IREB Certified Professional for Requirements Engineering

- RE@Agile Primer -

Syllabus and Study Guide

Version 1.1.0

September 24, 2020

Terms of Use:

1. Individuals and training providers may use this syllabus and study guide as a basis for seminars, provided that the copyright is acknowledged and included in the seminar materials. Anyone using this syllabus and study guide in advertising needs the written consent of IREB for this purpose.

2. Any individual or group of individuals may use this syllabus and study guide 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.

Acknwoledgements

This syllabus and study guide has been written by: Lars Baumann, Peter Hruschka, Kim Lauenroth, Markus Meuten, Sacha Reis, Gareth Rogers, Francois Salazar, Thorsten Weyer.

Review comments were provided by: Bernd Aschauer, Dirk Fritsch, Rainer Grau, Andrea Hermann, Krystian Kaczor, Niko Kaintantzis, Elisabeth Larson, Ladislau Szilagyi, Daniel Tobler, Erik van Veenendaal, Arun Vetrivel, Sven van der Zee.

English review by Joy Beatty, Gareth Rogers and Candase Hokanson.

We thank everybody for their involvement.

Approved for release on March 2, 2017 by the IREB Council upon recommendation of Xavier Franch.

Preamble

Purpose of the Document

This syllabus and study guide defines the foundation level of the certification "RE@AGILE" established by the International Requirements Engineering Board (IREB). The syllabus and study guide provides training providers with the basis for creating their course materials. Students can use the syllabus and study guide to prepare themselves for the examination.

Contents of the Syllabus and Study Guide

The foundation level addresses the needs of all people involved in the topic of Requirements Engineering and Agile. This includes people in roles such as project or IT management, domain experts, system analysts and software developers as well as Scrum teams.

Content Scope

RE@Agile is inspired both by IREB's view of Agile values as well as by an Agile view of Requirements Engineering values. Its content includes classification and assessment of Requirements Engineering artifacts and techniques in the context of Agile, of Agile artifacts and techniques in the context of Requirements Engineering, and of essential process elements in Agile product development. RE@Agile points to the motivation to use Agile in a development process. A very important topic is the synergy between Requirements Engineering and Agile: Agile Principles concerning Requirements Engineering and Agile mindset in relation to the core Requirements Engineering values.

The IREB's RE@Agile Certifications aim to support the following groups of people:

Requirements engineers who want to become involved in Agile development and who aim to successfully apply their techniques in this environment.

Requirements engineers who want to apply established concepts and techniques from Agile approaches to improve their Requirements Engineering processes.

Agile professionals who want to understand the value and benefits of the Requirements Engineering discipline in Agile projects.

Agile professionals who want to improve Agile development by using proven Requirements Engineering techniques and methods.

People from related disciplines – IT managers, testers, developers, architects and other representatives of the business involved in development (mostly, but not only software development) - who want to understand how to successfully combine the Requirements Engineering and Agile approaches in development processes.

Level of Detail

The level of detail of this syllabus and study guide allows internationally consistent teaching and examination. To reach this goal, the syllabus and study guide 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

Each module of the syllabus and study guide is assigned a cognitive level. A higher level includes the lower levels. The formulations of the educational objectives are phrased using the verbs "knowing" for level L1 and "mastering and using" for level L2. These two verbs are placeholders for the following verbs:

- **L1 (knowing):** enumerate, characterize, recognize, name, reflect
- **L2 (mastering and using):** analyze, use, execute, justify, describe, judge, display, design, develop, complete, explain, exemplify, elicit, formulate, identify, interpret, conclude from, assign, differentiate, compare, understand, suggest, summarize

All terms defined in the 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/#re-agile-glossary

This syllabus and study guide uses the abbreviation "RE" for Requirements Engineering.

Structure of the Syllabus and Study Guide

The syllabus and study guide consists of 4 main chapters. One chapter covers one educational unit (EU). Each main chapter title contains the cognitive level of the chapter, which is the highest level of the sub-chapters. Furthermore, the minimum teaching time a course should invest for that chapter is suggested. Important terms in the chapter, which are defined in the glossary, are listed at the beginning of the chapter.
Example: EU 2 FUNDAMENTALS OF RE@AGILE (L1)
Duration: 1 ¼ hours
Terms: Product Owner, Product Backlog, Sprint Backlog, Epics, User Stories, Story Maps

This example shows that chapter 2 contains education objectives at level L1 and 75 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 3.1.2

This example shows that educational objective EO 3.1.2 is described in sub-chapter 3.1.

The Examination

This syllabus and study guide is the basis for the RE@Agile Primer examination. Two different examinations are available:

- Proctored multiple choice examination with official RE@Agile Primer certificate, similar to the CPRE Foundation Level and Advanced Level multiple choice examinations, but 40 minutes duration.
- Online multiple choice self-assessment with confirmation of participation.

Proctored examinations can be held immediately after a training course, but also independently from courses (e.g. in an examination center). A list of recognized examination providers can be found on the IREB homepage https://www.ireb.org/exams/bodies.

The self-assessment will be available via the IREB homepage: http://www.ireb.org

A question in the examination can cover material from several chapters of the syllabus and study guide. All chapters (EU 1 to EU 4) of the syllabus and study guide can be examined.

Version History

Version	Date	Comment
1.0	March 28, 2017	Initial version
1.0.1	May 29, 2017	Fixed minor issues (typos, exam duration)
1.0.2	November 16, 2017	Glossary extracted as separate document. See https://www.ireb.org/en/downloads/#re-agile-glossary
1.1.0	September 25, 2020	Added detail informations about the product backlog; Fixed some spelling and grammar issues. Consequently used “Minimum” (e.g. Minimum Viable Product)

Content

The Vision of RE@Agile

Motivation and Background

The quality of the requirements determines the success or failure of the whole product development process, regardless of the development methodology applied. Contrary to popular belief, techniques and methods from the Requirements Engineering discipline are agnostic to their usage within specific development methodologies (like the waterfall model or Scrum). However, Requirements Engineering is most commonly perceived as a non-Agile development discipline leading to the misconception that the Requirements Engineering body of knowledge has no relevance for the success of Agile development processes.

In many cases, Requirements Engineering and Agile approaches are considered separately rather than together. Whilst in conventional development processes Requirements Engineering is established with dedicated roles as a separate discipline within the lifecycle of a system, in Agile development the importance of Requirements Engineering is often underestimated.

Agile approaches are based on direct communication, simplicity of solutions and feedback. One of their main values is the rapid response to changes. Thus, changes to requirements and their priorities represent an inherent concept of all Agile approaches. In fact, giving appropriate importance to the Requirements Engineering competence in Agile development processes can leverage the success of Agile projects while sustainably increasing the quality of developed systems and products. Conversely, the Requirements Engineering practice can significantly benefit from some very useful Agile principles and techniques independently of the specific development methodology applied.

Currently, in many cases, people are either experts in Requirements Engineering or in the application of certain Agile approaches. As a consequence, people from both sides have to find their own way to leverage the benefits from using principles and techniques from the other competence field. Certifications focusing on the integration of both fields of competences are currently not available, either from the Requirements Engineering community or from the Agile community. Hence it is highly promising to build a widely-accepted bridge between Requirements Engineering and Agile approaches and thus also between requirements engineers and Agile experts so that both can efficiently and effectively communicate with each other.

IREBs' answer to such a demand is the RE@Agile Certification.

About RE@Agile

Such a bridge should be built from two different directions: on the one hand, the Requirements Engineering community needs to understand how to successfully apply their various techniques and methods in Agile development processes, as well as how to apply specific techniques from Agile approaches in order to improve Requirements Engineering practice. On the other hand, since Agile approaches aim to deliver software of value as early as possible, Agile practitioners need to understand how to leverage this effect by applying proven concepts and techniques from the Requirements Engineering discipline.

The foundations of RE@Agile are RE's view on Agile values as well as Agile views on Requirements Engineering values. The content includes the classification, assessment and application of Requirements Engineering artifacts and techniques in the context of Agile approaches as well as the use of Agile artifacts, techniques and essential process elements in Requirements Engineering processes.

The core principles of RE@Agile are:

Requirements Engineering and Agile approaches can leverage each other

RE@Agile analyzes possible benefits and pitfalls of Requirements Engineering and Agile techniques. To this end, RE@Agile addresses the use of artifacts and techniques from the Requirements Engineering discipline in Agile processes as well as the use of artifacts, roles and techniques from Agile approaches in Requirements Engineering processes in the context of different development methodologies.

Lightweight and highly adaptive processes

Based on the philosophy of RE@Agile, the differentiation between predictive and adaptive development processes is of vital importance. RE@Agile proposes the idea of a lightweight and highly adaptive approach to performing Requirements Engineering activities within Agile development. In RE@Agile, Requirements Engineering is a core discipline rather than a single process step: a continuous process that must be performed systematically and that necessitates a high level of skill and experience.

Close collaboration within the team and with key stakeholders, and just-in-time-requirements

Frequent communication and close collaboration among all team members and key stakeholders is of particular importance for the success of Agile development processes. In RE@Agile, the team, together with key stakeholders, elicits, analyzes, refines and documents the requirements in a highly interactive fashion. RE@Agile supports practitioners in selecting the right activities at the right time to ensure high quality requirements before they are implemented.

Situational and selective requirements elicitation, analysis, specification and refinement

RE@Agile is based on the idea that not every requirement needs to be specified precisely and to a low level of detail before the implementation of the system begins. Rather, only the requirements which are overly complex (i.e. are not understandable to stakeholders or the development team) or critical (i.e. cannot risk misunderstanding) are refined and specified and in more detail. The overall process relies on the shared philosophy that changes to functional requirements are welcome and easy to accommodate.

Avoid less relevant activities and functionality and ensure the minimum viable product

One of the Agile principles is "Simplicity". According to this principle the first stage of system or product development in Agile processes is often the MVP (Minimum Viable Product). The MVP is a distinct, deployable system that offers only a base set of features providing just enough business value to end users to allow validated learning. The minimal scope of MVP allows elimination of waste during development and provides an opportunity for fast customer feedback. One of the next product stages is often then the MMP (Minimum Marketable Product) – a product with the smallest set of features that addresses the users' needs and therefore has market value. RE@Agile provides answers to two very important questions, which are quite important even in non-Agile development processes: "How to simplify the release management and product definition process?" and "How to define the MVP or MMP based on the requirements?"

About IREB RE@Agile Certification:

IREB RE@Agile provides two different Certifications and one self-assessment addressing different skill levels:

RE@Agile Self-Assessment (Open to all participants)

The RE@Agile Self-Assessment is addressed to people that want to show their knowledge but are not interested in the Primer or Advanced Level certificate or do not fulfill the prerequisites.

A person with the RE@Agile Self-Assessment Result:

- has shown relevant and measured (Scale 0-100%) fundamental knowledge of the relevant terminology of Requirements Engineering and Agile approaches.

RE@Agile Primer Certificate (Recommended to have CPRE FL and knowledge in Agile)

The RE@Agile Primer certificate is for professionals from related disciplines: Project Managers, Business Analysts, Architects, Developers, Testers, and also business people. This certificate focuses on communication between Requirements Engineering and Agile experts as well as on understanding terms from both areas. Certificate holders can talk to Agile experts about Requirements Engineering and to Requirements Engineering specialists about Agile approaches and Agile development.

A person with the RE@Agile Primer Certificate:

- is familiar with the relevant terminology of Requirements Engineering and Agile approaches
- understands the role and importance of Requirements Engineering in Agile processes as well as the value of Agility in Requirements Engineering

RE@Agile Advanced Level Certificate (Prerequisite is the CPRE FL certificate. However, the Primer Certificate or another recognized Agile certificate, is also highly recommended. *[Note: Recognized Agile Certificates include those provided by the Scrum Alliance and the Scrum Org.]*)

The RE@Agile Advanced Level Module is for requirements engineers and Agile professionals. The AL Module RE@Agile focuses on understanding and applying methods 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 can work in Agile environments, whilst Agile professionals can make the Requirements Engineering-toolbox accessible within Agile projects.

An RE@Agile Advanced Level Module Certificate holder:

- is familiar with the additional terminology of Requirements Engineering in an Agile context as covered by the AL Module RE@Agile;
- can successfully plan, implement and perform Requirements Engineering techniques and methods in Agile projects;
- can successfully plan, implement and perform techniques and methods from Agile approaches in Requirements Engineering processes.

EU 1 MOTIVATION AND MINDSETS (L1)

Duration: 1 ½ hours
Terms: Values, Agile Manifesto, Practices, Activities, Sprint, Agile
Educational Objectives:
EO 1.1.1 Knowing the motivation to use Agile methods
EO 1.2.1 Knowing the RE Values from IREB
EO 1.2.2 Knowing the Core Values of the Agile Manifesto and the Principles derived from it
EO 1.3.1 Knowing the difference between principle, practice, and activity
EO 1.3.2 Knowing the differences between Agile and RE mindsets
EO 1.3.3 Knowing the synergy of the mindsets and values towards RE@Agile
EO 1.3.4 Knowing what "Documentation" means in an Agile Context (in alignment with the Agile Manifesto)
EO 1.4.1 Knowing the benefits, pitfalls, and misconceptions for the use of RE@Agile
EO 1.4.2 Knowing examples for misconceptions
EO 1.5.1 Knowing that Agile values can be transferred to conceptual work
EO 1.5.2 Knowing exemplary approaches that allow for Agility in conceptual work

EU 1.1 Motivation to use Agile (L1)

Several studies (see [MeMi2015]) show that the information technology business as a whole is undergoing an essential change: information technology is becoming a major driver in several business areas (e.g. electric commerce, social media) and technical domains (e.g. automotive or avionics industry). Consequently, the systems and products in IT-driven businesses have to undergo a constant adaptation to keep up with the changing needs of customers or the market. As soon as a change in the market occurs, the systems have to be adapted according to the changes.

Existing development methods that focus on long-term predictability and stability have not been developed for such circumstances and often fail in fast-changing businesses or project situations. Agile methods, driven by the Agile manifesto (see EU 1.2), have emerged to fill this gap. Agile (or Agility) itself is a difficult term and can be defined as follows (see [ShYo2006]):

A rapid whole-body movement with change of velocity or direction in response to a stimulus without loosing control.

This definition originates not from software engineering but from sports, yet reflects the essential motivation for using an Agile method: If the market or project situation requires rapid and controlled changes, Agile methods are suitable. An Agile method is, of course, more than just fast development (see EU 1.2), but in essence, all principles in the end focus on frequent delivery to a given quality.

It is important to recognize that neither Agile methods nor Agility are ends in themselves [Meyer2014]. An organization has to be able to select the proper development approach that fits the needs of their market, customers and organization. Gartner even states that the ability to develop IT with the right approach is the major success factor for digital business [MeMi2015].

EU 1.2 Mindsets and Values in RE and Agile (L1)

The mindset and values of RE are stated in the IREB definition of Requirements Engineering (see [IREB2015]):

A systematic and disciplined approach to the specification and management of requirements with the following goals:

1. Knowing the relevant requirements, achieving consensus among the stakeholders about these requirements, documenting them according to given standards, and managing them systematically

2. Understanding, and documenting the stakeholder's desires and needs

3. Specifying and managing requirements, to minimize the risk of delivering a system that does not meet the stakeholders' desires and needs

In RE, we talk about a system instead of a software or a product. The usage of the term system is not meant to exclude products, other types of software, or even other things (e.g. business processes or hardware). RE prefers the term system because the term emphasizes the fact that a system is a group of parts or elements that works together in an environment. RE calls the environment the system context. In the syllabus and study guide, we will always use the term system and it shall include products and any other type of software-related elements.

The IREB FL [IREB2015] furthermore defines a set of four main activities of RE: elicitation, documentation, validation/negotiation and management of requirements. This list of activities does not denote a specific set of steps or the sequence in which these activities are performed. A core value of the IREB FL is that RE is a process-agnostic approach: RE provides a rich body of knowledge consisting of various methods and a rich collection of techniques that can be applied in any development approach. It does not recommend or specify any one process.

This syllabus and study guide will use the term "Agile methods" to refer to the rich set of approaches that have emerged in the field of Agile (see EU 2). To distinguish Agile methods from other development methods (e.g. plan-driven or waterfall-style), this syllabus and study guide uses the term "non-Agile methods". These two terms warrant an evaluation of which is best - IREB is convinced that both types of methods (Agile and non-Agile) have their value.

The mindset of Agile is defined by the Agile manifesto and the twelve principles behind it (see [AgileMan2001]):

Agile Manifesto

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

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

That is, while there is value in the items on
the right, we value the items on the left more.

Agile Principles

1) Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2) Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

3) Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

4) Business people and developers must work together daily throughout the project.

5) Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6) The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

7) Working software is the primary measure of progress.

8) Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9) Continuous attention to technical excellence and good design enhances Agility.

10) Simplicity--the art of maximizing the amount of work not done--is essential.

11) The best architectures, requirements, and designs emerge from self-organizing teams.

12) At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

If we compare the values and mindsets of RE and Agile we cannot find any piece that would contradict with the other's values or mindsets. The most important value is shared by RE and Agile and that is to make the end user of the product happy because the solution fits their needs or it cures their greatest pains. Nevertheless, we have to recognize that the mindsets and values of RE and Agile are to some extent disjoint. The intersection of RE and Agile is defined by the field of RE@Agile, which will be further explained in the next EU.

EU 1.3 Bridging RE and Agile Principles towards RE@Agile (L1)

Before we dive into the details of RE@Agile, we have to clarify some terminology. Within the software industry and research, there is an extensive body of knowledge on how to work and behave when developing software. This knowledge exists on various levels of abstraction. In the following, we will introduce the differentiation of principles, practices and activities as three abstraction levels to talk about developing software (see [Meyer2014]):

- A principle is a prescriptive statement that is abstract and falsifiable
- A practice/technique is an instantiation of a principle for a certain context
- An activity is a real or planned execution of a practice

The important terms to differentiate the three definitions are prescriptive, abstract and falsifiable. Prescriptive means that the statement directs action instead of stating a fact or a property. Abstractness distinguishes a principle from a practice. For example, "test a software feature before delivery" is prescriptive and abstract, whereas "create a unit test for every software feature" is a practice based on the given principle. An activity for this example would be the creation of a unit test for the search function of a library system. Falsifiability means that a person with sufficient background knowledge can disagree with a principle. The above stated principle ("test a ...") satisfies this criterion. One might argue that testing may not be sufficient for safety-critical features, and instead, they should be verified by mathematical means. A statement that is not falsifiable ("strive for high quality") should not be considered a principle for guiding behavior.

Knowing the principles behind our practices (which itself is a principle) leads to conscious decisions on our actions. Knowing different practices to fulfill principles gives us the ability to react differently depending on the given situation. Falsifiability fosters discussions about the applicability of a principle or practice in a certain context and therefore helps to decide if a principle should be applied or not.

Knowing principles and practices is only a first step: the proper application of a practice is a competence on its own. For example, the definition of use cases is a widely-known practice for the principle "specify functional requirements for important features". However, writing a high quality use case is a competence in its own right and knowing the elements of a use case template is not sufficient for this.

In essence, we have now defined three levels of competences:

1. Knowing principles (corresponds to L1 EOs)
2. Knowing practices to fulfill the given principles (corresponds to L1 EOs)
3. Mastering a practice in a certain context (ability to perform an activity with high quality) (corresponds to L2 EOs)

Comparing the Agile discipline with the RE discipline (see EU 1.2), it is possible to see why the two disciplines are at times perceived to be in conflict: RE is concerned with the systematic elicitation and documentation of requirements as artifacts in their own right, while Agile stresses the importance of the working software over comprehensive documentation and values individuals and communication higher than processes and tools.

An exaggerated implementation of both mindsets can, in practice, lead to a conflict: a false interpretation of RE is that it is possible to create a complete, consistent and agreed requirements document that can be implemented without further modification. A similarly false interpretation of Agile is that a development project can start without any preparatory work and can succeed only by delivering software, in regular intervals, that is reviewed by stakeholders *[Note: Customers are a subset of stakeholders from an RE point of view.]* and improved based on the feedback.

It is our assertion that the RE and the Agile mindsets are not in fact in conflict: both approaches share the same goal of the delivery of software at a well-defined quality level. Agile methods can deliver working software in an efficient and fast way (reduced cycle time). RE provides the proper techniques to understand the stakeholders' desires and needs to develop the right software.

RE helps to facilitate the:

- Relevant understanding of the users' desires and needs to develop valuable software (1st Agile principle)
- Proper tools to recognize changes in the market for the stakeholders' competitive advantage (2nd Agile principle)
- Proper tools and techniques to foster efficient collaboration between stakeholders and developers (4th Agile principle)
- Proper tools and techniques to support verbal communication (6th Agile principle)
- Relevant understanding of the stakeholders' desires and needs to minimize the development of unnecessary software (10th Agile principle).

The important difference between the application of RE in Agile and other development methods is the timing and the process applied. With this syllabus and study guide, IREB defines the field RE@Agile which shows how to apply RE in the context of Agile methods. IREB prefers the term RE@Agile over the term "Agile Requirements Engineering " to make clear that RE is process independent.

The manifesto for Agile software development emphasizes the value of an increment of working software (or working product) over comprehensive documentation. An exaggerated interpretation has led to the false impression that Agile methods have abandoned documentation altogether. This interpretation is wrong: documentation that has a purpose is still welcome and recommended in Agile but only that which supports development or is part of the product. A perceived issue in the past was that many projects created documentation without a clearly stated purpose or added value – that kind of documentation shall be avoided, according to the principles.

EU 1.4 Benefits, Misconceptions and Pitfalls for the Use of RE@Agile (L1)

RE@Agile will offer several benefits. However, these benefits do not come free: there are misconceptions and pitfalls that should be avoided.

EU 1.4.1 Benefits of RE@AGILE

***RE & Dev competencies in the same team can reduce handovers**:* The practice of cross-functional teams in Agile methods requires that the team has all the skills that are needed for the development of a product increment based on the selected requirements. Performing RE tasks within the team can reduce the need for creating comprehensive documentation of requirements upfront since the team's members can explain certain details directly to other members. The benefit of documents in this context will be the documentation of results from the discussions and the preservation of knowledge.

***Incremental development allows for optimization of existing ideas**:* The core principle of Agile methods is the incremental development of software in iterations. The iterative process creates artifacts (e.g. a business process model, an epic, a user story, a use case, a user interface prototype, a process description, or the software), and improves these in a sequence of development and review activities. The benefit of such a procedure is that the quality of the artifact and/or the software is continuously improved and optimized. Furthermore, smaller increments allow earlier discussions with the customer and minimize the risk of large gaps between the customers' expectations and the development.

***Refinement is a principle to mature and validate requirements**:* In Agile development, a very good practice has been developed – continuous refinement. Here regular refinement meetings are held for the development team to review and detail requirements on an ongoing basis, all in close communication with stakeholders. Additionally, the good practice of the Definition of Ready is used as a quality gate to validate that a requirement is ready for implementation in a subsequent iteration.

***RE helps to define an initial product backlog**:* RE provides several techniques to gain a proper understanding of the stakeholders' requirements for the desired product. RE thereby provides a deeper understanding of the requirements needed for the initial definition of the product backlog. It is important to recognize that such an RE activity does not create a detailed specification. Instead the goal of such activities is to focus on a complete understanding of the product at a certain level of abstraction (e.g. understanding and defining the essential use cases or epics and user stories). For example, a complicated high level business process can be decomposed to epics and user stories using well established RE methods to create an initial backlog.

EU 1.4.2 Misconceptions of RE@Agile

In the world of development (mostly software development), some misconceptions and pitfalls regarding RE exist that need to be discussed:

Misconception - RE is only upfront analysis: Very often, RE is considered to be possible only as an upfront activity. RE, as a discipline, is in fact process independent and does not enforce complete upfront work. Instead, the activities of RE can be formed within an Agile method in the same way as other activities (e.g. coding or testing) are performed. RE is an embedded activity within each iteration.

Misconception - Upfront is evil: Preparing for iterative development is an essential part of any non-trivial development undertaking. Upfront thinking does not itself imply any particular software lifecycle or lengthy analysis phase or document: precisely how and when upfront thinking is performed will be determined by the project context. Practitioners should not interpret the Agile literature to mean that upfront thinking is in itself a bad thing; the manifesto and the principles presented above do not support such an understanding. Agile practices that reflect the value of upfront thinking are story mapping (see [Patt2014]), prototyping (see [Martin1991]) and test-driven development (TDD) (see [Beck2003]).

Misconception - RE equals documentation: RE is often associated only with the documents it produces. However, the documents are one potential result of an activity that creates knowledge. Good Requirements Engineering includes being aware of the fact that even the best document is never fully self-contained. Instead, a document serves to support the purposes as defined in EU 3.2: legal compliance, preservation of valuable information, facilitation of communication, and support of thought processes.

Misconception - User stories are enough: User stories are one popular method for capturing stakeholder needs. However, they are intended to start communication and not to represent the complete specification. The path from an unspecific need towards a full requirement is summarised in the practice "3C – Card, Conversation, Confirmation" . A much fuller picture of requirements can be achieved through a combination of user stories with other, approved RE techniques such as context diagrams, prototyping, use cases and user journeys.

Misconception - Documentation is worthless, only code has a lasting value: While it is quite possible that in some particularly process-heavy projects, over-specification might be an issue, it is not right to conclude that all documentation is worthless. Requirements documentation, just like design documentation, test documentation or operational documents are, in certain contexts, all equally valid and necessary artifacts resulting from the development process for any sustainable software product.

Misconception - Working software is the only way to validate requirements: The Agile manifesto values "working software over comprehensive documentation". When it comes to the validation of requirements, a wrong conclusion from this statement is that validating requirements based on working software is always preferable to validation of a documented form of requirements. Software as a means to validate requirements is preferable if the costs and risks connected to such a validation approach are acceptable compared to the outcome. If costs and/or risks are high, RE provides several tools that allow fast feedback and validation of requirements before a single line of code is written, for example: user interface mock ups or story boards (see EU 3.1.10).

EU 1.4.3 Pitfalls of RE@Agile

Pitfall - Treat requirements as a uniform type of information: A typical mistake related to RE in all implementation contexts/methods is to consider the 'requirement' as uniform type of information. Based on this misunderstanding, the documentation of requirements is typically considered a waste of time since the requirements will change so fast that they are invalid as soon as they have been written down. Requirements are NOT a uniform type of information; requirements can be stated in various levels of detail, abstraction and formats. For example, the system vision or goals for a system are requirements at a high level of abstraction with typically a long lifetime; an executable prototype is a means to validate a set of requirements or to elicit new requirements.

Pitfall - Losing the big picture: Agile methods are often misunderstood and implemented in a way that focuses on only the topics that are immediately in front of the team. From a developer's perspective, this might be treated a useful principle because the developer can focus his mental energy on the work at hand and is not distracted by long-term topics. However, if everybody only focuses on the work at hand, the big picture and long-term perspective get lost. Sustainable Agile methods address long-term and mid-term perspectives within dedicated sessions (e.g. refinement sessions, road mapping sessions, or visioning workshops). A related pitfall is proceeding with the solution without thoroughly defining the business problem (often called the need).

Pitfall - Overloading stakeholders with information: RE artifacts can have a high density of information and be created very fast in an Agile team. This approach is often used in "cutting edge" or high technology projects where subject matter experts are hard to come by. However, taking the complete iteration to work on RE artifacts places a high review load on stakeholders, requiring them to digest lenghty specifications. Better results may be achieved with a good mix of specification and development (prototyping).

Pitfall - Elaborating every topic in an incremental and iterative way: Not every requirement topic for a system should be developed in a fine-grained and incremental way. Topics with additive complexity (see [Meyer2014]) are suitable for incremental development. Typical examples of topics are processes that can be separated into independent elements (e.g. buying process in an online store). Topics with fully complete complexity (see [Meyer2014]) are not suitable for incremental development because every new insight on a topic will lead to a completely new understanding of the already known information. Examples here are calculations with complex input parameters and simple output parameters (such as insurance policies or engine control components).

Pitfall - Incremental development may not encourage radical or disruptive innovation: The incremental process of Agile may not encourage the development of innovative and/or disruptive ideas since a given artifact (e.g. the software or a feature/function of the software) is typically improved locally (e.g. fixing errors or adding missing elements) once it has been defined. Although the Agile Manifesto explicitly welcomes change, the incremental processes of Agile typically supports continuous innovation for products and services. Radical or disruptive innovations emerge through the consideration of multiple ideas and the recombination of existing ideas [LiOg2011]. Developing alternative ideas in terms of software is typically considered as waste in Agile environments (see 10th principle – maximize work not done). For radical or disruptive innovations, additional practices need to be incorporated such as lean startup ideas or design thinking approaches presented in EU 1.5.

Major and most important pitfall - Agile and culture change do not go together: Agile values promote changes in the way organizations are working, a loss of ownership on some deliverables and collective responsibility. Agile further promotes continuous retrospectives on the behavior of the team to improve its way of working: continuous change of the team and eventually of the whole organization is inevitable. Such cultural (organizational) changes require both time and qualified people to lead the teams in a new direction.

EU 1.5 RE@Agile and Conceptual Work (L1)

Agile development came into being in the world of software engineering to address challenges that came from the world outside of software engineering (see EU 1.1). Nevertheless, these challenges were not exclusively experienced by the software engineering world. Other branches of industry and society were suffering from similar challenges such as demanding customers and faster innovation cycles. Other fields developed approaches for conceptual work (i.e. creating concepts or specifications of systems) that were quite similar to Agile development. Several of them are especially useful from an RE perspective for development of innovations and product visions. They will be introduced here briefly including a discussion of their correspondence with Agile mindset (see EU 1.2). In the following, we will present three approaches as examples for Agility in conceptual work.

Design Thinking (see [Dsch2015], [LiOg2011]) is a method for solving so-called wicked (i.e., weakly defined) problems. From an RE perspective, design thinking is a combination of elicitation and validation techniques. At the center of this method are (a) a multidisciplinary team that works on the problem and represents a broad range of knowledge necessary to solve the problem; (b) a working environment in which the team can work on the ideas; and (c) an iterative process that consists of the following phases:

- Empathize: in this phase, the team develops empathy to understand the people behind the problem that has to be solved.
- Define: in this phase, the team rephrases the problem to get a shared understanding about the details of the problem that has to be solved.
- Ideate: in this phase, the team focuses on idea generation. The goal here is not to develop the idea. Instead the team develops as many ideas as possible. At the end of this phase, the team selects the most promising ideas for prototyping.
- Prototype: in this phase, the team creates very simple prototypes (not necessarily software!) from the developed ideas. The principle here is that the prototype shall be as realistic and as cheap as possible.
- Test: in the phase, the team tests the prototype with real customers to get feedback on their ideas. One main principle for the test phase is “show not tell”, i.e. that the prototype should be able to speak for itself so that the user can provide genuine feedback.

The phases of design thinking are scalable and can be performed in projects ranging from a few days to several weeks. Furthermore, the phases do not define a strict sequence. Whenever it is necessary, the team can decide to go back or jump forward in the process. With this in mind, design thinking is in line with the Agile value "responding to change over following a plan". The final result of a design thinking process is a set of prototypes that represent validated and innovative solutions to the problem defined at the beginning. Hence, design thinking can be used to develop ideas for high value software and thereby supports the 1st Agile principle. As stated above, the multidisciplinary team and working environment are core elements of the process which is in line with the 5th Agile principle. A major goal of the prototype phase is to produce cheap and light-weight prototypes, which is in line with the Agile principle of simplicity.

Design Sprint [KnZK2016] is a five-day process for developing ideas based on the principles of design, prototyping and testing with end customers. From an RE perspective, the design sprint is also a combination of elicitation and validation techniques. The center of this method is the time-boxed way of working; every day is dedicated to one of the following activities: unpack the team's knowledge, sketch ideas, decide which ideas to prototype, prototype the selected ideas, and finally test the ideas with real customers. The important difference as compared to Agile development is that the prototype does not need to be software. It is important to recognize that the term "sprint" does not refer to the Scrum sprint.

Lean Startup [Ries2011] is an approach for developing businesses and managing startups that is very well accepted in the Agile community. From an RE perspective, it also contains several ideas that are very interesting. Two example are the special product development approach and the minimum viable product. The product development approach is called build-measure-learn and especially focuses on continuous learning about the customer's needs. The minimum viable product (MVP) is "a version of a new product which allows a team to collect the maximum amount of validated learning about customers with the least effort" [Ries2011]. Another important concept of the Lean Startup is the pivot, "a structured course correction designed to test a new fundamental hypothesis about the product, strategy, and engine of growth" [Ries2011]. From an RE perspective, these ideas are a combination of elicitation and validation techniques. Instead of eliciting and validating requirements based on concepts or documents, the elicitation and validation is performed with the real product which is preferable according to Ries under circumstances of extreme uncertainty. Ries emphasizes that the MVP must not necessarily be fully functional and complete software. Instead the book mentioned a very simple webpage for selling shoes with a manual shipping process to validate the need for online shoe shopping.

These approaches show that Agile is really more than Scrum. The examples of design thinking, the design sprint and Lean Startup show that there are approaches for conceptual work in RE@Agile that share the mindset of Agile methods and are therefore fully compatible with organizations that want to develop software in an Agile way. These and other approaches should not be disregarded as a new form of the waterfall approach or upfront thinking. They can and should be used within the framework of Agile development (e.g. within one or more iterations) to design dedicated aspects of a system. Alternatively, they can be used as activities that precede Agile development (e.g. a fast phase of upfront thinking). Thereby, these approaches help to overcome the limited potential for innovation in Agile development (see EU 1.4.2).

EU 2 FUNDAMENTALS OF RE@AGILE (L1)

Duration: 1 ½ hours
Terms: Product Owner, Product Backlog, Sprint Backlog, Epics, User Stories, Story Maps

Educational Objectives:

EO 2.1.1 Knowing examples of Agile methods
EO 2.2.1 Knowing Scrum as an example: its roles, processes, artifacts and their relevance to Requirements Engineering
EO 2.2.2 Knowing the responsibilities of a Scrum Product Owner
EO 2.2.3 Knowing the concept of the product backlog with epics and user stories
EO 2.3.1 Knowing the difference between a classical requirements engineer and a Scrum Product Owner
EO 2.4.1 Knowing good reasons why Requirements Engineering should be part of a continuous process
EO 2.5.1 Knowing value-driven development (i.e. prioritization of requirements)
EO 2.5.2 Knowing that value is risk AND opportunity management
EO 2.5.3 Knowing examples of value in profit and non-profit organizations
EO 2.6.1 Knowing how to simplify the Product Definition Process and how to define a minimum viable product
EO 2.7.1 Knowing the value of continuous processes and their learning curve

EU 2.1 Agile Methods (An overview) (L1)

Many methods have been developed sharing the values of the Agile manifesto. This educational unit will give you an overview of some of them in order to see the diversity of approaches. The list is not meant to be exhaustive and will discuss the methods mainly from a Requirements Engineering point of view.

Crystal is a family of methods developed by Alistair Cockburn [Cock1998]. He suggests that each project needs its own tailored process model. Depending on size, complexity and criticality Alistair suggests adequate roles, activities and artifacts. Crystal Clear – the method for small and less critical projects – is very similar to XP. Crystal Orange and Crystal Red add a bit more formalism to cope with larger projects. From a requirements point of view Alistair suggests (among other things) to work with low-ceremony use case models and mock-ups.

Lean Development [Popp2003] **and Kanban** are based on principles first used in automotive production in the 1940s. They have been adapted for IT-projects in the context of Agile methods [Ande2012]. They strive to discover 7 kinds of waste in the production process (unfinished intermediate products, over-production, defects, …) and gradually eliminate each of them to speed up final delivery.

Scrum [Scrum2016] is a framework for developing and sustaining complex products. The framework concentrates on iterative, incremental development. It only identifies three key roles: a Product Owner (to manage the product backlog, i.e. define the vision and all relevant requirements of a product), a development team to implement these requirements in short sprints, and a Scrum Master to monitor the process and mediate amongst the other roles. We will discuss Scrum in more detail in the next section.

TDD (Test driven development) [Beck2003] is based on the idea of writing the test first before coding the corresponding feature. The test cases are both an exact and detailed specification of the requirements that the product has to fulfill.

XP (eXtreme programming) [Beck2004]**:** XP emphasizes direct communication between a customer and a programmer ("the on-site customer" sitting right next to the programmer, constantly discussing requirements and getting immediate feedback in the form of implemented features).

EU 2.2 Scrum (plus good practices) as an Example (L1)

Scrum is the most popular and most adopted Agile framework. Scrum is a lightweight framework to develop products in complex environments. The Scrum Guide [Scrum2016] provides a definition of Scrum and its essential components.

Scrum proposes 3 roles (Scrum Master, Product Owner, Development Team), four events (Sprint Planning, Daily Scrum Meeting, Sprint Review and Sprint Retrospective) and artifacts (Product Backlog, Sprint Backlog, Sprint Goal and Definition of Done). Scrum does not recommend any engineering practices.

Scrum suggests developing products iteratively and incrementally in a series of Sprints, a timebox of one month or less. Every Sprint results in a Product Increment: a partial product that could potentially be used by end users in a production environment.

The Sprint

The Sprint is the essential driver for the development as it is an iterative process of plan-do-check-act which allows short feedback cyles. Each Sprint begins with Sprint Planning, an event where the Scrum Team collaborates to define what can be delivered in the next Product Increment and how to achieve it (decomposition). The origin of this plan is pulled from the Product Backlog (an ordered and dynamic list of everything that might be needed in the product). The outcomes of Sprint Planning are: the Sprint Goal and the Sprint Backlog (selection of items and their decomposition for the Sprint).

The Scrum Team starts implementing the Product Increment. The Development Team is responsible for transforming selected items into tangible results, working with the Product Owner to refine Product Backlog items and sharing feedback on the current implementation. The Development Team synchronizes every day during the Daily Scrum Meeting to assess progress and update the Sprint Backlog.

The Development Team work is guided by the "Definition of Done": a definition of what is to be achieved for an increment to be complete. The "Definition of Done" helps stakeholders to unambiguously understand the product progress.

When the Sprint timebox is over (4 weeks or less), the Product Increment is inspected in the Sprint Review by the Scrum Team and the key stakeholders to assess the outcomes of the sprint and to discuss what could be done next. The Product Backlog is updated accordingly.

The Sprint ends with the Sprint Retrospective during which the Scrum Team inspects how the last Sprint went and how to improve efficiency. Immediately after the Sprint Retrospective ends, the next Sprint starts.

Good practices

Even though Scrum does not come with requirement engineering techniques, the Agile community developed some good practices which fit well to Scrum.

The Scrum Guide uses the term Backlog Items for the items listed in the Product Backlog. This is a generic term to identify any kind of information about the product to be developed, but many Product Backlog Items are indeed requirements or can be refined to become requirements.

The Agile community proposes:

- Decomposing requirements into: Epics, optionally Features, and User Stories (level of granularity)
- Distinguishing between Functional Requirements (ability), Quality Requirements (behavior), and Constraints (environment)
- Grouping requirements by Themes

A good (additional) practice was developed to describe characteristics of the product backlog: the backlog should be "DEEP" (Detailed appropriately, Estimated, Emergent ,Prioritized) [Cohn2004]. Within the Sprint there is backlog grooming and refinement, which is an essential part of the Agile development process.

With the "Definition of Done" (DoD) the team develops a common understanding of when a Product Backlog Item is complete and is ready to be released into production [Scrum2016].

Another good practice is to apply the INVEST rule [Wake2003]. The acronym covers the following criteria:

I: Independent of each other
N:Negotiable
V: Valuable
E: Estimable
S: Small enough to fit into one sprint
T: Testable

RE offers corresponding criteria for good requirements (see [IREB2015]):

- agreed
- unambiguous
- necessary
- consistent
- verifiable
- feasible
- traceable
- complete
- understandable

EU 2.3 Differences and Commonalities between Requirements Engineers and Product Owners (L1)

After discussing the principles of the Scrum framework, let us compare the role of the traditional requirements engineer with the role of the Product Owner.

The core activities of a requirements engineer are [IREB2015]:

- Requirements elicitation
- Requirements documentation
- Requirements validation
- Requirements management

As explained above the key responsibilities of a Product Owner are:

- Ensuring that the development team delivers constant business value: this means the Product Owner has to balance the longer-term vision for the product with short term needs, has to prioritize the product backlog from a business point of view and has to inspect the results of the development team together with the stakeholders at the end of each sprint.
- Manage all stakeholders: the Product Owner is accountable for forwarding consistent requirements to the team. He has to collect requirements from all stakeholders and make sure they do not contradict each other. Any dispute between stakeholders has to be settled in order to free the development team of such disputes.
- Continuously supply the development team with the highest ranked item from the backlog: The granularity of these requirements must be small enough to fit into one sprint. For any questions that arise after the sprint planning meeting, the Product Owner has to be available to quickly clarify.

Comparing these two roles, it turns out that both, requirements engineers and Product Owners (together with all other stakeholders), have to perform the key tasks of eliciting, documenting, validating, and managing requirements. The notations and tools used, however, are usually less formal in Agile environments:

- story cards instead of requirements documents
- more conversation and less writing
- more emphasis on current state of requirements, less emphasis on versioning and history

While continuing to retain an overall responsibility for high quality requirements, the role of Product Owner is broader than that of a traditional requirements engineer in that he/she is accountable for the success of the product as a whole, continuously gathering feedback from the business and updating and prioritizing the backlog accordingly.

EU 2.4 Requirements Engineering as Continuous Process (L1)

In Agile, then, Requirements Engineering is less a distinct phase during development and more an iterative and ongoing activity. It is not a goal to have all requirements ellicited and analyzed before design and implementation can start; requirements, and products, are created both iteratively and incrementally.

Requirements Engineering is therefore an ongoing activity lasting as long as the product itself. Nevertheless, this process has well defined intermediate results: the forecasted requirements that promise the largest business value should be “ready for implementation” (in the sense of the “definition of ready” described above). Other requirements that are less urgent from a business point of view will only be refined once the urgent ones are complete.

"Continuous process" does not exclude some important upfront activities. Even if requirements are clarified on a "need to know" basis there are some aspects of requirements that should nevertheless be addressed early in the lifecycle. Examples are defining visions or goals, knowing the stakeholders and establishing the product scope. Starting implementation without any such activities considerably raises the level of risk.

EU 2.5 Value-driven development (L1)

Agile Methods strive to continuously deliver business value to the end user. Often business value can be directly expressed in financial terms, in increased market share or in terms of customer satisfaction. This approach is often used in profit-driven organizations, but it is less clear how to define the value for non-profit organizations. Here, measures like usage-rate or happiness index in relation to a product (i.e. click-rates of websites or donations of a non-profit organization) may be more relevant.

A different kind of value is risk reduction. Good Agile approaches try to balance business value and risk reduction over the iterations.

In order to determine which requirements lead to the optimal value, Agile methods often strive for minimum viable products (MVPs) or minimum marketable products (MMPs) as explained in the next section.

EU 2.6 Simplicity as Essential Concept (L1)

In a complex world, simplicity is a way to embrace complexity by the process of:

- Creating a simple and potentially incomplete response to a problem, thus creating a small increment of value,
- Gaining the ability to learn more about the context, based on real world experience,
- Adapting and iterating on value creation and delivery at a sustainable pace,
- Saving resources from a non-profitable idea to reallocate them on a new idea by failing fast and learning quickly.

Simplicity is in some ways opposed to 'perfection'. This being said, simplicity most of the time does not mean 'poor quality' but rather 'minimal scope' or 'minimal service' – but always with high quality. Quality is not negotiable.

Two kinds of "simpler" products: MVPs and MMPs

A Minimum Viable Product (MVP) is a concept from lean startup (see [Ries2011]) and is defined as the smallest product that can create an end user experience and provide feedback to the team. This feedback is a major input to evolve the product. Many Start-ups are aligned on this way of working since it enables creating a rapid return on investment with a low level of risk.

The purpose of Minimum Marketable Products (MMP) goes one step further. The issue is not only providing early feedback thus driving the next requirements steps, but immediately creating value. Many products can be used in a simple "version 1" without already having all desired features and qualities thus already creating revenue to pay for continuous improvement of the product.

EU 2.7 Inspect and Adapt (L1)

Many Agile methods (including Scrum) emphasize the importance of frequent and fast feedback. After each iteration (sometimes even more often) the team should discuss whether the development process is working for them or should be improved.

In this feedback process, everyone should inspect the current development process at an early stage. The team should challenge the methods used, the tools, the cooperation in the team, etc. Everyone is asked to answer questions like: What worked well? What did not work well? What should we try in the next iteration?

It is important that the product or solution will be reviewed at the end of each sprint. The team should look at their team velocity and change or adjust the planned speed of implementation for the next iteration.

This also applies to the requirements process. The consequences of these insights should be short-term adaptation of process improvement steps.

EU 3 ARTIFACTS AND TECHNIQUES IN RE@AGILE (L1)

Duration: 1 ½ hours
Terms: Product Vision, Product Roadmap, Product Backlog, Sprint Backlog, User Story, Acceptance Criteria, Feature, Functional Requirement, Quality Requirement, Epic, Context Model, Story Map, Definition of Ready, Definition of Done

Educational Objectives:
EO 3.1.1 Knowing the difference between traditional specification documents and backlogs
EO 3.1.2 Knowing the value of visions and goals
EO 3.1.3 Knowing the added value for the use of context models in RE
EO 3.1.4 Knowing how to distinguish the three kinds of requirements
EO 3.1.5 Knowing the different levels of granularity in requirements
EO 3.1.6 Knowing the different specification formats for the different artifact types in Agile processes (i.e. textual vs. template-based vs. diagrammatic)
EO 3.1.7 Knowing the value of Terms, Glossaries and Information models
EO 3.1.8 Knowing the specification of Quality Requirements and Constraints in Agile Requirements Engineering processes
EO 3.1.9 Acceptance and Fit Criteria
EO 3.1.10 Knowing the use of Definition of Ready and Definition of Done in Agile Requirements Engineering processes
EO 3.1.11 Knowing the difference between Prototype and Increment
EO 3.1.12 Knowing the different artifact types in RE@Agile processes (Context Model, Epic, User Story, Backlog, Roadmap, Requirement, Definition of Done, Definition of Ready)
EO 3.2.1 Knowing how to elicit requirements in Agile Requirements Engineering
EO 3.2.2 Knowing how to create and maintain backlogs in Agile Requirements Engineering processes.
EO 3.2.3 Knowing how to validate and negotiate requirements in Agile Requirements Engineering
EO 3.2.4 Knowing how to manage requirements in RE@AGILE

EU 3.1 Artifacts in RE@AGILE (L1)

EU 3.1.1 Specification Documents vs. Product Backlog

In order to create products or solutions from requirements, the requirements cannot stand alone, but rather need to be organized and documented as an ordered list of everything that might be needed in the product [Scrum2016]. Independent of any methodology, this requirement collection is considered to be the main artifact for requirements engineers, business analysts or Product Owners. Different methods suggest different forms and names for such requirements collections.

Requirements Engineering usually calls this artifact, the result of the elicitation process, a User Requirements Specification, System Requirements Specification or Software Requirements Specification, depending on who writes it and what level of detail it covers. It is not necessarily document-based, but simply a collection of requirements in any physical form (paper, repository based, database …).

Agile methods, especially Scrum, prefer the term Product Backlog for this overall collection of requirements (and other product-related information, see EU 2.1) to be implemented in the future and Sprint Backlog for those requirements that have been selected for the next iteration (sprint in Scrum) [Scrum2016]. Again, the physical form of these backlogs does not matter. They might consist of index cards or sticky notes on a wall or be captured in some appropriate software tool. Although the names and the handling may differ, all artifacts follow the same ideas and offer a basis for elicitation, documentation, negotiation, validation and management of requirements.

In EU 2.2 we learned, that DEEP is a good practice so that the product backlog is detailed appropriately, estimated, emergent and prioritized. These characteristics are important for the product backlog and they are linked to or dependend on each other. For the product owner as the "value optimizer" the ranking or ordering of the product is the most valuable and important one to achieve. The estimation supports this and the appropriate detailling helps to focus on the important parts, so they are supportive to the ordering.Independent of the name for the requirements collection, certain elements should definitely be captured. This includes all kinds of requirements: goals and visions, the definition of the scope of a system or product, functional requirements, quality requirements and constraints, and a glossary (i.e. definitions of relevant terms and abbreviations). As discussed later, approaches may vary in notations, syntax and level of detail for such requirements specifications. Having no specification at all (i.e. only trusting verbal communication between stakeholders without any written requirements) is not typically an alternative since written documents are often the basis for negotiation, acceptance testing, legal purposes and more. The more all stakeholders communicate with each other, the less that writing is necessary, but the results, the requirements, should still be captured in a well known form (written or drawn). In the following paragraphs, the different parts of this overall requirements collection will be discussed in detail.

EU 3.1.2 Vision and Goals

Each development process should be guided by visions or goals that define the product capability which, if implemented, would mean that the solution is considered successful. Having such visions or goals, agreed upon by all relevant stakeholders as early as possible, is of utmost importance to any activity related to requirements of the respective system or product. In Agile development processes the term “product vision” is commonly used to emphasize that each outcome of the development process should have a distinct business value which relates to the product vision.

Goals from different stakeholders can be contradictory, meaning the related stakeholders must negotiate to come up with an agreed vision or set of goals. Alternatively, it could indicate that variants of the product (e.g. a small and a large iPhone) are required, or even different products altogether (an iPhone and an iPad).

Agile development often uses goals with different granularity, such as goals for different time horizons or planning intervals. For example, there might be one year goals to allow negotiations about the deliveries (time and content), three month goals for release planning and sprint goals for the next iteration/sprint [High2009].

Long-term product visions and short-term sprint goals are useful to emphasize the most important achievements that should be reached within a particular timeframe and help to align all stakeholders on a "common mission". All such goals can be usefully represented on a timeline within the roadmap.

The product vision or goals are the most abstract form of requirements and cannot be taken into development without further refinement. They provide easy to understand and comprehensive guidance for the whole Agile development process. Every requirement should be checked against the goals to verify the contribution of the requirement to the different goals. A requirement without a relationship to the set of goals may be an indicator for missing business value. Consequently the product vision statement and the agreed stakeholder goals are very important artifacts for the success of Agile development processes because they set the framework for all development activities without constraining developers' creativity unnecessarily.

EU 3.1.3 Context Model

The vision statement together with the stakeholder goals specify overall demands the system or product should satisfy to fulfill its purpose. Context models on the other hand represent a different viewpoint, as their aim is to describe particular properties of the environment (context) in which the system or product will operate.

The requirements of the system or product are often specified under consideration of assumptions about the environment. Context models are a structured way to document relevant assumptions about the context. It is beneficial to make such assumptions explicit in order to establish a shared and agreed view on the operational environment of the system or product for the whole development team and other relevant stakeholders. In case of uncertainty, the specified requirements will only be correct if the assumptions documented in the context models prove true, meaning the context models represent the actual operational context of the system or product correctly.

Context models are a powerful artifact as they clearly differentiate between the system or product to be developed and its context, consisting of, for instance, adjacent systems and human users (see [IREB2015]).

By using this differentiation, functionality can be allocated to differentiate between responsibilities of the system or product itself and responsibilities of adjacent systems or human users in the context - collaborating during an operation to fulfill the overall vision. Therefore, context models can also be used to clarify and specify the external interfaces of the system or product to be developed.

Such models can be documented using different formats, such as context diagrams proposed for structured system analysis, Use Case diagrams, SysML block definition diagrams, UML component diagrams or UML class diagrams. Any notation is suitable if it clearly differentiates between the system or product to be developed and external interfaces to persons or systems in the environment. It must additionally document the relationships between these elements and the system or product to be developed at an appropriate level of detail. Even simple hand-drawn box and lines diagrams can be pragmatic and useful.

Independently of the form of documentation, a context model is a very valuable artifact and is recommended during an Agile development process. It delimits the area (inside the scope) where analysts are free to make decisions, while the external interfaces (i.e. the boundary between scope and context) have to be negotiated with the adjacent systems.

EU 3.1.4 Requirements

Requirements, then, have to be captured based on the vision and goals and bounded by the context model. IREB defines three different kinds of requirements: functional requirements, quality requirements and constraints that will be discussed in the following sections.

EU 3.1.5 Granularity of Requirements

Stakeholders often communicate their needs in different levels of granularity: from coarse-grained requirements, such as general business goals, to fine-grained requirements specifying details of expected system functionality. Functional and quality requirements (see EU 3.1.8) can (and should!), therefore, be discussed and documented at different levels of abstraction.

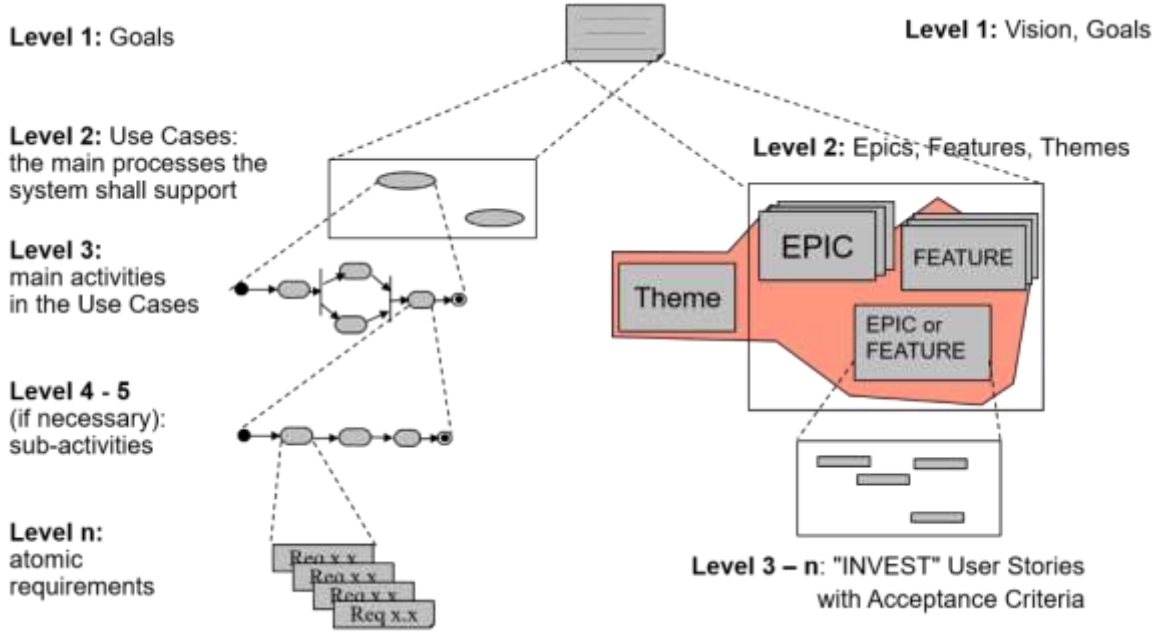

Figure 1: Requirements Granularity: use case decomposition and agile variants, Use Cases, Use case Specifications and Activity Diagrams

One approach typical of non-Agile methodologies is represented on the left-hand side of Figure 1. In this approach, use cases are used to first refine visions and goals and provide a sub-division of expected system functionality (See Figure 1 – Level 2). The picture only shows one example for a high-level grouping of requirements. Other examples include grouping by features, business objects, data flows, or components of existing solutions.

At the next level of granularity, each use case is elaborated to describe the steps involved in fulfilling that function. Several alternatives according to the level of complexity are available here:

1. Verbal description (Plain text)
2. Use Case templates (semi-structured format or narrative flows)
3. UML Activity diagram (or any other kind of graphical functional model like Data Flow Diagrams, BPMN, Context-Diagram, State Chart, Business Object Model, etc.)

Level 3 in Figure 1 shows use cases elaborated with UML activity diagrams, for example.

Based on problem complexity, further levels of granularity can be provided, e.g. decomposing activities into sub-activities (see Level 4 in Figure 1) or with textual requirement specifications of individual activities (see Level 5 in Figure 1).

Thus coarse-grained customer requirements are refined successively into fine-grained specifications of expected solution functionality.

Note that the above approach implies a top-down process of requirement analysis and decomposition. This somewhat idealized view is not always realistic. Requirements may in fact be first expressed at the level of solution detail, with more general goals only later established. The real-world process may therefore be top-down, bottom-up, or some combination of these.

The key point is that methods and techniques exist to support these discussions at their different levels of abstraction and to establish a meaningful hierarchy of requirements as the overall knowledge of requirements increases.

One option may be used to describe use cases with low complexity, using just a few sentences clarifying the required steps to be performed. For medium complexity use cases, other options may be a good choice, as the use case templates allow a description of more sophisticated process steps, which are part of the specific use case. This semi-structured format helps to understand the requirements captured in a use case with a clear guideline. Nevertheless, the use case template has limitations if there are parallel activities and high complexity with multiple scenarios.

Finally, activity diagrams, allows the requirements engineer to use a graphical format where multiple scenarios can be displayed. The diagram can represent multiple parallel and sequential streams of steps. This form of a use case description allows complex use case flows to be described in one diagram.

Agile terminology: Epics, Themes, Features and User-Stories

On the right-hand side of Figure 1, an equivalent approach is shown using Agile terms such as Epics, Themes, Features and User Stories.

Here, high-level business goals and complex functionality are captured as epics or features and grouped by themes. Such topics may be sufficiently complex that they require more than one sprint for a Scrum team to develop [Griff2015].

Such complex topics are then decomposed into finer grained user stories. The criteria for “good” user-stories have been discussed earlier (EU 2.2). They should follow the “definition of ready” [Kron2008] , such as fulfilling the INVEST criteria [Wake2003]. Especially the “E”, “S” and “T” are important: They should be estimable, small enough to fit into one iteration, and testable.

Once again, whether a project follows a strictly top-down process of refining epics into features into user stories, or whether user stories are first gathered individually with common themes and epics emerging later bottom-up, will depend on the nature of the project and the stakeholders involved. Either way, Agile provides artifacts to discuss and prioritize both the big picture and development-ready requirements.

EU 3.1.6 Graphical Models and Textual Descriptions

Independent of the granularity, there is always a choice about the notation for functional requirements. They can be expressed by writing them in semi-structured text form (like user stories according to a template), or as a plain natural or formal language (like Gherkin) that clearly states what the solution should do.

Since it is well known that natural language is sometimes not as precise and unambiguous as needed, many graphical notations have been developed to overcome that ambiguity or show interdependencies, such as between data and process. Examples include UML activity diagrams, BPMN-diagrams, flow charts, state charts, sequence diagrams, etc.

Most of these graphic based notations emphasize different aspects. Activity diagrams or BPMN-diagrams are well suited for rather linear processes showing consecutive steps, alternatives or loops. State charts, on the contrary, are perfect whenever asynchronous events can influence the flow. Sequence diagrams are perfect for "specification by example" since they show concrete scenarios of interactions without trying to be complete.

There are positives and negatives to modeling processes and data or data and interactions or processes and interfaces. They are interrelated (data supports process) so it can never be an either/or situation, suggesting only to use this model or that one. Such an approach would be quite counterproductive. While textual requirements are easier to understand by many stakeholders, they can also be easily misinterpreted or misunderstood by the same readers. Graphical models provide more formality thus avoiding different interpretations or misunderstandings. The choice of style should be determined by the key goals of Requirements Engineering, ensuring a common understanding between all stakeholders on one side and providing enough protection against incompleteness and misunderstandings on the other side [GoAk2003].

EU 3.1.7 Definition of Terms, Glossaries, and Information Models

Functional requirements are incomplete without a clear understanding of all the terms used in such sentences and graphical models.

Therefore, a collection of all relevant business terms used in any requirements artifact is a necessary artifact, but easily overlooked when only concentrating on business requirements.

The minimum form of this artifact is a list of (textually described) business terms and abbreviations, usually alphabetically sorted for easier look-up. This is sometimes called a dictionary, a glossary, or a list of definitions.

When the business is very complicated, this glossary can be structured by grouping simple terms into classes (or entities) and showing the overview of all terms in graphical format. Such artifacts are then called data models, information models, entity-relationship-diagrams, or UML class diagrams. In addition to the definition of terms, these models also include relevant associations between these entities, i.e. static relationships between the terms.

As mentioned above, the product backlog often consists of epics and user stories, emphasizing the functionality needed and deemphasizing definitions of business terms. Nevertheless, even in Agile approaches, a clear understanding of business terms is necessary to create a shared and aligned understanding of the terms used in artifacts like epics and user stories.

EU 3.1.8 Quality Requirements and Constraints

Besides functional requirements (specifying functions a system or product needs to provide), quality requirements and constraints are of crucial importance for the success of the system or product being developed. Traditionally, quality requirements and constraints are included under the umbrella term “non-functional requirements” [CNYM2000].

Quality requirements pertain to specific qualities a system or product needs to exhibit, for example, concerning performance, reliability, safety, security, or usability [ISO25010]. With an emphasis on functional customer requirements in the form of user stories, there is a risk with Agile that quality requirements are not explicitly stated. Quality requirements cannot be defined as user stories that can be developed within one sprint: they rather describe an emerging attribute of the product being developed, and must therefore be tested continuously for all user stories. RE’s checklists of quality aspects (see [IREB2015]), for example, can be useful. Quality requirements are notoriously difficult to build into existing systems by refactoring, thus it is all the more valuable to consider such aspects early in the process.

Constraints define overall restrictions on the solution space of the system or product to be developed [Glinz2014]. Constraints come in a number of forms: organizational constraints (budget limitations, tight schedules, a prescribed development process, etc.), technical constraints (requiring a certain DB-system, the use of a specific programming language, selected frameworks, etc.) or constraints of the environment itself within which the system will operate (standards, norms, regulations, etc.).

Similar to quality requirements, learning too late about key constraints can be very expensive, since many such aspects cannot be added incrementally. Planning and design decisions depend on a good understanding of these issues, and so here again it is crucial that key constraints are identified early in the process.

Similar to functional requirements, quality requirements and constraints may exist at different levels of abstraction and should be documented in the product backlog, made visible to the team (e.g. put on the Scrum board) and tested in every sprint. Therefore, it might be useful to add them to the “Defintion of Done” and implement their validation in the form of automated regression tests.

EU 3.1.9 Acceptance Criteria and Fit Criteria

All kinds of requirements are of limited value if their fulfillment cannot be validated, checked or tested. Therefore, each requirement needs a set of criteria that can be tested in order to check if the requirement has indeed been fulfilled. Such criteria additionally aid understanding (and encourage feedback) by describing in concrete terms what is expected.

The type of criteria used matches the level of granularity of the requirement:

- On higher abstraction levels (Vision, goals, capabilities and epics), Success Criteria [SAFe] are usually defined because it is only possible to measure if a needed functionality/capability is provided or not.
- On lower abstraction levels, Acceptance Criteria can be used to describe how the solution will be tested against the requirement in order to gain acceptance.

Non-Agile and Agile methods agree that requirements have to be verifiable. Non-Agile methods often use terms like “quality criteria” or “fit criteria“ or plain old “test cases”; in Agile the terms “Acceptance Criteria “ (for user stories) or “Success Criteria” (for epics or themes) are more common.

EU 3.1.10 Definitions of Ready and Done

While Acceptance and Fit Criteria belong to and complete the business requirements, the artifacts, “Definition of Ready” (DoR) and “Definition of Done” (DoD), support the formal process of development and ensure the quality of requirements and product increments. The DoD is an official artifact of the Scrum Guide [Scrum2016] and works as a quality gate for the Agile development process, while the DoR is a good practice that should help to create valuable requirements and avoid overloading the team with unqualified requirements. The DoR helps to bring the requirements to the right level of detail and deliver enough information for the negotiation that is intended between the Product Owner/Requirements Engineer and the Development Team.

EU 3.1.11 Prototype vs. Increments

Another way of dealing with requirements are prototypes since many stakeholders that have requirements for a system or product do not want to write or read documents in order to define a product – they want immediate success. For those people, the best way to understand their needs and get feedback is to demonstrate system functionalities or capabilities in the form of a running system.

One way of doing this is through the use of minimal, incremental products. Requirements engineering proposes two types: “horizontal” product increments to show more varieties for validation ("do more of the right things") and "vertical" product increments to verify that the development approach is right ("do the right thing").

By providing a direct interaction with a working system, prototypes can be very useful for gaining feedback. Usability properties, such as reaction time for example, are hard to specify but easy to identify when using working software. Prototypes may also, however, be a source of frustration - for users because they believe the development is already finished and for developers because they are often thrown away to be later replaced with better technology.

Agile methods try to avoid throw-away prototypes by immediately developing good quality increments of the "real" system or product. Agile strives for short iterations delivering demonstrable product increments which are in turn used to learn more about the requirements. Though the intention is not to throw away the developed code, refactoring is an Agile good practice as answer to changing functional or quality requirements based on user feedback.

The term "spike" in Agile is used to refer to a development iteration performed with the explicit purpose of understanding an area of complexity (system architecture, for example) and thus reducing risk. The term, although not defined precisely, can refer to validating one task or a whole iteration. Prototyping is a valid technique within spikes where, unlike other Agile iterations, the primary goal is knowledge gained rather than the working code.

EU 3.1.12 Summary of Artifacts

As mentioned in the last paragraphs, some artifacts are very important for successful development:

- It is a good practice to start with visions or goals.
- It is a good practice to always identify and know the most important stakeholders.
- It is a good practice to explicitly set the scope and delimit it from the context.

Even though non-Agile RE and Agile RE may use different terms, they agree that it is necessary to understand functionality as well as business terms and to capture functional requirements (including functionality and data).

In addition, Agile and non-Agile methods should identify quality requirements and constraints, as these may strongly influence design decisions. Ignoring them or learning about them too late may lead to a lot of rework and a product that does not meet the customers' expectations.

Good Requirements Engineering ensures that no relevant issues are forgotten. Non-Agile methods therefore often insist on documenting many potential areas of interest relating to the requirements for a system.

Agile approaches use less formal artifacts and replace the missing documentation with direct communication and quick feedback loops made possible via incremental product development or prototypes.

The often-quoted second principle of the Agile manifesto [AgileMan2001] explains exactly what we just discussed:

"… Through this work, we have come to value: … Working software over comprehensive documentation …"

We are convinced that a deliberate consideration of what needs to be captured in writing, what can be discussed, and what can be prototyped or shown in increments is very fruitful for any organization. The best results will be achieved when documentation, communication and prototyping/incremental development are balanced according to the constraints and culture of the company. Chapter 4 will cover more on the topic of balancing upfront requirements (and architecture) activities with iterative activities.

EU 3.2 Techniques in RE@AGILE (L1)

In EU 3.1 you have learned about important requirements artifacts. In this educational unit, you will learn about the key activities to be performed in Requirements Engineering. IREB structures these activities the following way:

- Requirements Elicitation
- Requirements Documentation
- Requirements Validation and Negotiation
- Requirements Management

The following subsections discuss these activities from an Agile perspective.

EU 3.2.1 Requirements Elicitation

Requirements Engineering in Agile development is founded upon intensive communication between all stakeholders (including the development team) to elicit requirements. Informal, direct communication among team members can itself be considered a good mixture of interviews and brainstorming. Techniques like XP's "onsite customer" can be equally successful in the form of sessions between the Product Owner and stakeholders (typically the development team refining backlog items). The aim in all cases is to gain greater insight into what is really needed.

Requirements Engineering provides a much broader range of techniques to discover and elicit requirements than are normally discussed in the context of Agile development. These include Q/A techniques (not only interviews, but also questionnaires), observation techniques, artifact-based techniques (reuse, system archeology, etc.) [IREB2015] and creativity techniques such as brainstorming or design thinking. These techniques support the idea of requirements in a User Story format, which are a basis for structured discussion rather than a prescription for implementation.

As mentioned in section EU 3.1.11, prototypes and product increments are another way to learn more about requirements. As soon as a product increment is presented in the sprint review or demo meeting, new ideas may come up which can be directly taken into the product backlog and prioritized by the Product Owner.

Requirements Engineering in Agile development can benefit greatly from non-Agile Requirements Engineering by studying the different elicitation techniques and making a deliberate decision which mix of techniques should be chosen. While intensive communication among stakeholders and quick feedback through product increments are excellent ideas, there is more to elicitation than this. If there are hundreds or thousands of stakeholders, for example, verbal communication alone is not sufficient. When trying to uncover innovative, subconscious requirements, creativity techniques may be required. When working under tight timing constraints (i.e. not enough time for intensive discussions), then techniques like snow cards may be most effective. When exploring new technologies, spikes (timeboxed, simple increments to explore potential solutions) are a great idea.

EU 3.2.2 Requirements Documentation

In Agile requirements are organized within one or more backlogs. The two main types of backlogs are the product backlog and the sprint backlog. Requirements may be documented in the form of story cards annotated with business value and priority. These may be organized within story maps or decomposed into simpler stories. The principle behind the cards is that the size of the card restricts what is written and helps to focus on the core details.

Nevertheless, documenting requirements is still considered an important activity to foster communication between all stakeholders. The formalism for documentation can be minimized if details are communicated verbally.

Defining an adequate degree of documentation depends on many factors like size of the projects, number of stakeholders involved, legal constraints, or safety-criticality of the project. Based on such factors, Agile projects try to avoid documentation overkill and find a minimum set of useful content of the documentation.

While working with a "living" product backlog is an efficient way to handle documentation, it is not always sufficient. So, let's take a look at other kinds of documentation.

From an RE perspective, we distinguish four types of documentation:

1. **Documentation for legal purposes**: Certain domains or project contexts (e.g. software in the health care sector or avionics) require documentation of certain information (e.g. requirements and test cases for a system) for a certain audience to obtain legal approval.

   The principle of creating documentation for legal purposes is: the legally necessary documentation has to be derived from the corresponding laws or standards and is an inseparable part of the product.

2. **Documentation for preservation purposes**: Certain information about a system has a lasting value beyond the initial development effort. Examples include the goals that the system was built to achieve, the central use cases it supports or decisions that were made during its development, for example to exclude certain functionalities. Documentation for preservation purposes can become the shared archive of the team, a product or an organization. It can relieve a dependency on the memory capacity of the individual team members and can ease discussions about previous decisions (E.g. "why did we decide not to implement this?").

   The principle is: the team decides on what to document for preservation purposes.

3. **Documentation for communication purposes**: Effective and efficient communication is an important tool in Agile methods because of its interactivity and short feedback cycles. In practice, there are several situations that may hinder direct verbal communication: distributed teams, language barriers or time restrictions of those involved. Furthermore, information is sometimes so complex that direct communication may be inefficient or misleading. A paper prototype or a diagram of a complicated algorithm can, for example, be reread later. Sometimes stakeholders simply prefer written communication to reading source code or reviewing software. In these cases documentation facilitates the communication process between all involved parties and conserves the results.

   The principle for creating documentation for communication purposes is: a document is created as an additional communication means if stakeholders or the development team notice a value in the existence of the documentation. The document should be archived when the communication has been successful.

4. **Documentation for thinking purposes**: An often forgotten aspect of writing a document is that writing is always a means to improve and support the thought processes of the writer. Even if the document will be thrown away later in the process, the benefit of improving and supporting thinking is lasting. For example, writing a use case forces the writer to think about concrete interactions between the system and the actors including, for example, exceptions and alternative scenarios. Writing a use case can therefore be understood as a tool to test your own knowledge and understanding of a system.

   The principle for creating documentation for thinking purposes is: the thinker decides on the document form that supports his or her thinking the best. The thinker does not need to justify his choice of documentation form for thinking. The document may be discarded when the thinking process is finished.

Agile methods can benefit significantly if these four types of documentation are identified and applied in the proper context. Summarizing we can say that documenting requirements is not an end in itself, but that it should facilitate the communication among stakeholders, especially between requester (often substituted by the Product Owner) and the development team.

EU 3.2.3 Requirements Validation and Negotiation

While Requirements Engineering emphasizes requirements validation via methods like reviews, walkthroughs, inspections or perspective-based reading, Agile methods strive to validate requirements through early and frequent feedback on valuable product increments. One good practice to support this is automated regression testing, which provides continous validation of the development and the related requirements. The aim of requirements validation includes identifying missing, ambiguous or incorrect requirements as well as controversial or conflicting requirements where negotiation and conflict resolution techniques can be applied.

Since iterative, incremental development is a key strategy in Agile methods, the need for formal validation of documents decreases. It is replaced by constant negotiation among all stakeholders about the requirements so that conflicts are discovered and resolved early on. Another way to validate requirements is automated (regression) testing.

Formal validation is also reduced by showing quick results in the form of integrated product increments. If the increment does not meet all requirements of all stakeholders, the delta is put back into the product backlog in form of new requirements and evaluated and prioritized with all other backlog items.

Nevertheless, walkthroughs of the product backlog, discussions of business value, discussions of risks and immediate negotiations of requirements are all valuable techniques in Agile Requirements Engineering. All these techniques may be used within refinement meetings where the Product Owner and the development team (and stakeholder if available) work together to find the level of detail needed for implementation, using the principles of continuous refinement.

In the first versions of the Scrum Guide, the so-called product backlog refinement meeting was only mentioned indirectly. With the new version [Scrum2016] it has been made explicit and is good practice for validating requirements, uncovering problems early and reducing the time needed for the sprint planning meeting later on.

EU 3.2.4 Requirements Management

In traditional RE, requirements management is concerned with all activities to handle requirements over time. This includes version management, change management, configuration management, traceability, as well as adding attributes like status, estimates, priorities, links to conflicting requirements, and the people involved in capturing, checking, signing, implementing or testing the requirement.

As discussed earlier (see EU 3.1.1), the key artifacts for maintaining requirements within Agile are one or more backlogs. Unlike traditional requirements management, backlogs are designed to keep only the latest and best version of all requirements yet to be implemented. Backlog items are typically deleted as soon as the product fulfilling these requirements is delivered.

Requirements management activities that do take place in the backlog include:

1. Requirements prioritization: determine their business value to decide when to implement them. The higher the business value, the more important a requirement will be, since Agile projects try to deliver the highest business value first. Often influencing factors other than busines value exist and will be discussed in the Advanced Level of RE@Agile.

2. Requirements estimation: determine how much work is involved in fulfilling them. Too big estimates are a clear message to a Product Owner that more work has to be done in order to bring them to the definition of ready (DoR, see EU 2).

This is not to say that other activities concerned with the historical aspects of requirements management cannot take place in Agile. Such activities will typically be recorded outside of the backlog, however.

In deciding what requirements management activities are appropriate in a given context, the requirements engineer should seek a balance between minimizing overhead, allowing for early delivery of working solutions, and the longer-term needs of the organization such as legal compliance, operational documentation or handover to new development team members.

Conclusion

Requirements activities like elicitation, documentation, validation and negotiation, as well as requirements management still have to be performed in general in Agile development. Preferred techniques for elicitation and documentation may differ between Agile and non-Agile development, but learning from each other reveals the best solution as it combines the strength but reduces the waste or overhead. This way of working represents the five values of Scrum (commitment, courage, focus, openness and respect) and their representation in the three pillars of Scrum (transparency, inspection, and adaptation) [Scrum2016].

Especially openness and respect are useful when bringing the craftsmanship of RE into the world of Agile and bringing the ideas and principles of Agile to the craftsman of RE.

In this educational unit, you have learned that even in Agile Requirements Engineering, there are more artifacts than just user stories in the product backlog and that the key Requirements Engineering activities should not be forgotten - but may be performed with different emphasis and methods – based on the Agile principles explained in EU 1.

EU 4 ORGANIZATIONAL ASPECTS OF RE@AGILE (L1)

Duration: 1 ½ hours

Educational Objectives:

EO 4.1.1 Knowing the interplay between organizational structure and RE@AGILE
EO 4.2.1 Knowing the interaction with stakeholders in Agile Requirements Engineering processes
EO 4.2.2 Knowing how Communication and Collaboration can help to improve results
EO 4.2.3 Knowing the Role of Management in Agile
EO 4.3.1 Knowing motivation for scaling
EO 4.3.2 Knowing dimensions for organizing teams
EO 4.3.3 Knowing approaches for organizing communication between teams
EO 4.3.4 Knowing example frameworks for scaling
EO 4.3.5 Knowing the main impacts of scaling on RE
EO 4.4.1 Knowing criteria to decide on the level of upfront vs. continuous Requirements Engineering
EO 4.4.2 Knowing the right level of detail for backlog items
EO 4.4.3 Knowing the value of validation in RE@Agile
EO 4.4.4 Knowing the right update cycles for the product backlog
EO 4.4.5 Knowing how to find the right timing of the devlopment cycle

EU 4.1 Influence of Organizations on RE@AGILE (L1)

Agile has its roots in manufacturing and empirical process-control (see [TaNo1986]). Agile principles are easy to understand and Agile practices and frameworks like Scrum are easy to use in a greenfield environment like startups or small companies. It is hard to implement them in larger organizations which behave like a living organism, defending each intruder. But, on the other hand, as organizations discover the benefits of applying such principles and practices, they try to mix them with their own DNA like described in Conway's law [Conw1968].[Conw1968] Doing so results in a growing interest in topics like Agile management (see [ACP/PMI]) and Agile organizations (see [Denn2015], [Appel2011]), which leads to discussions of Agile that frequently extend beyond (software) development.

The ideas of putting the customer at the center, self-organizing teams, enabling and empowering individuals and continuous improvement all have resonance in the wider world of business. Nevertheless, the focus of RE@AGILE, in line with the mainstream disciplines of RE and Agile methods, lies firmly with development and hence we look primarily at the development when considering the impact of Agile on the organization (demand and supply).

In the world of software development, many Scrum implementations fail in development because the rest of the organization was not itself able to change to support the Scrum teams.

Why is such organizational change needed?

Let's take two examples of why the rest of the enterprise must also change in order to support Agile development:

1. The demand (sub-) organization must be able to deliver enough good requirements (good means detailed enough but not too detailed – following the INVEST principle) in order to keep development running at a steady pace. This, in combination with frequently changing requirements, requires a continuous demand flow.

2. The HR department needs to understand which people to hire in order to support the Scrum teams correctly. Often job offers for Scrum Master can be found in which programing skills and certifications are required, which shows that the principles of the three Scrum roles are misunderstood.

In EU 4.2 we discuss organizational aspects when embedding an Agile organizational unit, such as a Scrum team, in a non-Agile environment. In many cases, although the introduction of Agility starts in product development, the entire enterprise may not necessarily follow Agile principles.

The influence on the organization in the case that more than a single Agile team is needed to solve a complex problem is discussed in EU 4.3. The main focus is again on the IT organization and its interfaces to the business. The transition of a business to a fully Agile organization is not explicitly considered here as it is beyond the scope of a discussion on RE.

EU 4.4 focuses on organizational aspects in timing, analyzing especially the question of when RE activities should be executed.

EU 4.2 Agile development in a non-Agile environment (L1)

EU 4.2.1 Interaction with stakeholders outside the IT organization

The role of the development organization within the enterprise is to deliver solutions and services to enterprise customers (both inside and outside of the organization).

Agile places the customer at the center of product development. This means that the customer is involved throughout the product development lifecycle, that the feedback on incremental deliveries is actively sought and that new requirements in line with the needs of the business are accommodated. With the ongoing involvement of the customer, RE also becomes a continuous process (see EU 2.4). The person responsible, like the Product Owner in Scrum, should engage their customers in open and direct communication, listening for new needs and changing expectations and capturing these within the backlog.

In practice, this communication may take many forms. If the customer is in fact available to work with the team on a daily basis, then communication can indeed be direct and mostly informal in nature, where just results or decisions are documented. It is important to recognize factors outside the control of the development team (for example, geographical separation or simply a lack of availability), that mean direct interaction is not always practical. Here, a more efficient form of communication should be planned, either by inviting customer representatives to regular planning and review meetings, or by performing timeboxed, intensive sessions (e.g. design thinking or design sprint, see EU 1.5) at an early stage with resulting input captured in the backlog.

EU 4.2.2 Product vs. project organization

Agile promotes direct, open and non-hierarchical relationships within the organization and flexibility in precisely how products evolve over time. Larger organizations, traditionally based on top-down management structures, place a high value on planning and predictability. Project and resource planning, for example, are realities from which enterprises cannot simply opt out.

Software development has traditionally been project-based, meaning that it takes place as a series of temporary undertakings to produce unique products, services or results (see [PMI]). Groups of related projects with shared aims or goals are typically called programs, while the planning and control of the entirety of projects and programs within an organization is referred to as portfolio management.

True to its roots in product development, the Agile approach is more product-centric. A backlog of product improvements is maintained and these are implemented in an iterative process of continuous improvement. Agile alone does not define an end date as such. As long there are improvements to be made or benefits to be realized, work should, in principle, be continued if the benefits outweigh the effort/costs.

While these approaches are not mutually exclusive (the scope of a project may be to deliver a particular product and additional projects are established for later improvements) the differences of perspective and terminology may be a source of tension and misunderstanding between Agile software development and non-Agile organizations.

One approach to resolve such tensions is that while software development itself takes place in a strict Agile manner, the functions of portfolio and program management provide a higher level of planning and control which uses approaches from both worlds (see also EU 4.3). Key to making such an approach work is the ability to bridge conceptual gaps between the planned fulfilment of business goals and features at portfolio and program levels and an iterative and flexible delivery of individual software features.

RE provides concepts and methods necessary to differentiate between requirements at these different levels of abstraction with business level requirements at the portfolio and program levels and derived and detailed software requirements suitable for the development backlog. The requirements approach shifts from more detailed and precise requirements used as exact orders for development to requirements that are used as a common basis for discussion and alignment just in time and as detailed as needed for the current level of planning.

EU 4.2.3 The role of management in an Agile context

Disciplines & Teams: Staff within IT departments have traditionally been organized by discipline: developers, testers, requirements engineers, business analysts, project managers, etc. Project teams are assembled from these various skillsets for the fixed duration of the undertaking. Agile, and Scrum in particular, promotes the idea of more cross-functional teams. Aside from the specialist roles of Product Owner and Scrum Master, all team members should be capable of acting in different capacities, with different individuals supporting Requirements Engineering or testing activities as required. The goal of the team is to be able to deliver customer requirements in full, whatever the technological or organizational elements involved. This can be achieved by having RE competencies within the development team (preferred solution) or outside the development team (often used in reality as support for the Product Owner) which then is officially not part of the Scrum framework.

IT managers: It is the duty of IT managers (called "people developers" in [SAFe]) to find the right balance in their organization between specialist and generalist skillsets and to help the teams to organize those skills into an appropriate number of teams. With respect to Conway's law[Conw1968], the team structure will be a mirror of the product structure (components) or the system structure in IT. This is even more important in scaling the number of teams. If a company organizes their teams by components or systems, scaling by increasing the number of teams does not help as it creates even more dependencies. Scaling would only be possible by scaling the number of team members, but only up to a point as communication effort will also rise dramatically. Cross-functional teams that inherit all capabilities to deliver whole increments from frontend to backend can be scaled easily – but they are not easy to build and it can be time consuming to do so.

Product Owner vs. Project Manager: As discussed earlier (see EU 2.2), Product Owners expand on the responsibilities of requirements engineers by assuming responsibility for business priorities in the requirements fed to product development teams. Product Owners must be enabled (knowledge) and empowered to make business decisions. Some decisions previously taken by project managers are thus no longer required when working with Agile approaches as the development teams are self-organizing and only need the work to do (requirements on a level of detail that allows development within a a sprint) to organize their work (task breakdown and assignment within the team) themselves.
What is required of Agile IT managers is a clear setting of the vision for Agile development and a clear communication of the cultural prerequisites for this approach to be successful.

Getting the balance right while meeting the expectations of the business is no simple matter; some criteria for success are discussed below in EU 4.4.

Role of Requirements Engineering (RE)/ Requirements Management (RM): The previous patterns also apply to Scrum as an example. RE people can work directly with the team as part of the development team and will therefore not be named separately or they can form a team themselves supporting one or multiple Product Owners as a team. Both approaches have their advantages and can be used in combination without violating Agile principles. RE will become the backbone of successful Agile development.

EU 4.3 Handling of complex problems by scaling (L1)

EU 4.3.1 Motivation for scaling

The Agile values of direct, daily, non-hierarchical communication are typically represented by small, close-knit teams such as the Scrum team with its recommended 5-11 Scrum team members (3-9 development team members + Product Owner + Scrum Master). Team members should ideally be physically co-located and cross-functional in terms of business and technical skills.

In larger organizations, this ideal view on the Agile development may not be feasible for many reasons:

- complex problems may involve stakeholders and knowledge from different parts of the business that cannot easily be accommodated in a single team.
- complex problems may involve a range of technical specialists and knowledge that cannot easily be accommodated in a single team.
- the scope required by a defined rollout date is simply beyond the achievable velocity of a single team.
- staff within global businesses may be geographically distributed.

The term Scaled Agile is used to describe situations where multiple teams (mostly Scrum Teams) are required to work together on one product/solution sharing common goals. Scaled Agile approaches require decisions as to how Scrum teams are organized and how communication among the teams is to be coordinated. The goal is to achieve an effective approach for handling complex problems, while retaining as many advantages of Agility as possible.

EU 4.3.2 Approaches for organizing teams

The question to be answered is just how the organization should arrange itself into teams of a size that allows the teams to be cross-functional (minimum size) but at the same time effective (maximum size) with regards to communication and alignment. Organizing along functional lines (e.g. a team specializing in a defined business area, or even on a feature within a business area) has the advantage of concentrating business knowledge within a single team. This may ease the elicitation of requirements, for example, by reducing the number of business stakeholders directly involved with the team.

A disadvantage of this approach is that delivering end-to-end functionality in a business area is likely to involve several different technical specialties, such as user interface design, process engines, databases and core platforms such as ERP or mainframes which can easily overstrain the max size of an effective team.

An alternative way to organize development is along technical lines, with teams specializing in technical components or platforms.

The advantage of component or platform teams is the deep knowledge in their respective technology field. The disadvantage is the dependencies that such an approach creates between teams working together to deliver the whole product increment to schedule.

The scaling frameworks (like those named inEU 4.3.4) show ways to handle this situation. Requirements engineering activities must align with the framework to achieve a common view of what can be delivered.

In this scenario, RE has a particularly important role to play in first breaking down business goals and requirements into constituent sub-system requirements that can then be assigned to individual teams, and secondly following the development through to ensure that an integrated solution is the result and that the business benefit is indeed delivered.

A mixture of all team types may provide the best and most pragmatic solution to benefit from the ideas of feature teams while taking into consideration company-specific constraints.

EU 4.3.3 Approaches for organizing communication

In answering the question of how many teams can work together efficiently, we can distinguish between two different approaches:

1. Use methods from single team approaches
2. Introduce additional concepts to organize communication and responsibilities

Use methods from single team approaches: The first approach follows the idea that additional artifacts and roles are not required and the communication between a number of teams should be supported with the existing techniques from single team approaches. Additional roles and artifacts would be contrary to Agile thinking and lead to additional complexity in the organization. No overhead based on new roles should be created, following the basic principle "keep it simple". The communication and coordination between the teams is usually initiated by the Product Owners of the teams but done by team delegates. Constructs like Communities of Practice enable the team members across the teams to share experiences and coordinate overall processes.

Introduce additional concepts: The second approach recommends dividing larger problems into smaller problems and managing responsibilities by different roles for different abstractions. Thus additional artifacts should be introduced for the different abstraction levels (e.g. Business Epics, Architectural Epics, Investment Themes, Features, User Stories). Depending on the framework used, additional roles are also introduced with responsibility for the different abstraction levels, (e.g. Portfolio Managers, Product Managers and Product Owners). Because of the increasing complexity, additional artifacts and roles are also required to manage the planning and communication among the different teams and to achieve integrated results per iteration (e.g. Roadmaps and Release Managers). In addition, specific meetings have to be organized to promote the communication between new and already established roles. RE provides a lot of techniques that could help to sub-divide larger problems and to support the new roles at the different abstraction levels (e.g. context modelling, goal modelling).

Both approaches have their advantages and disadvantages and each user/enterprise needs to find their way best based on the business case.

EU 4.3.4 Example Frameworks for scaling RE@Agile

Frameworks that support scaling of Scrum and Agile: There are many different frameworks available that support these approaches and because of the increasing importance of scaling Agility, this number is also growing rapidly. You will find a selection of the most well-known frameworks following:

- Scaled Agile Framework (SAFe)

  SAFe is a knowledge base of proven success patterns for implementing lean-Agile software and system development on enterprise scale. [SAFe]

- Large-Scale Scrum (LeSS)

  LeSS is Scrum applied to many teams working together on one product. [Less]

- Nexus

  Nexus is a framework that drives to the heart of scaling: cross-team dependencies and integration issues. [Nexus2015]

- Scrum@Scale:

  The Scrum at Scale framework is a minimal extension of the core Scrum framework that keeps the modular structure at the core of the Scrum framework, and allows to scale a Scrum implementation tailored to the unique needs of your company. [SatS]

- Disciplined Agile Delivery (DAD)

  Disciplined Agile (DA) is a process decision framework for lean enterprises. It is tactically scalable at the team level and strategically scalable across all of the enterprise [AmLi2012]

EU 4.3.5 Impacts of Scaling on RE@Agile

The abstraction layers discussed above mean that RE activities are managed by more roles such as Product Owner, Product Manager, Portfolio Manager, Business Analyst. Every role will create corresponding RE artifacts for their respective area of concern (Epics, Features, User Stories and the traceability between them). Additional meetings for RE activities are required (e.g. story times, feature times, system demos), while communication with stakeholders is performed by different roles at various levels within the organization.

As Scrum itself does not scale easily on its own terms, RE plays a critical role in determining how overarching requirements are decomposed and distributed appropriately amongst multiple Agile teams in order to keep any scaling approach alive. RE techniques help in structuring problem analysis and refining coarse-grained requirements to fine-grained requirements like features and user stories appropriate for individual teams. A structured approach, applying appropriate abstractions and different analysis perspectives, will provide a sound basis for a sub-division of tasks amongst semi-autonomous Agile development teams.

This especially applies to dependencies within the requirements that need to be found and discussed as early as possible to avoid waste in the development.

An additional challenge will occur if teams or other roles are working at different locations. The complexity of managing communication increases not only because of a possible time shift or different languages. In most cases, the main challenge is based on different cultures. Because communication is one of the main tasks of Product Owners and Product Managers, this has a significant influence on the required skills of the RE-relevant roles.

EU 4.4 Balancing upfront and continuous Requirements Engineering in the context of scaling (L1)

On a very abstract level, Agile methods can be characterized as a continuous, iterative process in which the system is developed incrementally based on the backlog items. From a Requirements Engineering perspective, five parameters (see following subchapters) can be identified that drive this process:

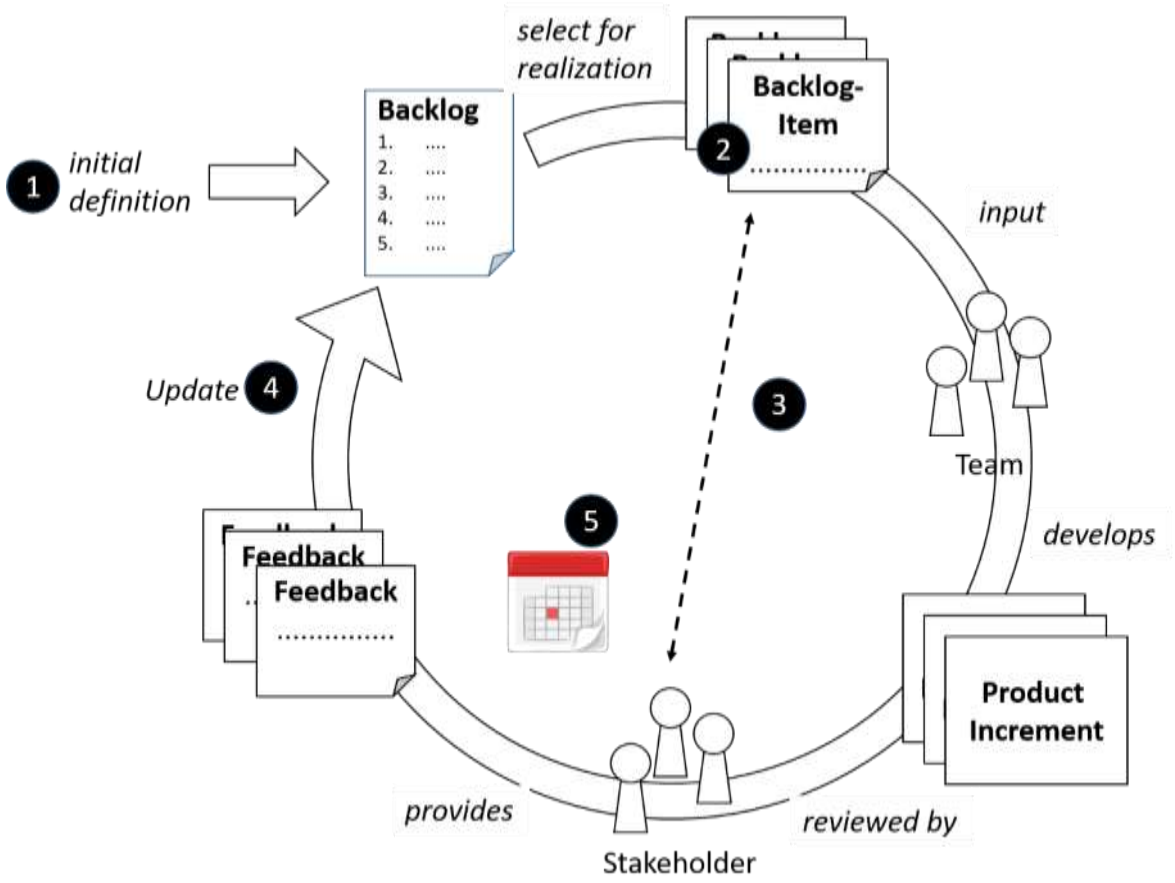

Figure 2: Continuous Requirements Engineering Process

EU 4.4.1 Initial Requirements Definition

Before starting the continuous development process, an initial backlog has to be created (see [TaNo1986]). This initial definition of the backlog is often referred to as "upfront definition". Requirements Engineering is sometimes understood to stipulate that this upfront definition be a complete and detailed specification of all requirements. This might indeed have been the case for some specific methodologies, but it must not always be so.

The baseline for the initial definition of the backlog is the amount of information that is needed to start the first iteration of the continuous development process. Just when to start the iteration process is a key decision. Depending on the system and the context, uncertainty concerning requirements (starting without enough knowledge) may lead to additional delays, costs or potentially to project failure. Conversely, starting too late may have a negative impact on time to market and the suitability to end users' needs at a particular point in time.

Requirements Engineering tells us that those requirements that can be identified as having a high impact on the architecture, on the overall feasibility of the solution or on key choices concerning infrastructure and hardware should be elicited and detailed earlier. Such architecture-relevant issues should in fact be elicited and analyzed before the first iteration of development. Lower impact requirements may then be refined during the iterations as a continuous process.

Within the iterative and incremental processes of Agile, requirements engineering becomes a continuous process that delivers and refines requirements just-in-time and in just enough detail to feed the development cycle. The process resembles a funnel for crushing stones, where big stones enter the funnel and are crushed into mid-size and finally into small stones by the time the process is finished.

EU 4.4.2 Level of Detail for Backlog Items

The level of detail of a backlog item constrains the freedom of the development team for the realization of a backlog item (see [Pich2010]). All aspects of a backlog item that have not been defined are left as decisions for the team, which should allow the team to be more creative but within the defined boundaries of the business.

The available competencies of the development team can serve as a rule of thumb. If the development team has sufficient competence to decide on the details of a backlog item (e.g., an expert of authentication & authorization is part of the team), the decision on the details should be left to the team.

EU 4.4.3 Validity of Backlog items

Knowing the validity of a backlog item before the implementation takes place helps to minimize unnecessary implementation work (see [Denn2015]). Waiting to recognize a wrong or incomplete requirement in the implemented software is a very expensive approach for requirements validation.

Determining the validity of a backlog item is closely related to the level of detail of the particular backlog item. The validity of a backlog item can only be determined when the backlog item has sufficient detail. Therefore, the effort for elaborating and validating a requirement prior to the implementation has to be compared with the assumed effort for implementing the requirement and validating it afterwards in the delivered software.

When the stakeholder preference related to a backlog item can be determined with acceptable effort, such requirements validation should take place before the backlog item is developed. Typical examples for such types of requirements (including an exemplary validation approach) are: overall design of the user interface (e.g. with UI mockups), authorization and authentication mechanisms (e.g. with use case reviews), data structures that have to be stored in the system (e.g. with data model reviews) and requirements for interfaces to existing systems (e.g. with activity diagram reviews).

Backlog items which are high risk (e.g. critical business functions, safety critical functions, innovative functionalities) or have a high testing cost for the implementation (e.g. the software has to be tested in an expensive prototype) should be validated prior to their implementation.

EU 4.4.4 Feedback and Update of the Backlog

Backlogs are often updated based on feedback from an inspection activity such as the sprint review. Such an approach is possible for small scale or detailed requirements for which the impact of a change can be analyzed and grasped quickly in an environment with one or two small development teams. It is not advisable to modify requirements which are of greater complexity or that have multiple dependencies at short notice. In such situations, the modification of the backlog takes more time, stakeholders may not be available and additional analysis will be necessary.

Another factor that may have an impact on the modification of backlog items is the organization's decision-making process. In organizations where significant decisions may take some time (e.g. the responsible council only meets once every three months), the principle of continuous refinement needs to take the form of concrete meetings involving all affected stakeholders. Such meetings need RE as preparation and decision support.

EU 4.4.5 Timing of the Development Cycle

The final parameter for the development process is the time schedule or length of the iteration. This has a significant effect on the Requirements Engineering activities that need to be performed while working on backlog items not currently under development. Furthermore, the iteration length determines the frequency in which results are delivered to business stakeholders or available customers for review.

Consequently, shorter iteration lengths increase the workload on the business stakeholders for three reasons (cf., e.g. [Rein1997]):

1. Business stakeholders must be available throughout the iteration for working on the backlog to create input to the next iteration.

2. Business stakeholders must review the results created by the team in order to provide feedback.

3. Business stakeholders suffer from several context switches between their daily business and project work.

Longer iteration lengths reduce the pressure but also reduce the capability of influencing the backlog for product development.

The definition of the iteration length has to be made with the availability of the business stakeholders in mind. Business stakeholders are typically not 100% available for development activities since they have other duties in the organization for which the system is developed.

As a rule of thumb, a shorter iteration time provides frequent feedback and more opportunities to discover errors early, thus shorter iterations tend to accelerate the development activities. If a short cycle time represents an unacceptable load for the stakeholders, then a compromise iteration length should be found, though this may shorten in proportion to the project's priority.

Another factor that can have an impact on the cycle time is the average size and complexity of backlog items. Larger or more complex backlog items consume more time for understanding and analysis. Therefore the cycle time can be increased to deal with larger or more complex backlog items within a single iteration. For example, if the system under development is in an early stage, a longer cycle may be advisable to give the team more time to gain an initial understanding of the system. Nevertheless, this factor must be balanced with the goal of using shorter cycle times to get frequent feedback. The decision whether to have longer or shorter cycle times has to be made together as a team weighing up the need for analysis with the goal of early feedback. Changing cycle times is always based on the "inspect & adapt" principle, bearing in mind that past experience is not always a guarantee for the future. Changing the cycle time should take place before the sprint, never within.

EU 5 DEFINITONS OF TERMS, Glossary (L2)

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

EU 6 REFERENCES

[ACP/PMI] Agile Certified Practisioner: http://www.pmi.org/certification/Agile-management-acp.aspx. Last visited January 2017.

[AgileMan2001] Agile Manifesto: http://www.Agilemanifesto.org, 2001. Last visited January 2017.

[AmLi2012] Ambler S.; Lines, M.: Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise. IBM Press, 2012

[Ande2012] Anderson, D.J.: Lessons in Agile Management: On the Road to Kanban. Blue Hole Press, 2012

[Appel2011] Appelo, J.: Management 3.0 Addison-Wesley Professional, 2011

[Beck2003] Beck, K.: Test-Driven Development by Example. Addison Wesley – Vaseem, 2003

[Beck2004] Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley Professional, 2004

[CNYM2000] Chung, L.; Nixon, B. A.; Yu, E.; Mylopoulos, J. : Non-Functional Requirements in Software Engineering. Springer Science & Business Media, 2000

[Cock1998] Cockburn, A.: Surviving Object-Oriented Projects. Addison-Wesley, 1998

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

[Conw1968] Conway, M.: How Do Committees Invent? Datamation 14(4):28–31, 1968. Article available at http://www.melconway.com/Home/Conways_Law.html. Last visited 2017

[Denn2015] Denning, S.: How To Make The Whole Organization Agile. http://www.forbes.com/sites/stevedenning/2015/07/22/how-to-make-the-whole-organization-agile/#658d3f65135b, 2015. Last visited January 2017.

[Dsch2015] d.school: An Introduction to Design Thinking – Process Guide. https://dschool-old.stanford.edu/sandbox/groups/designresources/wiki/36873/attachments/74b3d/ModeGuideBOOTCAMP2010L.pdf, 2015. Last visited January 2017.

[Glinz2014] Glinz, M.: A Glossary of Requirements Engineering Terminology, Version 1.6. https://www.ireb.org/downloads/#cpre-glossary, 2014. Last visited January 2017.

[Griff2015] Griffiths, M.: PMI-ACP Exam Prep. Rmc Publications, 2015

[GoAk2003] Gordijn, J.; Akkermans, J.M.: Value-based Requirements Engineering : exploring innovative e-commerce ideas. Springer, 2003

[High2009] Highsmith, J.: Agile Project Management: Creating Innovative Products. Addison-Wesley Professional, 2009

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

[IREB2015] IREB e.V.: Syllabus CPRE Foundation Level, version 2.2. https://www.ireb.org/downloads/#syllabus-foundation-level, 2015. Last visited January 2017.

[KnZK2016] Knapp, J.; Zeratsky, J; Kowitz, B.: Sprint: How to Solve Big Problems and Test New Ideas in Just Five Days. Simon & Schuster, 2016

[Kron2008] Kronfaelt, R.: Ready-ready: the Definition of Ready for User Stories going into sprint planning. http://scrumftw.blogspot.de/2008/10/ready-ready-definition-of-ready-for.html, 2008. Last visited January 2017.

[Less] Large Scale Scrum: http://less.works. Last visited January 2017.

[LiOg2011] Liedtka, J.; Ogilvie, T.: Designing for Growth: A Design Thinking Tool Kit For Managers. Columbia Business School Publishing, 2011

[Martin1991] Martin, J.: Rapid Application Development. Macmillan Coll Div, 1991

[Meyer2014] Meyer, B.: Agile! – the good, the hype, and the ugly. Springer, 2014

[MeMi2015] Mesaglio, M., Mingay, S.: Bimodal IT: How to Be Digitally Agile Without Making a Mess, Gartner 2015 https://www.gartner.com/doc/2798217/bimodal-it-digitally-agile-making. Last visited January 2017.

[Nexus2015] Nexus Guide https://www.scrum.org/Portals/0/NexusGuide%20v1.1.pdf, 2015. Last visited January 2017.

[Patt2014] Patton, J.: User Story Mapping. O'Reilly, 2014

[Pich2010] Pichler, R: Make the product backlog deep. http://www.romanpichler.com/blog/make-the-product-backlog-deep/, 2010. Last visited January 2017.

[PMI] PMI Project Management Institute. http://www.pmi.org/. Last visited January 2017.

[Popp2003] Poppendieck, M.: Lean Software Development: An Agile Toolkit. Addison-Wesley Professional, 2003

[Rein1997] Reinerstsen, D. G.: Managing the Design Factory – A Product Developer's Toolkit. Simon & Schuster, 1997

[Ries2011] Ries, E.: The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses, Crown Publishing Group, 2011

[SAFe] SAFe – Scaled Agile Framework. http://www.scaledAgileframework.com. Last visited January 2017.

[SatS] Scrum at Scale Framework: https://www.scruminc.com/scrum-incs-scrum-at-scale-framework/. Last visited January 2017.

[Scrum2016] Schwaber, K. & Sutherland, J.: The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game, July 2016. http://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf. Last visited January 2017.

[ShYo2006] Sheppard, J. M.; Young W. B.: Agility literature review: Classifications, training and testing. Journal of sports sciences 24(9): 919-932, 2006

[TaNo1986] Takeuchi, H.; Nonaka, I.: The new new product development game. Harvard Business Review 64(1), January/February 1986, p.137-146

[Wake2003] Wake, B.: Invest in Good Stories and Smart Tasks, http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/. Last visited January 2017.