Certified Professional for Requirements Engineering

RE@Agile

Syllabus

Practitioner | Specialist

Stefan Gärtner, Peter Hruschka,
Markus Meuten, Gareth Rogers,
Hans-Jörg Steffe

2.1.0 | May 1, 2024

Terms of Use

1. Individuals and training providers may use this syllabus as a basis for seminars, provided that the copyright is acknowledged and included in the seminar materials. Anyone using this syllabus in advertising needs the written consent of IREB for this purpose.
2. Any individual or group of individuals may use this syllabus as basis for articles, books or other derived publications provided the copyright of the authors and IREB e.V. as the source and owner of this document is acknowledged in such publications.

© IREB e.V.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without either the prior written permission of the authors or IREB e.V.

Acknowledgements

This syllabus has been written by: Lars Baumann, Stefan Gärtner, Peter Hruschka, Kim Lauenroth, Markus Meuten, Sacha Reis, Gareth Rogers and Hans-Jörg Steffe.

Review by Rainer Grau. Review comments were provided by Jan Jaap Cannegieter, Andrea Hermann, Uwe Valentini and Sven van der Zee. English review by Gareth Rogers.

Approved for release on February 18, 2022 by the IREB Council upon recommendation of Xavier Franch.

We thank everybody for their involvement.

Copyright © 2017–2024 for this syllabus is with the authors listed above. The rights have been transferred to the IREB International Requirements Engineering Board e.V.

Purpose of the Document

This syllabus defines the educational objectives and a summary of the educational content for the RE@Agile Practitioner and Specialist certifications, established by the International Requirements Engineering Board (IREB). The syllabus provides training providers with the basis for creating their course materials. Students can use the syllabus to prepare themselves for the examination.

Contents of the syllabus

The module RE@Agile adresses professionals with career profiles like *Requirements Engineering, business analysis, business engineering, organizational design,* who wish to extend their knowledge and skills in the area of RE@Agile.

Content scope

On the Practitioner/Specialist level – as in the Foundation Level – Requirements Engineering principles are provided that are equally valid for any system – such as embedded systems, safety-critical systems, traditional information systems. This does not mean that the suitability of approaches for the individual areas, accounting for their particularities, cannot

be dealt with in a training course. However, it is not the goal to present specific Requirements Engineering methods of a particular domain.

This syllabus is not based on any specific software development approach and associated process model that makes a statement about the planning, control and sequence of application of the addressed Requirements Engineering concepts and techniques in practice. It is not intended to particularly emphasize a specific approach, neither for Requirements Engineering nor for software engineering overall.

It defines what constitutes the knowledge of Requirements Engineers, but not the exact interfaces with other disciplines and processes of software engineering.

Level of Detail

The level of detail of this syllabus allows internationally consistent teaching and examination. To reach this goal, the syllabus contains the following:

- General educational objectives,
- Contents with a description of the educational objectives and
- References to further literature (where necessary).

Educational objectives / cognitive knowledge levels

All modules and educational objectives in this syllabus are assigned a cognitive level. The levels are classified as follows:

- **L1: Know** (identify, remember, retrieve, recall, recognize) — The candidate will recognize, remember and recall a term or concept.
- **L2: Understand** (summarize, generalize, abstract, classify, compare, map, contrast, exemplify, interpret, translate, represent, infer, conclude, categorize, construct models) — The candidate can select the reasons or explanations for statements related to the topic, and can summarize, classify, compare, categorize and give examples for the concept.
- **L3: Apply** (implement, execute, use, follow a procedure, apply a procedure) — The candidate can select the correct application of a concept or technique and apply it to a given context.
- **L4: Analyze** (analyze, organize, find coherence, integrate, outline, parse, structure, attribute, deconstruct, differentiate, discriminate, distinguish, focus, select) — The candidate can separate information related to a procedure or technique into its constituent parts for better understanding, and can distinguish between facts and inferences. Typical application is to analyze a document, software or project situation and propose appropriate actions to solve a problem or task.
- **L5: Evaluate** (critique, judge) — Give a well-argued critique of a given artifact; make a profound judgment in a given case.

Note that an educational objective at cognitive knowledge level Ln also contains elements of all lower cognitive knowledge levels (L1 through Ln-1).

Example: An educational objective of the type "Apply the RE technique xyz" is at the cognitive knowledge level (L3). However, the ability to apply requires that students first know the RE technique xyz (L1) and that they understand what the technique is used for (L2).

All terms used in this syllabus and defined in the IREB Glossary have to be known (L1), even if they are not explicitly mentioned in the educational objectives.

The glossary is available for download on the IREB homepage at https://www.ireb.org/en/downloads/#cpre-glossary-2-0.

This syllabus and the related handbook use the abbreviation "RE" for Requirements Engineering.

Structure of the Syllabus

The syllabus consists of **six chapters**. One chapter covers one educational unit (EU). For each EU, teaching time and practice time are suggested. They are the minimum a course should invest for that EU. Training companies are free to devote more time to the EUs and the exercises, but make sure that the proportions between the EUs are maintained. Important terms of each chapter are listed at the beginning of the chapter. They are either defined in the glossary or explained in the chapter.

Example:
Chapter 2: A Clean Project Start (L2)
Duration: 120 minutes + 60 minutes exercise
Terms: Product Vision, Product Goal, Stakeholder, Persona, Product Scope, System Boundary

This example shows that Chapter 2 contains education objectives at level L2, and 180 minutes are intended for teaching the material in this chapter.

Each chapter can contain sub-chapters. Their titles also contain the cognitive level of their content.

Educational objectives (EO) are enumerated before the actual text. The numbering shows to which sub-chapter they belong.

Example: EO 3.3.1 Mastering and using the concepts of telling INVEST user stories

This example shows that educational objective EO 3.3.1 is described in sub-chapter 3.3.

The Examination

This syllabus covers educational units and educational objectives for the certification exams of the

- RE@Agile Practitioner
- RE@Agile Specialist

The exam to achieve the RE@Agile Practitioner certificate consists of **a multiple-choice exam**.

The exam to achieve the RE@Agile Specialist certificate consists of a **written assignment**.

Both exams include exam questions covering all educational units and all educational objectives in the syllabus.

Each exam question may include material from multiple chapters of the syllabus as well as from multiple educational objectives or portions of an educational objective.

The **multiple-choice exam** for the **Practitioner** certificate

- tests all educational objectives of the syllabus. However, for the educational objectives at cognitive knowledge levels L4 and L5, the exam questions are limited to items at cognitive levels L1 through L3.
- can be taken immediately following a course, but also independently of that (e.g., remotely or at a test center).

The **written assignment** for the **Specialist** certificate

- tests all educational objectives of the syllabus at the cognitive knowledge levels indicated for each educational objective.
- follows the task description for RE@Agile Specialist, found at https://www.ireb.org/en/downloads/tag:advanced-level-written-assignment#top.
- is self-paced and submitted to a licensed Certification Body.

The following generic educational objectives also apply to the **written assignment** for the **Specialist** certificate:

EO G1: Analyze and illustrate RE@Agile problems in a context that the candidate is familiar with, or which is similar to such a context (L4).

EO G2: Evaluate and reflect on the usage of RE@Agile practices, methods, processes, and tools in projects in which the candidate was involved (L5).

A list of IREB licensed certification bodies can be found on the website https://www.ireb.org.

Version History

Version	Date	Comment	Author
1.0.0	February 20, 2018	Initial Version	Bernd Aschauer, Lars Baumann, Peter Hruschka, Kim Lauenroth, Markus Meuten, Sacha Reis and Gareth Rogers
1.0.1	September 11, 2018	Typos fixed A couple of EO's reformulated to meet standard style. No change content wise. Statement on important terms clarified	Peter Hruschka, Stefan Sturm
1.0.2	September 24, 2018	Credits for reviewers added	Stefan Sturm
1.0.3	December 17, 2019	Consistent usage of the term refinement meeting and product backlog refinement	Hans-Jörg Steffe
2.0.0	July 1, 2022	Educational objective levels updated New version of chapter 6 Corrections in all chapters Rework on the suggested teaching time and practice time on all chapters Information about Advanced Level exam split added.	Peter Hruschka, Markus Meuten, Gareth Rogers, Stefan Gärtner, Hans-Jörg Steffe
2.1.0	May 1, 2024	New Corporate Design implemented, Cognitive Knowledge Levels synchronized, Term "Advanced Level removed".	Stan Bühne

Content

About the module RE@Agile

The module RE@Agile is addressed to Requirements Engineers and agile professionals. It focuses on understanding and applying practices and techniques from the Requirements Engineering discipline in agile development processes as well as understanding and applying concepts, techniques and essential process elements of agile approaches in Requirements Engineering processes. Certificate holders with Requirements Engineering knowledge will be able to work in agile environments, whilst agile professionals will be able to apply proven Requirements Engineering practices and techniques within agile projects.

1 What is RE@Agile (L2)

Duration: 45 minutes

Terms: Stakeholder cooperation, iterative Requirements Engineering, incremental Requirements Engineering, product owner

Educational objectives

EO 1.1 Know the definition of RE@Agile (L1)
EO 1.2 Understand the goals of RE@Agile (L2)
EO 1.3 Recognize that the responsibility for good requirements within SCRUM is with the product owner (L1)

IREB defines RE@Agile [IREB2017] as a cooperative, iterative and incremental approach with four goals:

1. Knowing the relevant requirements at an appropriate level of detail (at any time during system development)
2. Achieving sufficient agreement about the requirements among the relevant stakeholders
3. Capturing (and documenting) the requirements according to the constraints of the organization
4. Performing all requirements related activities according to the principles of the agile manifesto

The key ideas of this definition are in detail:

- RE@Agile is a cooperative approach:
  "cooperative" emphasizes the agile idea of intensive stakeholder interaction by frequently inspecting the product, providing feedback on it and adapting and clarifying the requirements as all have learned more [AgileManifesto2001].
- RE@Agile is an iterative process:
  This suggests the idea of "just in time"-requirements. Requirements do not have to be complete before starting technical design and implementation. Stakeholders can iteratively define (and refine) those requirements that are to be implemented soon to the required level of detail [LaBai2003].
- RE@Agile is an incremental process:
  Implementation of those requirements that deliver the highest business value, or that mitigate against the most significant risks, should form the first increment. Early increments strive to create a minimum viable product (MVP) or a minimum marketable product (MMP). From then on subsequent increments add to the product, prioritizing those requirements that promise to deliver the highest business value, thus increasing the overall value of the result [Reinertsen2009].

The first goal ("relevant requirements known at the appropriate level of detail") again refers to the iterative approach: "relevant" are those requirements that should be implemented soon. And those have to be understood very precisely (including their acceptance criteria) – especially by the developers. They have to conform to the "definition of ready" (DoR). Other requirements – that are not highest priority yet – can be kept at a higher abstraction level – only to be detailed further as soon as they become more important.

The prerequisite for the second goal ("sufficient agreement among relevant stakeholders") is to know all stakeholders and their relevance for the system under development. As the person responsible for requirements (usually the product owner) you have to negotiate the requirements with those relevant stakeholders to determine the order of their implementation.

Agile approaches suggest valuing intensive and ongoing communication about requirements more than their documentation. Nevertheless, the third goal emphasizes the importance of documentation at an appropriate level of detail (which differs from situation to situation). If an organization has to keep documentation about requirements (for legal purposes, for traceability, for maintenance, and so on) even agile approaches have to ensure that this kind of documentation is produced. However, it does not have to be created upfront. It might save time and effort to create that documentation in parallel to the implementation, or even after the implementation.

Requirements management summarizes all activities to be performed once you have existing requirements and requirements-related artifacts. This includes version management and configuration management as well as traceability among requirements and traceability to other development artifacts. RE@Agile suggests managing the effort spent on such activities carefully to balance cost against benefit. For example:

- Extensive version management can sometimes be replaced by quick iterations leading to product increments (i.e. change history of requirements since they were first encountered is less interesting than their current valid state).
- Configuration management (baselining) is included in the iterative determination of sprint backlogs, i.e. grouping those requirements that currently promise the highest business value.

Therefore, some of the time- (and paper-) consuming requirements management activities of non-agile approaches are substituted by activities based on agile principles.

As explained in the RE@Agile Primer [IREB2017], Scrum has introduced the role of a product owner. This product owner has the responsibility for good Requirements Engineering in an agile environment although other stakeholders (like business analysts, Requirements Engineers and the developers) will assist the product owner in that process and maybe do most of the work related to Requirements Engineering.

In this syllabus – for the sake of shortness – we will refer to the product owner (as responsible person) when it comes to perform requirements related tasks – thereby in no way excluding work done by the other stakeholders mentioned above.

The following chapters of this syllabus will go into more detail on the various aspects of RE@Agile.

Chapter 2 will discuss the prerequisites for successful system development: balancing visions/goals, stakeholders and scope of the system.

Chapter 3 and 4 will discuss handling of functional requirements, quality requirements and constraints on different levels of granularity.

Chapter 5 will discuss the process of estimating, ordering and prioritizing requirements to determine the sequence of increments.

Chapters 2 – 5 mainly emphasize handling the requirements of a single team of developers (of 3 – 9 persons).

Chapter 6 discusses scaling Requirements Engineering for larger, potentially distributed teams, including overall product planning and road mapping.

2 A clean project start (L3)

Duration: 120 minutes + 60 minutes exercise

Terms: Product Vision, Product Goal, Stakeholder, Persona, Product Scope, System Boundary

Even in agile approaches some important prerequisites have to be established before successful iterative and incremental system development work can start.

2.1 Vision and goal specification (L3)

Duration: 30 minutes + 15 minutes exercise

Educational objectives

EO 2.1.1 Apply vision and goal specification (L3)

The system vision or product vision describes the overall goal that shall be achieved with the system/product. Agile literature often refers to the product vision instead of a system vision. Both terms (product or system vision) are interchangeable and are used depending on the particular domain or context of the development activity. The vision is of utmost importance for every development activity. It defines the cornerstone and serves as an overall direction for all development activities. Every requirement should support achieving the system vision [Scrumguide].

The difference between a goal and a requirement can be defined as follows [Glinz2014]:

- A goal is a statement about a desired state of affairs (that a stakeholder wants to achieve).
- A requirement is a statement about a desired property of the system.

In certain domains, the term problem (which is to be solved by a particular system) is used instead of the term goal. From an RE perspective the terms are directly related: a problem is a statement about an existing and undesired property, while a goal expresses the desired future state.

Alternative approaches to the formulation of goals or visions are:

- SMART goal formulation [Doran1981]. SMART is an acronym and stands for Specific, Measurable, Achievable, Relevant, and Time-bound.
- PAM goal formulation [Robertson2003]:
  - What is the purpose (P)?
  - What is the business advantage (A)?
  - How would we measure that (M)?
- Product vision box / Design the box [Highsmith2001]. Create a package for your product that shows the key benefits/ideas of a product to potential customers.
- News from the future [HeHe2010]: Write a short newspaper article supposed to be published the day after a successful product launch describing why the product is so great.

- Vision Boards: a graphical collage of goals, potentially sorted by short term, mid-term and long-term visions. A more formal approach for vision boards can be found at [Pichler2011].
- Canvas Techniques [OsPi2010] like the Business Model Generation Canvas or the Opportunity Canvas (and many similar canvases) offer techniques to create focused and easy-to-understand visual representations.

Whatever form one chooses, every stakeholder has the right to know what the team is striving for. The definition of the vision and initial goals must therefore take place at the beginning of a development effort.

Changes to the vision or goals are rare, but not impossible. They may occur for example when new stakeholders are introduced or because works on the system reveals a fundamentally new understanding of the system or its context. Changes to the vision or goals are likely to have a significant impact on development and should be treated with care.

One option is to work with different vision documents, each covering a different time horizon into the future. At any particular point in time a valid and accurate six month, one-year and two-year vision, agreed with all relevant stakeholders, may be available to the developers. These vision documents are reviewed and updated at regular points in the cycle.

It is also not uncommon to focus on the realization of a particular subset of goals for a certain period of time (e.g. 6 months) and afterwards change the goals to follow another direction. Too frequent changes of the vision/or goals, however, are an indicator that there is no clear direction driving product development [Reinertsen2009].

2.2 Specifying the system boundary (L3)

Duration: 30 minutes + 15 minutes exercise

Educational objectives

EO 2.2.1 Apply the specification of the system boundary to delimit scope from context (L3)

A shared and common understanding of the scope (according to IREB: "the range of things that can be shaped and designed when developing a system") and the context (according to IREB "the part of the system environment relevant for understanding the system and its requirements") of the system is a prerequisite for an effective and efficient development effort [Glinz2014].

Misunderstandings related to the system boundary may lead to:

- The development of functionalities or components that were not under the responsibility of the development effort (the assumed scope was too big)
- The false assumption that functionalities or components that are in fact part of the system should have been developed outside the system (the assumed scope was too small)

The definition of the scope and the context go together by defining the system boundary and the context boundary. The system and the context boundary can be defined by discussing:

- Which features or functionalities have to be provided by the system and which have to be provided by the context? (definition of the system boundary)
- Which technical or user interfaces have to be provided by the system to the context? (definition of the system boundary)
- What is irrelevant for the system, i.e. does not influence its scope in any way?
- Which aspects of the system context can be modified during system development? (definition of the scope)

Keep in mind that the context boundary is always incomplete because the context boundary can only be defined by the things that one explicitly excludes from the system context. Further aspects of the system context include in particular stakeholders. The identification and management of stakeholders is described in 2.3.

The system boundary can be documented and clarified with several techniques, for example:

- A context diagram: documents the system, elements of the context, and their relationship
- Natural language, i.e., a list of features, functionalities, aspects of the context, aspects of the scope and further aspects outside of the context
- A use case diagram: a UML diagram type that models the actors and the use cases of a system. It describes the context/scope on a detailed level and is especially useful for clarifying scope and context.
- A story map: a two-dimensional arrangement of user stories into a useful model to help understand the functionality of the system. It describes the context/scope on a detailed level and is especially useful for clarifying the scope.

The definition of an initial scope must take place at the beginning of a development effort. Scope and context will inevitably change during a development effort because of a new or changed understanding of the system or context. The documentation of the scope and context should therefore be updated regularly.

2.3 Stakeholder identification and management (L3)

Duration: 30 minutes + 15 minutes exercise

Educational objectives

EO 2.3.1 Apply stakeholder identification and management (L3)

Failure to identify and include an important stakeholder in a development effort can have a great impact: Discovering such a stakeholder's requirements late (or missing them altogether!) may lead to expensive changes or a useless system [PoRu2015].

The onion model from Ian Alexander [Alexander2005] is a simple tool for stakeholder identification and classification. The model consists of three types of stakeholders: Stakeholder of the system, stakeholder of the surrounding context, and stakeholder from the wider context. Stakeholders of the system are also called direct stakeholders. Stakeholders from the surrounding and wider context are also called indirect stakeholders.

If a system has a human user, the user is one of the most important direct stakeholders.

The users of a system typically cover a wide spectrum of people with different expectations, attitudes, and prerequisites. Understanding the spectrum of users for a particular system is an important first step. If the number of users is small, it is advisable to get to know them (or their representatives) by personal interviews. If the number of users is large or even unknown, the spectrum of users should be captured with other means, e.g. personas [Cooper2004] that represent exemplary users with extreme characteristics. In the age of data analytics, google analytics and big data, online user behavior can often be analyzed directly for deployed product increments with automated tools.

Indirect stakeholders can be found in the surrounding context of the system and may be just as important for a development effort as the users themselves. Examples include legal representatives, governmental or standardization bodies, audit organizations for compliance verification in regulated markets (medical, transportation, aviation), or looking even wider NGOs, unions or competitors.

Systematic identification of key stakeholders should take place at the beginning of a development effort and the results managed throughout the development effort as a continuous activity. A simple list including contact details and relevant attributes will suffice in most contexts. Changes to this list can occur at any time, either because a stakeholder was initially overlooked or due to changes in the context, such as a new NGO being established. Every participant of a development effort (e.g. developers and product owner) should be aware of the importance of stakeholders and look for signs of new or missing ones.

Depending on the particular system and the domain, existing documentation and legacy systems may also be important sources of requirements, and these should be systematically identified and managed in a similar way as stakeholders, see [IREB2019].

2.4 Balancing of vision and goals, stakeholders, and scope (L3)

Duration: 30 minutes + 15 minutes exercise

Educational objectives

EO 2.4.1 Apply the balancing of visions & goals, stakeholders, and scope (L3)

The definition of vision and goals, stakeholders and the system boundary depend on one another [PoRu2015]:

- The relevant stakeholders formulate the vision and the goals. Therefore, the identification of a new relevant stakeholder may have an impact on the vision or the goals.
- Vision and goals can be used to guide the identification of new stakeholders by asking: Which stakeholder may be interested in achieving the vision/the goals or is affected by achieving the vision/the goals?
- Vision and goals can be used to define an initial scope by asking: Which elements are necessary to achieve the vision/the goals?
- Changing the system boundary (and thus the scope) may have an impact on the vision/the goals. If an aspect is removed from the scope, it has to be verified that the system still has sufficient means to achieve the vision/the goals.
- Stakeholders define the system boundary. Therefore, the identification of a new relevant stakeholder may have an impact on the scope.
- A change of the scope (e.g. to fulfill a goal) requires agreement from the relevant stakeholders.

These interdependencies should be used to balance all three elements and to examine the impact of changing one of the three elements on the other.

Because of these tight dependencies between vision and goals, stakeholders, and scope we recommend treating all these elements together and in a coherent way.

3 Handling functional requirements (L4)

Duration: 195 minutes + 120 minutes exercise

Terms: Functional requirements, theme, epic, feature, user story, acceptance criteria, splitting and grouping techniques, Definition of Ready (DoR), INVEST

This core EU primarily takes a **static view on functional requirements**, i.e. **structuring a larger set of requirements** into abstraction hierarchies.

3.1 Different levels of requirements granularity (L1)

Duration: 15 minutes

Terms: Requirements granularity, hierarchies of requirements

Educational objectives

EO 3.1.1 Know that functional requirements exist on different levels of granularity (L1)

Stakeholders usually communicate requirements on different levels of granularity. Sometimes they ask for big chunks of functionality, sometimes they ask for minor details to be added or changed. It is the job of the product owner (supported by other stakeholders) to elicit all those requirements and structure them [Scrumguide].

Both, coarse grained requirements and fine-grained requirements are useful. The coarse-grained requirements allow the product owner to keep an overview of all functional requirements. This is especially necessary for long term planning, road-mapping, selecting requirements that promise early business value for further investigation, high level estimation and much more.

The fine-grained requirements are necessary to achieve a thorough understanding of details that is necessary for the developers to implement those requirements [Patton2014].

Within Agile the terms theme, epic and feature are used to represent distinct levels of granularity, although no clear consensus has yet emerged as to their exact order. Often the hierarchy Epic-Feature-User Story is preferred, with parts of the hierarchy grouped by themes. See also Figure 1 in [IREB2017] in Chapter 3.1.5 and the discussion in the next chapter.

3.2 Identification, documentation and communication of functional requirements (L4)

Duration: 45 minutes + 30 minutes exercise

Terms: Value-based and design-based and groupings of requirements, epic, theme, feature, stories, T-approach

Educational objectives

EO 3.2.1 Analyze and Master a top-level decomposition of requirements supporting iteration planning and road-mapping (L4)
EO 3.2.2 Analyze and Master different decomposition strategies in the large (L4)
EO 3.2.3 Analyze and Master identification, documentation, and communication of functional requirements on different levels of granularity (L4)

RE@Agile strives for "just in time requirements" [IREB2017]. To determine which requirements should be defined in sufficient detail an overview is necessary.

The vision and the goals may not be sufficient to make such decisions. Therefore, an intermediate goal for a product owner is to come up with an overview of all requirements within the scope of a system, i.e. cover full breadth without much depth. This is often called a T-approach, since this broad overview resembles the horizontal bar of the letter T while the drill-down of those requirements that promise highest business value resembles the vertical bar of the letter T.

Different criteria can be used to decompose the vision or the goals in order to yield a set of high level requirements that – considered together – span the intended scope. Many methods have suggested to use a process-oriented decomposition [GoWo2006] [Lamsweerde2009], i.e. splitting functionality into business processes, use-cases or large stories. This kind of decomposition fulfills the first three criteria of INVEST as discussed in the next chapter, i.e. the resulting processes are independent, negotiable and – most important – valuable.

Alternative decomposition strategies could be based on business objects, on subsystem decomposition of an existing system (i.e. a history based or design-based approach), hardware distribution or geographical distribution of subsystems.

While the results of a process-oriented decomposition are called use-cases or user stories, the resulting chunks of the alternative decompositions are often called epics, themes or features.

Whatever the big chunks of functionality are called, they provide a good basis for (coarse) estimation and can already be ordered, thus creating a roadmap or a preliminary schedule of iterations (or sprints – in the Scrum terminology).

All of these high-level requirements have to be detailed before they can be implemented by the developers (see next chapters).

For communication and documentation stories, epics, feature or themes are often captured on physical cards, collected in the product backlog. Alternatively, many tools are available to capture those requirements electronically.

As soon as higher-level requirements are refined into a set of lower level requirements, hierarchies of requirements are created. Ideally the product owner can manage the different levels, without losing the higher-level groupings after refinement.

Story Maps [Patton2014] are one way to arrange and visualize cards in two dimensions, so that epics and features are still visible even if they have been refined to user stories.

3.3 Working with user stories (L3)

Duration: 30 minutes + 30 minutes exercise

Terms: User story, INVEST criteria, 3C-Principle, acceptance criteria

Educational objectives

EO 3.3.1 Apply the concepts of telling INVEST user stories (L3)
EO 3.3.2 Apply the extension or alignment of themes, epics, features and user stories with additional requirements artifacts to improve communication with stakeholders (L3)

User stories are a popular approach to structure requirements. [Cohn2004] suggests a formula to capture user stories. The three constituents of this formula ensure that

1. we have someone who wants that functionality ("As a user ..."),
2. we know what the user wants ("... I want ...") and
3. we understand the reason or motivation ("... so that ...").

The formula helps to think about who wants what and why, but it is not so much the formalism that makes user stories successful but asking and answering these three questions.

Bill Wake created "INVEST" [Wake2003] as a nice acronym to discuss characteristics of a user story. The meaning of the 6 letters is as follows:

- **I**: user stories should be independent. That is, it should be possible to implement user stories in any order, with minimal dependencies among them.
- **N**: user stories should be considered not as a written contract but rather as a promise for future negotiation, or a memento of past discussions. In the role of a Requirements Engineer, professionals are skilled facilitators of this negotiation between business experts and developers to explore the meaning of the user story.
- **V**: every user story should create value. More details about business value follow in 5.
- **E**: a user story should be estimable. If the effort required for its implementation cannot be estimated the story is not clear enough yet. More details about estimation follow in 5.
- **S**: user stories (in the end) should be small enough to be developed within a sprint and still provide value to the business or other stakeholders. Product owners have to break down stories into such small chunks in order to allow the developers to finish that story in one iteration. Techniques for story splitting are discussed in 3.4.

- **T**: user stories should include acceptance criteria to make them testable. Such criteria are typically written prior to the development of the story. Techniques for specifying acceptance criteria are discussed in 3.5. Product owners should understand the pitfalls of non-testable user stories and be able to reformulate them into testable user stories.

Stories are usually captured on cards. The 3C principle (card, conversation, confirmation) [Jeffries2001] emphasizes that the card is only a means to an end – a physical token in tangible and durable form of what would otherwise just be an abstraction. The card should not be considered as an equivalent for a well-written requirements statement. The physical card (or its equivalent in a tool) is there to trigger a conversation about that subject, to bring product owners, other stakeholders and the developers together to discuss that card, gain a deeper understanding and clarify open issues. No matter how much conversation takes place, the confirmation is the acid test for that story: usually the acceptance tests attached to that story, i.e. the things that the product owner will check when the team claims to have finished the implementation of the story.

While Agile emphasizes the use of user stories, it does not preclude the use of traditional analysis artifacts such as context diagrams, business use cases, system use cases, object models, state chart diagrams, activity diagrams and so on, where appropriate in the context of a particular project (cf., e.g., [Scrumguide]). The skilled product owner should understand how user stories can be used together with other artifacts, and be able to explain their meaning where required to stakeholders.

3.4 Splitting and grouping techniques (L4)

Duration: 45 minutes + 30 minutes exercise

Terms: Compound story, vertical slicing, horizontal slicing, splitting patterns, grouping and abstraction patterns, story map, spike

Educational objectives

EO 3.4.1 Analyze and Master splitting techniques for coarse-grained functional requirements to gain finer, more precise requirements (L4)

EO 3.4.2 Analyze and Master grouping or abstraction of fine-grained functional requirements into coarser requirements to deal with complexity, have a better overview and do higher level planning (L4)

In order to generate user stories that are small enough to fit within a single sprint, larger stories may be sliced into more fine-grained ones. A number of patterns can be applied for this purpose, ranging from reducing the feature list to narrowing down the business variations or input channels [Leffingwell2010]. Where a user story represents an unacceptable level of uncertainty, spikes may be used to focus the developers on the problem during a dedicated sprint.

Even fine-grained user stories should, however, be defined in such a way that they deliver some value for at least one stakeholder.

Decomposition of user stories will result in requirements hierarchies as discussed in 3.1. This hierarchical and 3-dimensional structure can be visualized as a two-dimensional story map [Patton2014]. The horizontal dimension shows bigger groupings (like large stories, epics, features) thus maintaining an overview of the requirements; the vertical dimension allows to attach all lower level details for the bigger groups and order them for assignment to sprints and releases.

Such decomposition and grouping of requirements are often best performed with other team members who may have greater insights into business and technical dependencies that may exist among user stories.

3.5 When should you stop decomposing (L4)?

Duration: 45 minutes + 30 minutes exercise

Terms: Definition of ready (DoR), Definition of done (DoD), estimating, acceptance tests

Educational objectives

EO 3.5.1 Analyze and Master the refinement of requirements to a level that is adequate for implementing those requirements (L4)

EO 3.5.2 Analyze and Master the quality assurance of user stories during agile software development (L4)

The product owner is responsible for continuing discussions with the developers until both sides have a common understanding of the requirements [Meyer2014]. The Pareto principle can be used in assessing when this point has been reached: requirements must not be defined 100% perfectly, but well enough to address the team's key questions and clearly enough that the implementation effort can be estimated. Starting implementation with too many open questions may reduce development speed considerably and cause delays against forecasts.

For this level of joint understanding Agile has defined the Definition of Ready (DoR) [AgileAlliance]. A story is ready when it fulfills the INVEST criteria [Wake2003], especially the last three of the letters:

- The developers have been able to estimate the story and the estimated value is small enough to allow the story to fit into one iteration. Lawrence suggests that the story should not only fit in one iteration, but it should be so small that 6 – 10 stories can be assigned to the next iteration.
- The product owner did provide acceptance criteria for the story. Based on the CCC principle everyone agrees that there has been enough conversation and criteria for confirmation of success in terms of acceptance tests are defined.

For formulating acceptance criteria different styles are available [Beck2002]. They can be informal natural language sentences to be checked after implementation. They could be a little bit more formal using the Gherkin syntax (Given/When/Then) to structure these sentences.

Some methods even advocate to use TDD (Test Driven Development) by formally coding the test cases so that they can automatically be executed after implementation [Meyer2014]. This formal approach – while very precise – may be hard to do and understand for product owners and business-oriented stakeholders.

For the product owner the DoR is the equivalent to the definition of done (DoD) of the developers. DoD defines criteria to determine whether a story has been successfully implemented while DoR defines that the developers have sufficient information about a user story so that it can be "Done" by the developers within a sprint.

Discussing requirements with developers needs time and is best done prior to the sprint planning. Planning can then focus on selecting the right user stories and assigning these to the responsible developers. Ideally, developers will have seen the requirements evolving, and helped the product owner by asking questions and performing estimations.

Different forms of product backlog refinements are possible. Refinement meetings may, for example, be a more efficient way of performing refinement than repeatedly disturbing individual developers. The product backlog refinement and all the surrounding activities consume time from the overall iteration capacity. The scrum guide recommends a maximum of 10% capacity from the developers for refinement: if more time than that is required, the iteration should be replanned for getting more accuracy into the product backlog as this is a warning sign for poor quality of the requirements. Product owner should understand the relationship between iteration length, risk and iteration overhead, and know that there are shorter feedback loops than the sprint itself.

3.6 Project and product documentation of requirements (L2)

Duration: 15 minutes

Terms: project and product documentation, reasons for preserving documentation

Educational objectives

EO 3.6.1 Understand how to distinguish between project and product information/documentation (L2)

EO 3.6.2 Know methods and techniques to preserve information for future use (L1)

For the project team story cards are sufficient reminders of ongoing discussions to serve as a basis for development [Cohn2004]. For developers that have been continually engaged on a project, often the code itself is sufficient to understand what has been done previously.

Beyond those directly involved in project development, however, are many other stakeholders – the customer, the wider business, support teams, related project teams, etc. – whom code and user stories do not provide enough context and structure to understand how the work will impact them. Such stakeholders therefore represent different target audiences for documentation.

Furthermore, the product that results from a development project will itself (hopefully) have a life beyond the project, and may indeed evolve throughout multiple development projects. Requirements Engineering artifacts can typically be classified as relevant to the project only, or forming part of the product documentation that should be preserved for future use.

Separating product concerns from project concerns may represent a good investment in sustainable documentation.

In line with the principles of Agile, effort should only be spent on documentation when that documentation has a consumer, and it is important to obtain regular feedback from that consumer when developing that artifact. Techniques to minimize documentation effort include creating dedicated, wiki-oriented documentation systems, in particular for product-oriented artifacts that will evolve over time [Weerd et al.2006].

4 Handling quality requirements and constraints (L3)

Duration: 90 minutes + 30 minutes exercise

Terms: Quality Requirements, Constraints, Quality Tree, DoD

This EU takes a look at quality requirements and constraints in agile projects. Even though the term "non-functional requirements" is still often used in practice as an umbrella term, we use the more concrete and precise categories "quality requirements" and "constraints" according to [Glinz2014].

Initially quality requirements are often deliberately vague. They have to be captured in their vague format as a starting point. Vague quality requirements and constraints can be refined into more precise requirements. Sometimes concrete functional requirements will be derived from them.

4.1 Understanding the importance of quality requirements and constraints (L2)

Duration: 15 minutes

Educational objectives

EO 4.1.1 Understand the importance of quality requirements in agile projects (L2)
EO 4.1.2 Understand categorization schemes for quality requirements and constraints (L2)

Many agile methods concentrate on functional requirements only and do not put enough emphasis on qualities and constraints [Meyer2014].

Key constraints and qualities envisaged for the system should be made explicit early in the lifecycle of a product, since they determine key architectural choices (infrastructure, software architecture and software design). Ignoring them or learning too late in the project may endanger the whole development effort. Other qualities can be captured iteratively, just in time, as with functional requirements [Meyer2014].

Categorization schemata for quality requirements and constraints (e.g. [RoRo2012], [ISO25010]) can be used as checklists so as not to forget important categories.

4.2 Adding Precision to quality requirements (L2)

Duration: 30 minutes

Educational objectives

EO 4.2.1 Understand the detailing or decomposing of quality requirements and constraints (L2)
EO 4.2.2 Understand the derivation of functional requirements from quality requirements (L2)
EO 4.2.3 Understand the specification of acceptance criteria for quality requirements (L2)
EO 4.2.4 Understand the added value of quality trees (L2)

Quality requirements have to be communicated to the developers in a way that is both unambiguous and that supports requirement refinement and decomposition in the same way as for functional requirements.

Decomposing (or detailing) a quality requirement means specifying quality requirements on a lower level of detail, e.g. by using the generalizations in the categorization schemes like "usability" and making them more precise by finding requirements for "ease of use" and "ease of learning".

Deriving means that quality requirements can be achieved by defining functional requirements, i.e. suggesting functions that achieve the desired quality or constraints. An example for refining a security requirement is introducing a role concept and passwords. Quality trees ([BOSSANOVA], [Clements et al.2001]) are also a proven way to structure quality requirements.

For quality requirements acceptance criteria need to be defined as well.

As for other types of requirements, quality requirements should be testable [PoRu2015].

The type of acceptance criteria used will depend on the category of the quality: a quantifiable acceptance criterion for a usability requirement, for example, might be "90 out of 100 users must be able to enter an invoice in less than three minutes".

4.3 Quality requirements and backlog (L3)

Duration: 30 minutes + 30 minutes exercise

Educational objectives

EO 4.3.1 Apply attaching of quality requirements to functional requirements if applicable (L3)
EO 4.3.2 Apply creating separate backlog items for quality requirements (L3)
EO 4.3.3 Understand other quality requirements as part of the DoD (L2)

Generalized quality requirements need to be linked to more specific functional requirements [PoRu2015], e.g., some quantifiable throughput attached to a user story, or specific security features attached to an epic.

Other qualities, e.g. scalability, maintainability, or robustness should be made known to development and checked in each iteration. A common way of achieving that is including them in the Definition of Done. This is often supported by automated testing [Leffingwell2010].

Another approach is to have a separate recording (away from the product backlog) of such qualities to keep them visible for the teams e.g. as a common list or in the form of checklists; such requirements are all on the same rank (because all must be fulfilled) [Leffingwell2010].

It is also good practice to make the relationships of functional vs. affected quality requirements visible by setting up a matrix on a wall, indicating the "affected by" relationship with marks in the respective cells.

4.4 Making constraints explicit (L2)

Duration: 15 minutes

Educational objectives

EO 4.4.1 Understand different kinds of constraints in agile projects (L2)
EO 4.4.2 Characterize the reuse of constraints (L1)

Constraints are an important type of requirements that limit the design choices of the developers [Glinz2014]. Constraints can be categorized as either product constraints or process constraints. Product constraints include required use of technologies, reuse of existing components, make or buy decisions or resources in form of material, knowledge and competence, while process constraints prescribe either organization or development processes. These include organizational policies and regulations, financial limits, norms and standards, compliance regulations and audits, legal and governmental constraints.

It is important to make such constraints explicit so that everyone in the team is aware of them. The most limiting ones should be known early in the project. Others should be captured as soon as they are discovered.

Such constraints are normally applicable to a wider range of projects. So as soon as they are captured once they can easily be reused.

5 Prioritizing and estimating requirements (L3)

Duration: 120 minutes + 90 minutes exercise

Terms: Backlog ordering and prioritization, business value, MVP, MMP, ROI, cost of delay, WSJF, time to market, Planning Poker, Estimation Wall, Cone of Uncertainty, Velocity, Sizing, reference stories, T-Shirt-Sizing, Fibonacci Sequence

5.1 Determination of business value (L3)

Duration: 45 minutes + 15 minutes exercise

Educational objectives

EO 5.1.1 Understand the determination of business value (L2)
EO 5.1.2 Understand how to use business value to order backlog items (L2)
EO 5.1.3 Apply alternative calculations for Business value (L3)
EO 5.1.4 Understand how to align business value measurement to strategic goals of the organization (L2)
EO 5.1.5 Understand the concept of MVP (minimum viable product) (L2)
EO 5.1.6 Understand the concept of MMP (minimum marketable product) (L2)

Agile approaches aim to maximize the overall business value and to permanently optimize the overall business value creation process [Leffingwell2010]. All requirements (whether coarse or fine) should be ordered primarily by the value they can bring to the organization. A prerequisite to doing so is an agreed definition of what business value for this product/company is.

Business value is not only defined by profit: alternative calculations include Return on Investment, Payback Period, Net Present Value, Weighted Shortest Job First (WSJF), Cost of Delay and Balanced Scorecard. Market value, time to market and reducing potential risks all potentially represent types of business value, as do operational and organizational excellence [Reinertsen2009].

Indeed, the definition of business value may be different in every organization, every project, and from the perspective of different stakeholders. Professionals should understand how to align business value measurement to the strategic goals of the organization, and be able to adapt this alignment as these goals change.

One technique to identify business value within the requirements is the KANO model. In general, relative sizing of business value is sometimes preferable to absolute business value calculation.

Key terms within Agile are MVP (minimum viable product) and MMP (minimum marketable product) [IREB2017]. MVP can reduce the effort drastically while the business value remains approximately the same. MVP's can be defined not only for the whole product but also for a particular feature.

While an MVP is great for validated learning, it is not enough for long-term operation; for real deployment to the business the MMP is important. MVP is primarily about scope reduction: produce the business value with the minimal possible feature set and expand later if the feature is successfully used.

5.2 Business value, risks, and dependencies (L3)

Duration: 45 Minutes + 45 minutes exercise

Educational objectives

EO 5.2.1 Apply backlog prioritization (L3)
EO 5.2.2 Apply different basic prioritization strategies (L3)
EO 5.2.3 Apply the determination of requirement dependencies (L3)
EO 5.2.4 Understand the modification of the order of backlog items by considering dependencies (L2)
EO 5.2.5 Understand the dependencies between potential business value and related risks (L2)

Different basic prioritization strategies in the different types of backlogs (enterprise, product, sprint) are possible [Leffingwell2010]. An enterprise might adopt a strategy of early business value gain, for example, if its primary goal were to deliver a product early and establish market share. A strategy of early risk reduction might be preferred if a supplier wanted at all costs to avoid a product return due, for example, to inadequate performance or security.

Very often potential business value and risks are interdependent. Focusing on a specific business value might raise specific risks, changing the focus of the business value might change the risks as well [Reinertsen2009].

In each case the ordering of requirements should be adjusted in line with the selected strategy, taking into account dependencies among the requirements.

5.3 Estimating user stories and other backlog items (L3)

Duration: 30 Minutes + 30 minutes exercise

Educational objectives

EO 5.3.1 Apply forecasts and estimates (L3)
EO 5.3.2 Understand how to derive a mid-term forecast (L2)
EO 5.3.3 Understand the advantage of relative, categorizing and group estimations (L2)
EO 5.3.4 Understand estimation techniques (L2)

Even in a perfect agile world forecasts are needed in order to determine how much work can be "done" within a previously specified iteration (timebox). No un-estimated element is allowed to enter a sprint in scrum for two reasons [Cohn2005]:

1. It is not clear if the element can be completed within the sprint and thereby prevents delivering working software at the end of the sprint.
2. Without discussion and estimation, the team would have no reference point (planning vs. actual doing) for learning for the upcoming sprints.

Additionally, development organizations that exceed one team usually need forecasts in order to prioritize and plan work properly.

Forecasting of development takes place over several time scales. As the understanding of what requirements are assigned to the next iteration should be quite detailed, longer-term iteration plans will have less detail but should allow for large scale estimating and planning.

Such planning also allows reporting of progress, namely which epics, features and user stories have indeed been delivered against their expected release dates. It supports as well detecting requirements to split because they are too large to fit into one estimation [Leffingwell2010].

Initial project estimates are often inaccurate, but become increasingly precise as the activity is iterated (a principle known as the Cone of Uncertainty). By analyzing what has been delivered in previous iterations, the team's velocity can be calculated allowing an improved estimation of the capacity of future iterations [McConnel2006].

To support better mid-term estimates, big requirements like EPICs are broken down into features in order to perform estimates on feature level and sum them up to the EPIC. Those estimates are still not very precise but are helpful for EPIC estimation and serve additionally as a kind of check regarding the knowledge of the EPIC (assumptions etc.).

Agile methods define rules that help to do better and more accurate estimates:

- Everyone involved in the estimation must have the same understanding of the work that needs to be "done".
- Estimations must be performed by those doing the work, the cross-functional team (Developers in Scrum). This helps to bring all involved people on the same level of knowledge by exchanging knowledge and assumptions about the work to be done.
- Estimations should be done relatively / relative to work already done (Estimation by analogy), since those estimates are more likely to be accurate than absolute estimates.
- Estimates should be done in an artificial unit representing effort, complexity and risk in one. Using an artificial unit is necessary to bring everyone on the new way of estimation because the usage of hours / days in estimations tends to reinforce existing, traditional patterns of behavior.
- The usage of the extract from the Fibonacci sequence or T-Shirt sizes help to support that way of estimation (usage of terms like "bigger than/ smaller than" instead of "exactly 2 times as much as ...").

Several techniques support the relative estimation. The most famous technique is the so-called Planning Poker [Cohn2005]. In Planning Poker, the developers discuss the requirement, and everyone selects a card from the Planning Poker set which is based on the Fibonacci sequence to represent the relative size of the new requirement in relation to a common basis. After everyone has selected his or her poker card the team members with the lowest and the highest estimate discuss the reasons for their estimates and try to convince the other team members from their argumentation. Afterwards the next estimation round is started.

If the team cannot agree on one common value within three rounds the requirement is send back to the product owner. If the team can agree the common value is assigned.

The advantage of Planning Poker is, that it is a very good technique for new and unexperienced teams to find their estimates because it avoids anchoring by single team members. The disadvantage is that it is very time consuming. [Note: The book Thinking, Fast and Slow from D. Kahneman [Kahneman2013] gives a great introduction into anchoring and other psychological effects that related to thinking and judgement.]

To overcome this disadvantage for more experienced teams improved techniques are used.

One simplification of the Planning Poker technique is based on the same principles as planning poker but uses a different way of determining the right estimate. Instead of every team member doing a personal estimate one set of poker cards is spread across a table and the reference requirements are placed in the corresponding "container" represented by the poker card. Afterwards the requirements are selected by the team members in a round-robin approach where the team members are allowed either to place a new requirement in the corresponding "container" or reassign one already placed requirement in a different container. If one requirement is reassigned multiple times it will be removed and send back to the Product Owner. This approach is much faster but needs a team that is mature enough to disagree with assignments done by other team members instead of easily agreeing ("anchoring").

The next step of evolvement is usually called "Affinity Estimation" and is used for estimating bigger amounts of requirements e.g. for rough estimation in preparation of Release Plannings. The difference to the previous approach is, that the requirements will not be assigned by round-robin approach, but every team member gets a part of the requirements and assigns it silently to the "containers" represented by the poker card set. After the silent assignment, all involved people are allowed to inspect the assigned requirements and mark those that are questioned. Usually this leads to a quota of 20-30% requirements that need to be discussed and 70-80% that are accepted by all team members.

6 Scaling RE@Agile (L2)

Duration: 105 minutes

Terms: Scaling frameworks, Scaling Requirements and Teams, structuring Requirements and Teams in the Large, Roadmaps and Large Scale Planning, Product Validation

6.1 Scaling requirements and teams (L2)

Duration: 30 minutes

Educational objectives

EO 6.1.1 Describe common examples of scaling frameworks (L1)
EO 6.1.2 Understand the challenges and mechanism for scaling agile requirements (L2)

There are different frameworks and implementations for scaling agility. The frameworks are generalizations of particular implementations, driven by established experts and consulting companies. All scaling frameworks are based on agile values and principles. Each specific framework is based on a selection of agile good practices and particular methods, techniques and other elements combined within a coherent overall concept. Scaling frameworks vary in their maturity level, the number of good practices, guidelines and rules, and the degree of adaptability to the specific needs of an organization.

Since around 2010 a number of different agile scaling frameworks have been developed. Among them are Nexus [NEXUS], SAFe [SAFe], LeSS [LeSS], Scrum@Scale [S@S], BOSSA Nova [BOSSANOVA], Scrum of Scrums [SofS]and Spotify [Spotify2012].

Scaling frameworks are also used because it is sometimes not enough to work with just one team: if either the product is very complex or a shorter time to market is needed, agile development has to be scaled up to involve multiple teams. In addition, geographical distribution or skill distribution could lead to distributed teams. As soon as we have multiple teams working in parallel, the challenge is that requirements have to be coordinated and the communication among the teams has to be defined.

Requirements have to be grouped in order to be assignable to different teams. For these groupings the frameworks suggest different names for different abstraction levels. We strongly recommend that any organization should agree on the names for these different level requirements and define guidelines for the different levels, and then stick to those. These names are often predefined by the scaling framework, or the tools used to capture requirements.

Every team needs someone taking responsibility for requirements management. For a single team this person is usually called product owner, as we have explained in the previous chapters. If requirements have to be managed for multiple teams, some frameworks suggest different job-titles for managing requirements in larger teams. Independent of the job-titles: make sure that there is a clear responsibility for requirements management on every organizational level.

Individual teams work in short iterations, often called sprints (as discussed in the previous chapters). At the start of a sprint a set of requirements is selected from the product backlog, and at the end of the sprint their implementation is checked. When scaling, the frameworks often group these short iterations into longer iterations (for example lasting two or three months instead of 2 – 4 weeks), called releases. This creates the need for release backlogs, setting the context and goals for the participating teams to organize their sprints and the integration of sprint results.

6.2 Criteria for structuring requirements and teams in the large (L2)

Duration: 30 minutes

Educational objectives

EO 6.2.1 Know the principles to organize a backlog and to communicate about requirements within a network of teams (L1)
EO 6.2.2 Understand requirements splitting in different (project) settings (L2)

From a requirements perspective we have to "close the loop" from the initial (business-) requirement demanded by stakeholders, through the splitting of complex requirements into smaller pieces manageable by developers, and then onto ensuring that the assembled results combine to form a solution that can be released to the business.

Sophisticated structures and practices are needed in order to support team collaboration, requirements changes and rapid product delivery in large-scale product development. Otherwise, developers may waste effort coordinating with teams that are not relevant for their work.

Complex requirements must be understood and managed by several product owners and developers. For this purpose, requirements are recursively split into further requirements until they are simple enough to be developed by one team, thus establishing a hierarchy of requirements [GoWo2006]. Communication is required both up and down this hierarchy, for example among product owners working on related requirements (or agreeing on priorities) at different levels of abstraction, as well as among developers, for example in assembling deliverables to form end-to-end product increments.

To properly support collaboration and communication, requirements must be managed using one logical backlog. The key idea is that each requirement is held in one place only, avoiding redundancies or contradictions. This can still be achieved even when further subdividing the backlog into team backlogs. While refining coarse-grained requirements, product owners may work on backlog items not yet associated to any team, or they may split complex requirements and hand the resulting backlog items to the teams for further refinement (see [NEXUS], [SAFe], [LeSS] for further information). To ensure traceability among requirements on different abstraction levels, product owners should link the respective backlog items.

Product development will find it hard to react to changes in a timely fashion if each team depends on a complicated web of interactions with other teams to approve any decision.

A team structure is required that allows teams to self-organize around value creation: to better respond to stakeholder feedback, to make reasonable decisions independently and to deliver end-to-end features [Anderson2020], [Reinertsen2009].

Requirements splitting is necessary to break down coarse-grained requirements so that they can be assigned to developers. To deliver shippable product increments with minimal dependencies on other teams, agile teams should work on loosely-coupled, end-to-end features (compare to feature teams as introduced in [Larman2016]). To achieve this, the scope of the product is partitioned into smaller, loosely-coupled units of internally consistent functionality (i.e. feature-based requirements splitting). Each unit has well-defined functional boundaries. The boundaries should be clear to enable effective collaboration. If requirements are also split according to these functional boundaries, then product owners assigned to a particular unit can work on features with a greater degree of independence.

Unfortunately, in many cases it is not that easy to decompose requirements based around loosely-coupled units of end-to-end functionality. Due to architectural design (for example technology, infrastructure, system components, common platform, architectural layers such as front- and backend) as well as organizational considerations (specialist skills, team location, sub-contractors), units of functionality may overlap. This means that different agile teams must work together to implement specific features and their respective product owners need to collaborate more closely on requirements.

To implement features collaboratively, agile teams require a shared understanding of requirements and their business context. They must also agree on cross-cutting requirements, constraints and common technical interfaces so that deliverables from different teams can be integrated to working increments. Integration and testing of features become more complex and synchronizing teams using backlogs and roadmaps is even more critical. Another approach is to form a separate agile team around certain features or to borrow the engineers necessary to develop specific features independently. See also [Anderson2020] for further details on agile organizational design and practices.

6.3 Roadmaps and large scale planning (L2)

Duration: 30 minutes

Educational objectives

EO 6.3.1 Characterize the difference between a roadmap and a backlog (L1)
EO 6.3.2 Understand the creation and the management of a roadmap (L2)

In large-scale product development, product owners manage requirements in the product-focused backlog. In contrast to the backlog, a roadmap is used for planning product development incrementally.

A roadmap is a prediction of how the product will grow [Pichler2016]. Roadmaps do not change the content of backlog items, but arrange them onto a timeline. It answers the question when we can roughly expect which features.

A roadmap is a useful means to communicate (strategic) goals and decisions to the developers and other stakeholders. It breaks down a long-term goal into manageable iterations, represents dependencies among the teams and provides direction and transparency to the stakeholders.

At the beginning of the agile product development, little is known about the product, or the work done by the teams. Thus, the scope of the product, as well as the cost estimates, are subject to a high level of uncertainty. As more iterations are completed and as more feedback is gathered from the stakeholders, the uncertainty gradually decreases leading to more reliable planning and a stable roadmap. This principle is known as the cone of uncertainty [Boehm1981].

Although this principle is generally true for all agile development projects, it becomes even more important in large-scale product development, as the risks due to product complexity and the potential for misalignment across multiple teams – and consequently the need for more planning – are even greater.

A roadmap shows strategic goals, milestones and coarse-grained requirements. Important milestones may be either internal or determined by external events such as a trade show or the introduction of new regulation to the market. The representation of a roadmap depends on its purpose, target group and planning horizon.

For customers, management sponsors and the business a long-term **product roadmap** containing strategic goals and coarse-grained product requirements is often sufficient [Pichler2016]. Conversely, developers need to know the further details represented by fine-grained backlog items (for example stories and tasks) as well as the dependencies among them. This information is provided by mid-term **delivery roadmaps** [SAFe].

To develop a long-term product roadmap, a product owner must first define a product vision and strategy. Afterwards, product owners must then elicit coarse-grained requirements by engaging with the necessary stakeholders. There is no need to invest time on detailed requirements at this point. Although requirements are subject to a high level of uncertainty at this stage, the product roadmap as a rough, first-cut iteration plan is good enough to support planning and synchronization.

To create a mid-term delivery roadmap, product owners must refine and prioritize the backlog items from the existing product roadmap. These items need to be roughly estimated by the developers, even if the estimates are still imprecise (for example T-shirt sizes) at this stage. The estimate only has to be good enough to provide an overview of upcoming iterations.

Creating and updating delivery roadmaps typically happens at face-to-face planning events known as big room plannings (or PI Planning in SAFe), held at regular intervals.

6.4 Product validation (L2)

Duration: 15 minutes

Educational objectives

EO 6.4.1 Understand concrete methods to validate product requirements in agile development (L2)

The principal idea of agile development is to develop a small slice of the product, generate feedback by involving stakeholders and adapt the product development according to the insights and findings. For this purpose, product owners must use a newly released product version to verify its business value and to examine whether the product requirements were understood correctly.

A sprint review is a suitable means to present a product increment developed in the last sprint to stakeholders. In large-scale product development, the same idea can be used as well. But instead of reviewing a single product slice developed by one team, all team deliverables are integrated to a working product increment worth validating. The increment is demonstrated in a **product review (demonstration)** to the stakeholders leading to a better impression of the entire product [SAFe], [Larman2016], [LeSS].

Another approach for product validation in large-scale product development is **data analysis** [Maalej et al.2016]. The integrated product increment is delivered to users and, based on their behavior, measurements are made as to whether the product features have a positive, neutral or negative impact. Product owners can use the results to identify potentially poorly-designed features. To better understand the identified problems, they may need to again apply regular requirements elicitation and analysis techniques.

DEFINITONS OF TERMS, Glossary

The glossary defines the terms which are relevant in the context of RE@Agile. The glossary is available for download on the IREB homepage at https://www.ireb.org/en/downloads/#re-agile-glossary

REFERENCES

[AgileAlliance] Glossary of the Agile Alliance: Definition of term "Definition of Ready": https://www.agilealliance.org/glossary/definition-of-ready. Last visited April 2024.

[AgileManifesto2001] Agile Manifesto: http://Agilemanifesto.org 2001. Last visited April 2024

[Alexander2005] Alexander, I. F.: A Taxonomy of Stakeholders – Human Roles in System Development. International Journal of Technology and Human Interaction, Vol 1, 1, 2005, pages 23–59.

[Anderson2020] Anderson, J.: Agile Organizational Design – Growing Self-Organizing Structure at Scale. Leanpub, 2020.

[Beck2002] Beck, K.: Test Driven Development: By Example. Addison-Wesley 2002.

[Boehm1981] Boehm Barry W.: Software Engineering Economics. Prentice Hall, 1981.

[BOSSANOVA] BOSSA nova https://www.agilebossanova.com/#bossanova, last visited April 2024.

[Clements et al.2001] Clements, P., Kazman, R., Klein, M.: Evaluating Software Architectures. SEI Series in Software Engineering, 2001.

[Cohn2004] Cohn, M.: User Stories Applied For Agile Software Development. Addison-Wesley, 2004.

[Cohn2005] Cohn, M.: Agile Estimating and Planning. Prentice Hall, Nov 2005.

[Conway1968] Conway, M.E.: How Do Committees Invent? Datamation Magazine , 1968. http://www.melconway.com/Home/Committees_Paper.html. Last visited April 2024.

[Cooper2004] Cooper, A.: The Inmates are Running the Asylum: Why High Tech Products Drive Us Crazy and How to Restore the Sanity. Pearson Education, 2004.

[Doran1981] Doran, G.T.: There's a S.M.A.R.T. way to write management's goals and objectives. Management Review, 1981. AMA FORUM. 70 (11): 35–36.

[Glinz2014] Glinz, M.: A Glossary of Requirements Engineering Terminology. Standard Glossary for the Certified Professional for Requirements Engineering (CPRE) Studies and Exam, Version 1.6, 2014. https://www.ireb.org/en/downloads/#cpre-glossary. Last visited April 2024.

[GoWo2006] Gorschek, T., Wohlin, C.: Requirements Abstraction Model. Requirements Engineering Journal, Vol. 11, No. 1, pp. 79–101, 2006.

[HeHe2010] Heath, C., Heath, D.: Switch: How to Change Things When Change Is Hard. Crown Business, 2010.

[Highsmith2001] Highsmith, J.: Design the Box. Agile Project Management E-Mail Advisor 2001, http://www.joelonsoftware.com/articles/JimHighsmithonProductVisi.html. Last visited April 2024.

[IREB2019] IREB e.V.: CPRE Advanced Level Elicitation Syllabus, 2019. https://www.ireb.org/en/downloads/tag:al-e-c#top. Last visited April 2024.

[IREB2017] IREB e.V.: CPRE – RE@Agile Primer – Syllabus and Study Guide, 2017 https://www.ireb.org/en/downloads/tag:re-agile-primer#top. Last visited April 2024.

[ISO25010] ISO/IEC 25010:2011: Systems and software engineering –– Systems and software Quality Requirements and Evaluation. ISO/IEC Standard 25010:2011.

[Jeffries2001] Jeffries, R.: Essential XP: Card, Conversation, Confirmation, 2001, https://ronjeffries.com/xprog/articles/expcardconversationconfirmation/. Last visited April 2024.

[Kahneman2013] Kahneman D.: Thinking, Fast and Slow. Farrar, Straus and Giroux, 2013.

[Kniberg2012] Kniberg, H.: Scaling Agile @ Spotify with Tribes, Squads, Chapters & Guilds https://blog.crisp.se/2012/11/14/henrikkniberg/scaling-agile-at-spotify. Last visited April 2024.

[LaBai2003] Larman, C., Basili, V. R.: Iterative and Incremental Development: A Brief History. IEEE Computer, Vol 36, No. 6, 2003, 47-56.

[Lamsweerde2009] van Lamsweerde, A.: Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley, 2009.

[Larman2016] Larman, C.: Large-Scale Scrum: More with LeSS. Addison Wesley, 2016.

[Leffingwell2010] Leffingwell, D.: Agile Software Requirements – Lean Requirements Practices for Teams, Programs, and the Enterprise. Addison Wesley, 2010.

[LeSS] Large-Scale Scrum: https://less.works/. Last visited April 2024.

[Maalej et al.2016] Maalej, W., Nayebi, M., Johann T., Ruhe, G.: Toward Data-Driven Requirements Engineering. IEEE Software (Volume 33, Issue 1), 2016.

[McConnel2006] McConnel, S.: Software Estimation, Demystifying the Black Art. Microsoft Press, 2006.

[Meyer2014] Meyer, B.: Agile! The Good, the Hype and the Ugly. Springer, 2014.

[NEXUS] Scaling Scrum with Nexus™: https://www.scrum.org/resources/scaling-scrum. Last visited April 2024.

[OsPi2010] Osterwald, A., Pigneur, Y.: Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers. John Wiley and Sons, 2010.

[Patton2014] J. Patton, J.: User Story Mapping –Discover the Whole Story, Build the Right Product. O'Reilly, 2014.

[Pichler2011] Pichler, R.: Product Vision Board, 2011 http://www.romanpichler.com/blog/the-product-vision-board/. Last visited April 2024.

[Pichler2016] Pichler, R.: Strategize: Product Strategy and Product Roadmap Practices for the Digital Age. Pichler Consulting, 2016.

[PoRu2015] Pohl, K., Rupp, C.: Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam – Foundation Level. Rocky Nook, 2015.

[Reinertsen2009] Reinertsen, D.G.: The Principles of Product Development Flow – Second Generation Lean Product Development. Celeritas Publishing, 2009.

[SAFe] Scaled Agile Framework 4.5 ® http://www.scaledagileframework.com/. Last visited April 2024.

[S@S] Scrum at Scale™: https://www.scruminc.com/scrum-scale-case-modularity/. Last visited April 2024.

[Scrumguide] The Scrum Guide ™ : http://www.scrumguides.org/scrum-guide. Last visited April 2024.

[SofS] Scrum of Scrums https://scrumguide.de/scrum-of-scrums, last visited April 2024.

[Spotify2012] Scaling Agile @ Spotify https://blog.crisp.se/wp-content/uploads/2012/11/SpotifyScaling.pdf. Last visited April 2024.

[Sterling2012] Sterling C.: Affinity Estimating – A How-To https://scrumology.com/guest-post-affinity-estimating-a-how-to/. Last visited April 2024.

[RoRo2012] Robertson J., Robertson S.: Mastering the Requirements Process – Getting Requirements Right, 3rd edition. Addison Wesley, 2012.

[Robertson2003] Robertson, S.: Stakeholders, Goals, Scope: The Foundation for Requirements and Business Models, 2003, https://www.volere.org/wp-content/uploads/2018/12/StkGoalsScope.pdf. Last visited April 2024.

[Wake2003] Wake, B.: INVEST in Good Stories, and SMART Tasks, 2003, https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/. Last visited April 2024.

[Weerd et al.2006] van de Weerd, I., Brinkkemper, S., Nieuwenhuis R., Versendaal, J., Bijlsma, L.: On the Creation of a Reference Framework for Software Product Management: Validation and Tools Support. International Workshop on Software Product Management (IWSPM 2006).