows to be closed when all the residents are absent, so that no window remains open.

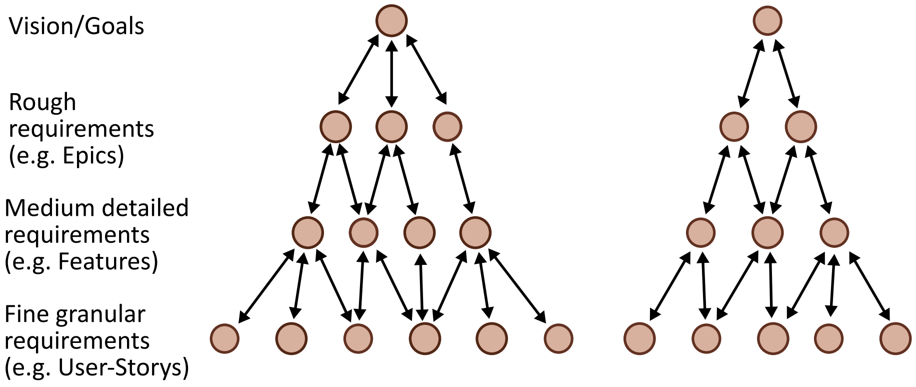

Vision/Goals
Rough requirements (e.g. Epics)
Medium detailed requirements (e.g. Features)
Fine granular requirements (e.g. User-Storys)

Figure 6.1: Parent-child relationship between requirements

However, there is no criterion for classifying requirements into epics, features, or user stories. Rather, it is a rough classification. Sometimes, it may also prove useful to distinguish only between rough and fine requirements. And as mentioned earlier, epics, features, and user stories are only one way to distinguish between backlog items.

The task is to break down the backlog items, i.e. the requirements, into a practical size. There are several reasons for doing this:

- Requirements are to be implemented in sprints. Therefore, the requirement should be detailed enough so that it can be implemented in one sprint.
- The needs of the stakeholders are usually of a more concrete nature. In our example, the residents of the house do not just want any smart home system, but certainly have more specific wishes.
- The understanding of the content that is transported by requirements often only becomes clear when the team thinks about the details.

These are just a few of the reasons why we break down rough requirements into more detailed requirements.

When decomposing the backlog items, we often follow the INVEST principle, which was defined by Bill Wake in 2003. The INVEST principle describes properties or criteria that the backlog items should fulfill. We can broadly say that the more detailed a backlog item is, the more seriously we take the INVEST principle.

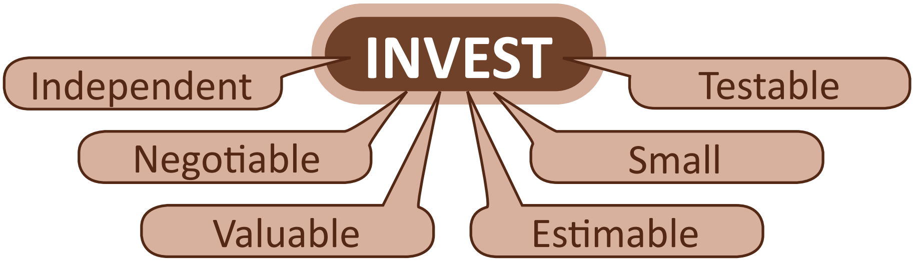


Figure 6.2: The INVEST principle

INVEST means the following:

- **Independent**: Backlog items should be independent of each other. This means that a backlog item can be considered on its own and implemented without the need for further backlog items.
- **Negotiable**: Backlog items are negotiable. They do not represent a contract that has to be implemented in exactly the same way, but leave the developers enough freedom to work out the details in a sprint together with the stakeholders.
- **Valuable**: Backlog items provide a value. Usually a value for the customer, or for the future user. In this context, it is irrelevant how large a backlog item is. Even the smallest backlog items must deliver a value. If this is not the case, then you should seriously question their existence and the need to implement it.

- **Estimable**: Backlog items must be estimated (see chapter 6.3). This estimation helps, for example, to sort the backlog item into the product backlog, or to be able to make a statement about how large the backlog item is. In order for the backlog item to be estimated, it needs to be understood by everyone involved. Otherwise, estimation will not be possible.
- **Small**: Backlog items should be small. The smaller they are, the more specific they usually become. But this is not the only decisive factor; the sprint length also plays a role. After all, the backlog item should be able to be fully implemented in one sprint. Accordingly, we have to choose the size so that the item can fit into the sprint but also leave enough buffers so that the smallest surprise does not immediately jeopardize the implementation in the sprint. But be careful. Do not tend to cut the backlog items too small. This will only increase the administrative workload without adding any value.
- **Testable**: A backlog item should be testable. This means that it is sufficiently well understood as to be able to describe how it can be verified whether the backlog item has been implemented according to the wishes. The use of acceptance criteria for the respective backlog items is suitable for this purpose.

You are certainly wondering yourself how can you satisfy certain aspects of the INVEST principles regarding the rough backlog items. This question is quite justified. The complete INVEST only applies to detailed backlog items. However, less detailed backlog items should at least satisfy the first three aspects, i.e. the INV.

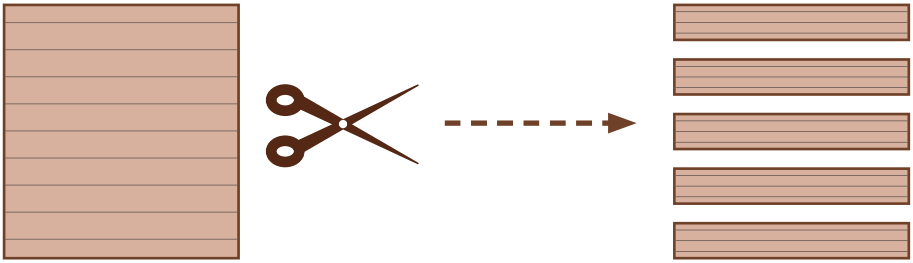


Figure 6.3: From rough to detailed

As soon as the time to implement backlog items comes closer, you have to make sure that they are of a proper size. However, you are not completely alone in this: you can also ask developers for help. For example, backlog items are often decomposed in a common backlog refinement meeting. But we can‘t always do everything in the big picture. After all, developers have other tasks as well. For example building a system out of backlog items.

This means you can‘t avoid decomposing or refining backlog items - depending on which term you want to use - on your own. You could say that we do preliminary work before we put the finishing touches on the backlog items in the common meeting. In any case, you should keep the INVEST principle in mind, so that the detailed backlog items will ultimately be good backlog items. There are a variety of approaches according to which criteria a rough backlog item can be split.For example, you can go by workflow steps, or by data, etc..

Take into consideration that an approach to splitting backlog items that worked well once will not automatically always work. You will have to vary depending on the given circumstances. However, because you will continue to learn more and more, over time you will notice that splitting becomes easier and easier. In general, however, we advise against splitting by system or software units. It may seem easier for developers to have backlog items for the frontend and backlog items for the backend. But you will quickly lose sight of the goal, which is a useful, functional, high-quality product for the future user.

6.2 Discuss Backlog Items

Product backlog items are a communication promise according to the 3C principle (card, conversation, confirmation).

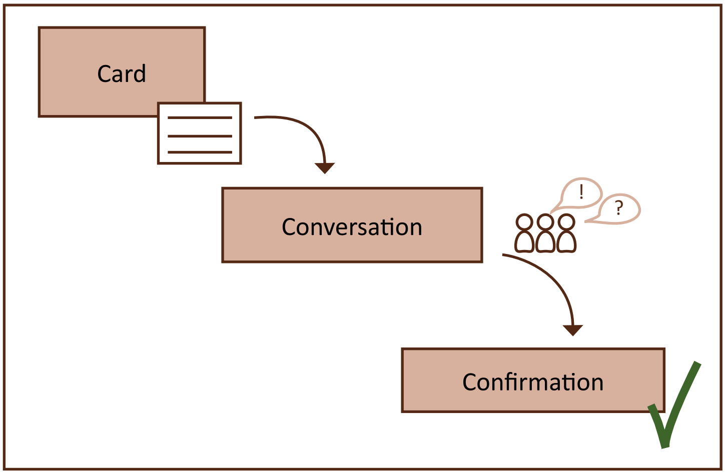


Figure 6.4: The 3-C Model

It is primary intended to be talked about. Important conversations about the backlog items take place between the product owner and the developers. This is because the product owner has the knowledge about the information behind the PBI and the development team needs this knowledge or understanding in order to implement the PBI.

In the agile world are various points in time at which these conversations can take place. For example, in the context of the sprint planning or probably more intensively in a refinement meeting between PO and developers.

The goal of these discussions is to ensure that

- PO and the development team have a common understanding of the PBI
- Open issues or matters about the PBI have been identified
- In the case of a PBI that is to be implemented in the upcoming Sprint, whether the details of the PBI are known to everyone involved.

Most of the time, these discussions are conducted in a way that the product owner opens the backlog item, briefly describes it and then all the participants discuss it. You may have already had the experience that these discussions are not entirely satisfactory.

- The discussion quickly drifts into topics that do not belong in the PBI
- The solution is already being discussed in detail, although the problem is not even yet for all understood
- After half an hour of discussion, the team no longer knows what decisions were made at the beginning
- The discussion group loses sight of the goal of the discussion

Es kann also viel Zeit investiert werden, das Ergebnis aber nicht angemessen ausfallen. Deswegen sollten Sie sich für diese Gespräche im Vorfeld schon Zeit nehmen und sie auch vorbereiten.

Perhaps you would like to structure the discussion by working with the team to create another development artifact (e.g., test cases)? These are just a few examples of the different methods for successfully conducting a meeting to discuss PBIs, which we have successfully used in our projects.

6.3 Estimate

Estimating backlog items is not a requirements engineering activity per se. However, we need this estimation for other requirements engineering tasks (e.g. checking the requirements for comprehensibility) and therefore we dedicate a few words in this brochure for this.

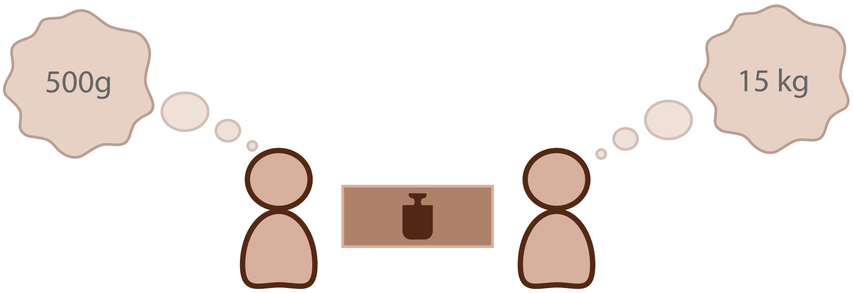


Figure 6.5: Estimation

The backlog items must be constant estimated regarding their size. This not only serves to fulfill the properties of a backlog (DEEP see chapter 5.1), but also has other reasons. First, the estimation allows a statement about whether a backlog item is small enough so that it can be implemented in a sprint or whether it needs to be broken down further. In addition, a statement can be made about how simple or complicated a backlog item is. Release planning can also be created with the starting point being an estimation (see chapter 8).

Probably the most important reason for estimation is not the estimation result itself, but the estimation itself. In agile frameworks, estimation is not performed by individuals, but always by more individuals belonging to a group. It makes sense that this is done by those who have to implement the backlog items, i.e. the developers. If several people estimate a backlog item, there will logically be deviations in the estimates of each of them. If these deviations are large, then this is always a clue that in those group different ideas about what is hidden behind a backlog item exists. For us, this is a clear indication that there is still a need for clarification and coordination within the group.

It is easier to make an estimate relative to each other than absolute. Let‘s imagine two people in a photograph. If we do not know these persons, we cannot say how tall these persons are. But we can make statements about how the size of one person is in comparison with the size of the other person. In addition, by estimating relatively, we take away the worry of having to estimate an absolute effort against which we are subsequently measured. Accordingly, we do not estimate the effort behind a backlog item, but the complexity of the effort.

Estimation is the process of determining the complexity of a backlog item. This is then recorded in story points, T-shirt sizes or other units so that it is cannot be seen as an effort and no conclusions can be drawn about performance.

There are various methods for estimating, such as Planning Poker, T-shirt sizes, Magic Estimation or Wall Estimation, to name the most popular ones.

6.4 Prioritize

In chapter 5.1 “The Product Backlog“ we have said that the product backlog items are to be sorted in the product backlog. Sorting the backlog items is a task of the product owner and has two main goals

- To make the priority of a backlog item visible;
- the higher a backlog item is in the backlog, the higher is its priority
- To direct the concentration to the important backlog items;

the higher a backlog item is in the backlog, the more we need to deal with it.

In order to sort the backlog items we need to have knowledge about its priority However, as a product owner, we are not alone in determining this priority. Depending on what the priority means, we can get useful help from stakeholders or the developers. It is only our responsibility that the priority will be determined.

In order to set a priority, we must become aware and define the criteria we will be using to determine the priority. We determine the criteria by asking ourselves what is important at the moment. This could be, for example, that we want to provide the future user of the system with the most helpful functions or it may be that certain quality requirements are very important at the moment. These are only two examples of what can be understood under priority criteria.

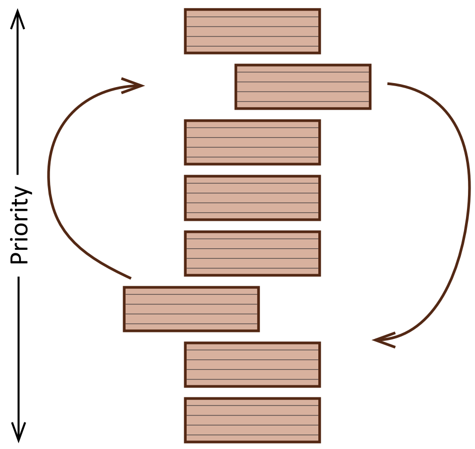


Figure 6.6: Sorting the product backlog according to priority

In essence, our first concern should be to prioritize the value for the future user(s). Because they are the ones for whom we develop the product. Of course, there may be cases (and we are sure there will be) where other criteria play a greater role. But whenever there is no other particular reason to deviate from this, the user/customer satisfaction should come in first place.

Various techniques are available for the prioritization itself. Well-known techniques are MoSCoW, Kano model, linear prioritization or Weighted Shortest Job First.

It is important to know that prioritization is not a one-time thing, but is to be done on a regular basis. Because every day new requirements can appear, which then have to be prioritized accordingly. Moreover, the criteria for prioritization can also change.

7. Product Backlog and other Artifacts

A central artifact within agile frameworks is the product backlog. This is where we find all currently known requirements. If we are being honest, however, not all of them. Some requirements are better off elsewhere, for example in the definition of done.

In addition to the requirements, there are other ways to document knowledge. For example, in addition to the product backlog, a data model in the form of a class diagram may be helpful. Perhaps process descriptions are helpful when discussing the backlog items in order to see the backlog items in their context. These could be documented with BPMN or activity diagrams. It might be helpful for developers to understand what kind of people the eventual users are. A set of documented personas could serve this purpose. This is just a small preview of various additional documentation for the Product Backlog. Everything that helps and makes sense in order to strengthen and promote the common understanding of the stakeholders' needs is permitted and desired.

8. Roadmap, Release Planning

8.1 Roadmap

By working in sprints, the focus of the work is often on the current and the upcoming 1-2 iterations. This means we can make statements about what is being worked on in the current iteration and what the scope of the upcoming iteration will be. Possibly the scope of the next but one iteration is also foreseeable, but usually we do not look further into the future.

However, the distant future may also be interesting for us. There may be restrictions in terms of budget or deadlines. Users want to know when they can expect which features in the product. Developers are also interested in what is coming in the near and distant future. These are just a few reasons why we need to take a look into the future every now and then, even when working within an agile framework.

In order to make this possible we need

- The backlog items
- The priority of the backlog items
- An estimation of the backlog items
- The velocity of the developers

With the help of these four elements, we can now arrange the backlog items on a timeline and roughly but accordingly estimate the time at which we expect a backlog item to be implemented. For this purpose, it will be helpful to divide the timeline into manageable sections, for example, one year into four sections.

The backlog items are then assigned to these sections. We take into account the priority of the backlog item, the estimate and the velocity.

In this way we achieve an approximate planning of the backlog items even over a longer time horizon.

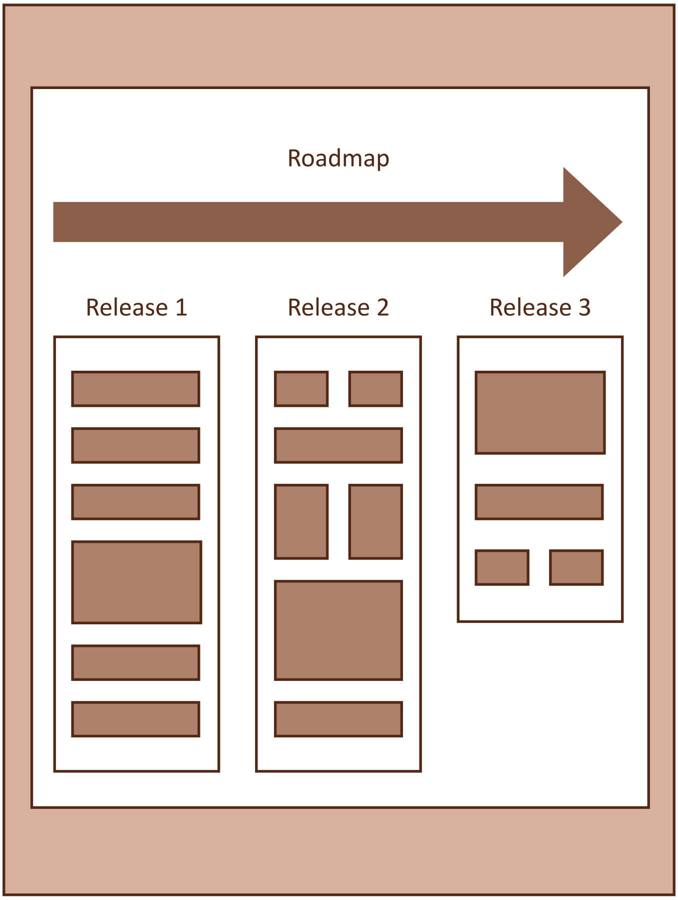


Figure 8.1: Release planning with roadmap

Important: These so-called roadmaps only reflect the current assessment. It must be updated regularly and must not be seen as a commitment that the scrum team is making.

Generally, the further the plan extends into the future, the less accurate it will be. After all, many more unforeseen events can happen in a year from now than in the next two weeks. It is therefore important to regularly check the plan and update it if necessary. This is the only way to recognize at an early stage whether the plan will work out or not.

If the plan does not work out, then this is a sign that you find yourself in the real world. Whereby, we assume with this statement that your plan reaches further into the future than maybe 1-2 months.

Once we realize that the plan is not working out, we have several options as means of reaction:

- Changing the scope of the product
- Postpone the completion date
- Assign more teams for the implementation

The first two options are usually easier to implement, as the addition of further employees naturally requires a training period. However, this is not true in all cases. Sometimes the deadline simply cannot be changed.

8.2 Development Strategies

A possible targeted development strategy is directly related to the roadmap. This is because it does not always make sense to work through the backlog items in exactly the order that the priority actually specifies or would specify.

We may start with the backlog items that we can implement the fastest. Or we want to be able to show something to the stakeholders as early as possible. But maybe we want to try something out first, and then find out how future users will handle it.

So there are different reasons to decide for a certain order of processing backlog items. Depending on which reason plays a bigger role for us, we will choose a certain development strategy.

Common development strategies are:

- Minimum Viable Product (MVP)
- Minimum Marketable Product (MMP)
- Low Hanging Fruits/Quick Wins
- Weighted Shortest Job First (WSJF)
- Risk Reduction

SOPHIST Self-productions

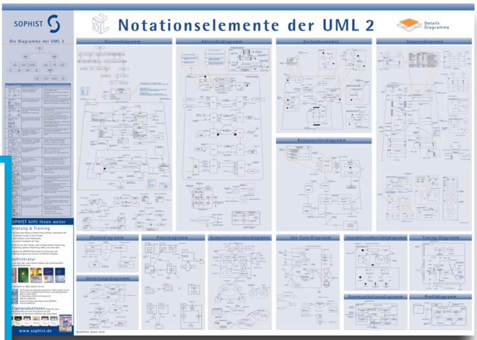
A different kind of knowledge carriers!

Freebie!

WISSEN
for free

Posters

The SOPHIST UML-poster

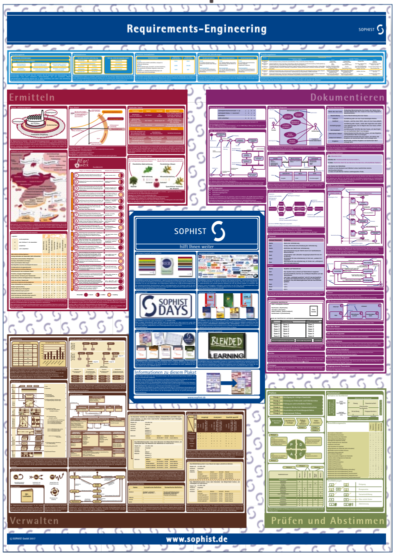


SOPHIST

Notationselemente der UML 2

Details Diagramme

The SOPHIST RE-poster...



Requirements-Engineering

SOPHIST

Ermitteln

Dokumentieren

SOPHIST hilft Ihnen weiter

SOPHIST DAYS

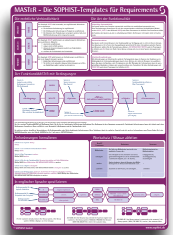
BLENDED LEARNING

Verwalten

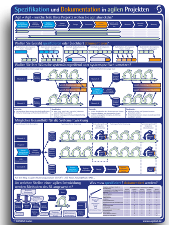
Prüfen und Abstimmen

www.sophist.de

... and its „cores“



MASTeR – Die SOPHIST-Templates für Requirements



Spezifikation und Dokumentation in agilen Projekten

9. Implementing Agility in the Organization

If you are faced with the challenge of establishing agile development practices for the first time, you have your work cut out for you.

First of all, we need to ask ourselves how agile we want things to be. Because we do not always work completely agile. Sometimes we use hybrid approaches to take advantage of agility when, if we cannot work completely agile due to constraints.

Another aspect that is interesting is the question of how do we introduce agility? We can take a top-down approach, where agility is introduced from management. Or we may prefer the bottom-up approach, where the employees themselves are involved in shaping agile working. Especially the latter offers the opportunity to make the introduction of agility itself agile. There are newer interesting approaches here, such as open space agility (OSA)

A major hurdle in the introduction of agility is often the need of change of the mindset among those involved. Developers now work in a self-organized way, the focus is on the product and the best product for the user/customer. We have higher transparency regarding who is working on which topic. It is more about a common understanding than concrete specifications of what exactly should be built. Achieving this change in mindset is a challenge in itself.

Another challenge is the environment. By this we mean the people who are not part of the scrum team, for example. Let‘s take the stakeholders. For them, agility can be completely unfamiliar because they may be very waterfall-driven themselves.

As we are often involved in the implementation of requirements engineering and recently particularly in the agile context, we have gathered our experience in an innovation project in order to provide a set of measures that can make this implementation successful.

10. Agile Working with Multiple Teams

In order for teams to work well together, they must not be too large. It is said that from about 9 people in, a team becomes too large to work effectively. If the product to be developed has a scope where a team would need a relatively large amount of time to build the product, then it may make sense to build the product with several teams at the same time.

If several teams are working on one product, additional challenges arise:

- Responsibilities of the teams must be clarified
- Dependencies between the teams must be made visible and reduced
- Dealing with cross-team issues (e.g., consistent design of the user interface)
- Maintaining an overview of the product as a whole
- Integrating the results of the individual teams into one product

10.1 Scrum@Scale

The Scrum@Scale metamodel uses scrum and scales it for multiple teams. In doing so, the developers of this metamodel have taken care to define as few additional rules as possible as in scrum itself.

Scrum@Scale sets the team size to a maximum of 5 developers, a product owner and a scrum master. Several such teams form a unit, whereby a unit has up to 5 teams. If there are more than 5 teams, multiple units are formed (cf. figure 10.1).

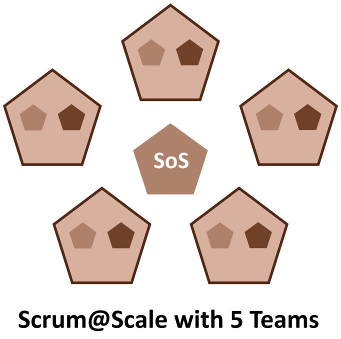


Figure 10.1: Scaling in Scrum@Scale

Let‘s assume we have up to 5 teams working on a product. Therefore, we would also have 5 scrum masters and 5 product owners. The scrum masters in turn form a scrum master team and the product owners form a product owner team. The scrum master team works together to improve collaboration and the product owner team ensures that the product backlog is filled, refined and prioritized. If it becomes difficult to make decisions in the product owner team, there is a chief product owner who then will have the final say.

10.2 LeSS

LeSS stands for large scaled scrum. The idea behind it is to extend classic scrum for multiple teams while making as few changes or giving as few restrictions as possible. The goal is to keep the flexibility and principles of scrum, but scale it to work with more people.

Introduction in LeSS-Framework:

The framework is designed for up to eight teams. In the event that more teams need to work on the same product, there is LeSS Huge, an extension to LeSS that introduces additional roles.

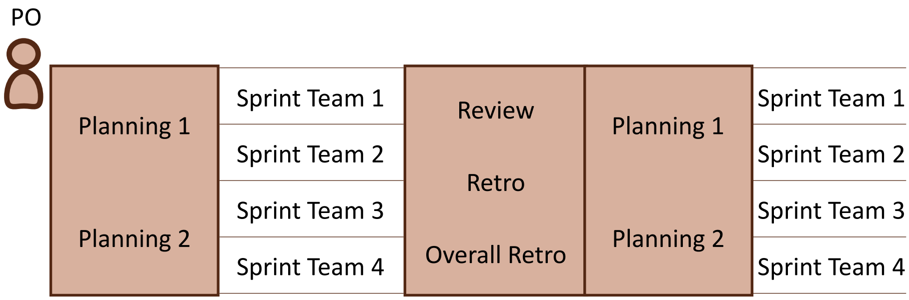
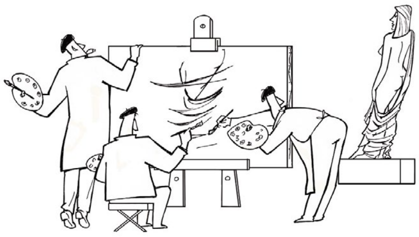


Figure 10.2: Working with the LeSS framework

The teams at LeSS are set up in the same way as in a traditional scrum. However, all teams have the same PO and work with the same product backlog. Scrum masters can supervise one or more teams if needed. In scrum, an interdisciplinary team develops a product. In LeSS, multiple interdisciplinary teams develop a product together, with each team focusing on a different functionality (feature) desired by the customer. This is why LeSS also refers to feature teams. The work of the feature teams is synchronized so that they all have the same sprint length and the same start and end time for the sprints.

Sprint planning is divided in two phases. In sprint planning 1, the teams or representatives of the teams meet with the product owner to clarify the sprint goal and decide which product backlog items should be implemented by which team. Each team keeps its own sprint backlog and plans in sprint planning 2 how it will implement the selected product backlog items.



Product backlog refinement should be done, if possible, with all teams or at least several teams together, as by doing so it will promote a shared understanding of the product vision and communication between teams. If all feature teams gather in a common refinement meeting, LeSS can bring together up to 72 people. Add to that product owners and, as needed, stakeholders, users, or subject matter experts to clarify questions and refine requirements. For a workshop of this size to work, good preparation and moderation are required. We would be happy to advise you on planning agile workshops and help you find out which of the many techniques and methods for knowledge transfer in groups and for communicating and documenting requirements are suitable for you.

Similar to sprint planning, the retrospective also takes place in two parts. First, each team conducts its own retrospective, as in classic scrum. In addition to the cooperation within the own team, the work between the different teams is also discussed in order to identify overlapping obstacles. Subsequently, the overall retrospective takes place, where the PO, scrum master, representatives of the teams and, if required, also the management meet. Cross-team obstacles in the collaboration or structural problems in the organization that prevent the teams from performing their tasks are discussed. Recipes for success and best practices that a team has discovered can also be shared here.

LeSS Huge – Expansion of LeSS to More than 8 Teams

To ensure communication and a smooth workflow when working with many teams, LeSS Huge introduces a new role. The product owner gets support from several Area Product Owners (APO), each of which is responsible for a sub-area of the planned functionality of the overall product. As with the individual product backlog items, the topic areas for APO assignment are also split up by subject matter so that a customer-facing functionality can be implemented as a whole.

10.3 Nexus

Similar to LeSS, nexus also builds directly on scrum and makes relatively few changes. In contrast to LeSS, however, a few more new roles, artifacts and events are defined.

Nexus works with 3 to 9 development teams, which all have the same sprint start and end. For all teams there is exactly one product owner. This is often supported by requirements engineers who are a part of the teams.

A special feature of nexus is the nexus integration team (NIT). This team consists of the product owner, a scrum master and other people who can also come from the development teams. The task of the NIT is to support the integration of the results of the individual teams into a common product. The NIT has more of a coaching role.

Sprint planning consists of two parts: Nexus sprint planning for all teams, in which backlog items are distributed and prioritized among the teams, and individual sprint planning for each team, that takes place afterwards.

In addition to the product backlog and the sprint backlogs (for the individual teams), nexus also has the nexus backlog. The nexus backlog is used to keep track of dependencies between the tasks of the teams and contains only the backlog items that have dependencies.

An additional extension is the nexus daily scrum, where a representative from each team is present. In the nexus daily scrum potential dependencies in the current work is to be briefly discussed.

Just like LeSS, the retrospective of nexus is divided into two parts. One retrospective for each team and a joint retrospective to highlight how the teams work together.

10.4 SAFe

The SAFe framework is designed to make an entire company agile. It envisions multiple products being developed with many teams and defines multiple levels to manage and synchronize collaboration. In this description, we start at the bottom of the team level and slowly work our way up to management.

Team Ebene

- Several scrum teams work together on one product. There is a common product backlog but each team has its own sprint backlog, PO and SM.
- There is no limitation how many teams can be integrated.
- Several sprints are combined to a product increment of usually 3-4 months. At the end of a product increment it is expected that an integrated increment has been developed.

Program Level

The program manager develops medium-sized requirements (features) from the epics of the higher levels. These are prioritized and then scheduled into the program increments. The program manager is supported by other roles, such as system architects or test managers.

Large Solution Level

This level is needed when the subject of consideration is a larger one. For example, if we want to build a smart factory, then this layer is used to coordinate different program layers with each other.

Portfolio Level

The top level in the company - this is where strategic decisions are usually made, which are added into the product backlog in the form of epics and then prioritized. Often, this role is fulfilled by the company‘s executive board.

11. Bibliography

[LeSS]	https://less.works
[Nexus]	https://www.scrum.org/resources/nexus-guide
[Rupp 21]	Rupp C. und die SOPHISTen: Requirements-Engineering und -Management. Das Handbuch für Anforderungen in jeder Situation, 7th Edition, Munich 2021
[SAFe]	https://www.scaledagileframework.com/
[Scrum@Scale]	https://www.scrumatscale.com/scrum-at-scale-guide/

Requirements Engineering in Agility

The discipline of requirements engineering has been the core topic of SOPHIST for many years. Whereas the earlier years of requirements engineering were primarily characterized by waterfall-like projects, nowadays agile development is becoming increasingly common, especially in software development. This also has an effect on the techniques and working methods in requirements engineering, because the framework conditions are completely different here. However, this does not mean that requirements engineering is no longer carried out in agile development, it just often looks different and is carried out at different times than in the classic approach.
In this brochure, we deal with the areas of agility that involve eliciting, communicating, and deriving good requirements. These are the main activities of Requirments Engineering. Keywords which can be found in this brochure are:

- User Stories
- Product Backlog
- Prioritizing
- Refinement
- Editing User Stories
- Roadmaps
- Scaling aility
- etc.

With all these terms, the main focus is always on our core topic:
Requirements Engineering