Syllabus

REQB®
Certified Professional for Requirements Engineering

Agile Practitioner

Requirements Engineering Qualifications Board

Version 1.0

2015

The copyright ® to this edition of the syllabus in all languages is held by the
Global Association for Software Quality, GASQ.

Overview of Changes

Version	Date	Comment
0.1	May 20th, 2014	First version of the syllabus
0.2	July 31st, 2014	Reduction of introduction and of chapters recalling general Agile concepts
0.3	September 9th, 2014	New version of the syllabus after first review
0.9	December 15th, 2014	New version of the syllabus after second review
1.0	January 31st, 2015	Final version 1.0

Acknowledgements

This document was produced by a core team from the Requirements Engineering Qualifications Board - Advanced Level Working Party.

At the time the Agile Practitioner syllabus was completed, the core team had the following membership: Salvatore Reale (Agile Stream Lead), Alain Ribault (WP Chair), Folke Nilsson (REQB Chair), Alexander Lindner.

Author: Salvatore Reale.

Reviewers: Werner Henschelchen, Beata Karpinska, Alexander Lindner, Judy McKay, Folke Nilsson, Ingvar Nordström, Tal Pe'er, Alain Ribault.

The core team thanks the members of the National Boards that have submitted contributions and suggestions.

Table of Contents

© GASQ – Global Association for Software Quality

© GASQ – Global Association for Software Quality

0 Introduction

0.1 Purpose of the Syllabus

This syllabus defines the training program to become a REQB Certified Professional for Requirements Engineering (CPRE) at the Practitioner Level for Agile Requirements Engineering. REQB developed this syllabus in cooperation with the Global Association for Software Quality (GASQ). The scope of the REQB program covers the process of requirements engineering for all types of IT-related products consisting of software, hardware, services, business processes, and documentation as well. Within this scope the Agile Practitioner syllabus covers the process of requirements engineering for software development organizations using Agile methodologies.

REQB provides this syllabus as follows:

1. To National Boards, to translate into their local language and to accredit training providers. Translation may include adapting the syllabus to the particular language needs and modifying the references (books and publications) to adapt to the local publications.
2. To Exam Boards, to create examination questions in their local language adapted to the Learning Objectives defined in this syllabus.
3. To Training Providers, seeking accreditation as REQB recognized training providers, to produce courseware. All areas of this syllabus must correspondingly be incorporated in the training documents.
4. To Certification Candidates, as preparation material for the certification exam (as part of an accredited training course or independently).
5. To the international requirements engineering community, to advance the profession of a Requirements Engineer.

0.2 Examination

The examination to become a Certified Professional for Requirements Engineering - Agile Practitioner - is based on this syllabus. All sections of this syllabus can thereby be addressed by exam questions. The examination questions are not necessarily derived from an individual section; a question may refer to several sections.

The format of the examination questions is multiple choice.

Examinations can be taken after having attended accredited courses or in open examination sessions (without a previous course). Detailed information regarding examination times can be found on REQB's website (www.reqb.org), on the website of GASQ (www.gasq.org) or on the website of a local examination provider.

Note:

This syllabus refers to requirements engineering basics described in the Foundation Level syllabus. Exam questions, as well as the terminology and the subjects covered during the training, may refer to the content of the Foundation Level syllabus [REQB_FL_SYL]**, so a reading of that document** [REQB_FL_SYL] **or knowledge of the concepts contained therein is recommended.**

© GASQ – Global Association for Software Quality

0.3 Accreditation

Providers of a REQB Certified Professional for Requirements Engineering - Agile Practitioner course must be accredited by the Global Association for Software Quality. Their experts review the training provider's documentation for accuracy and compliance with the content and Learning Objectives of the syllabus. An accredited course is regarded as conforming to the syllabus. At the end of such a course, a Certified Professional for Requirements Engineering examination (CPRE exam) may be conducted by an independent certification institute (according to ISO 17024 rules).

Accredited training providers can be identified by the official REQB Accredited Training Provider logo.

0.4 Internationality

This syllabus was developed in cooperation between international experts. The content of this syllabus can therefore be seen as an international standard. The syllabus makes it possible to train and examine internationally at the same level.

0.5 Business Benefits

Objectives, benefits and the main focus of the REQB Certified Professional for Requirements Engineering - Agile Practitioner program are presented in the following table.

Objectives	**Benefits**
Obtain new key qualification	Any software, hardware or service solution is based on stakeholders' requirements and aims to satisfy specific business needs. To be able to deliver a solution compliant with the needs of the target group, proper requirements engineering is necessary. Since 2001, when the Manifesto for Agile Software Development was made public, the number of companies, organizations, development groups that adopt Agile methodologies for software development has grown from year to year. Managing the requirements and developing them into the solution design, while working within an Agile context, is becoming a growing need. All those aspects are covered by an Agile Practitioner.
Increase your customers' satisfaction	Customer satisfaction is achieved when they experience the solution that meets their expectations and needs. Improved requirements engineering minimizes discrepancies between the expectations and the perception of conformance. The first principle of Agile software development puts the highest priority in satisfying the customer through early and continuous delivery of valuable software. Agile

© GASQ – Global Association for Software Quality

	requirements engineering provides the means to enhance the quality of the final product and to strengthen customer loyalty by providing what they want.
Minimize development and follow-up costs	Proper requirements engineering in Agile software development, by delivering potentially shippable increments and through frequent feedback cycles, minimizes the project and product risks and helps avoid costs related to rework resulting from discrepancies between the customer's expectations and the delivered solution. Agile requirements engineering helps in minimizing costs for changes and corrective actions.
Competitive advantage	The value of any feature on the market decreases over time. Therefore, to get the maximum profit an organization should be first to market, or at least early enough. Agile requirements engineering helps to deliver products and services faster, meeting all business needs and expectations.

The level of Agile Practitioner in the REQB Certified Professional for Requirements Engineering program is suitable for all persons who are involved in product/business solution development and maintenance, including business and system analysts, marketing teams, customer representatives, product owners, hardware/software designers, hardware/software developers, testers, project managers, maintenance and technical staff, Scrum Masters, IT auditors and quality assurance representatives.

The main purpose of the Agile Practitioner program is to provide a common terminology and a common understanding of the key concepts related to the requirements engineering process when working in an Agile context. The knowledge base provided in the Agile Practitioner syllabus supports common definitions and concepts related to Agile requirements engineering and it explains the Agile requirements engineering process together with its deliverables. The syllabus is based on generally accepted standards and best practices. Table 1 provides a brief description of the objectives and focus of this program.

Objectives	**Focus**
Agile Basics	Recalling the basic values and principles of Agile software development, its most commonly used methods, the flow and the key practices of an Agile project. Includes a mention about the applicability of the rules and standards to an Agile context.
Agile Requirements	Understanding the basic concepts related to Agile requirements, their classification and levels of abstraction; explaining the meaning of requirements attributes and the role of quality criteria.
From Requirements to User Story	Explaining the progressive refinement of a requirement until it is split into user stories that can be implemented and completely delivered within one iteration.
Roles in Agile Requirements Engineering	Explaining how the classic roles of the Agile organization contribute to requirements engineering.

Agile Requirements Development	Understanding what changes in requirements elicitation, analysis, specification, verification and validation are needed when operating in an Agile context.
Agile Requirements Management	Understanding what changes in risk management, requirements traceability, configuration and change management, and quality assurance are needed when operating in an Agile context.
Tools and Artifacts	Explaining the basic artifacts and tools supporting Agile requirements engineering and how they can fulfill the needs of most Agile organizations.
Exercises	Identifying stakeholders; identifying requirements; refining good requirements by means of user stories with INVEST properties; ensuring quality requirements; analyzing and estimating requirements; verifying and validating.

Table 1 Objectives of the Agile Practitioner program, its benefits and main focus

0.6 Learning Objectives

The Learning Objectives of this syllabus have been divided into different cognitive levels of knowledge (K-levels). This makes it possible for the candidate to recognize the "knowledge level" of each point.

Each section of this syllabus has a cognitive level associated with it:

- K1 - Proficiency/Knowledge: Knowledge of precise details such as terms, definitions, facts, data, rules, principles, theories, characteristics, criteria, procedures. Students are able to recall and express knowledge.
- K2 - Understanding: Students are able to explain or summarize facts in their own words, provide examples, understand contexts, interpret tasks.
- K3 - Apply: Students are able to apply their knowledge in new specific situations, for example, by applying certain rules, methods or procedures.
- K4 - Analyze: Students are able to analyze new specific problems and give appropriate solutions based on their varied knowledge and skills.

Note:

No K4 Learning Objectives are included in this version of the syllabus.

0.7 Business Outcomes

After completing the REQB Certified Professional for Requirements Engineering - Agile Practitioner program, a person can:

© GASQ – Global Association for Software Quality

- BO01 Communicate the fundamental concepts and models and understand the roles of Agile software development for a product lifecycle.
- BO02 Understand and communicate the fundamental concepts of requirements and their application into a product lifecycle within an Agile context.
- BO03 Create awareness of the meaning of the requirements engineering process and its deliverables in Agile software development.
- BO04 Create awareness of the attributes and the quality of the requirements in Agile software development.
- BO05 Identify the stakeholders and involve them in the requirements engineering activities: collection, workshops, planning, estimating, acceptance.
- BO06 Understand and communicate the responsibilities, tasks, skills and contribution in supporting the requirements engineering process from the people playing the key roles in an Agile software development project.
- BO07 Understand how to improve the requirements engineering process within the Agile context.
- BO08 Understand, communicate and apply the differences in the requirements development between traditional and Agile development/maintenance processes.
- BO09 Understand, communicate and apply the differences in the requirements management between traditional and Agile development/maintenance processes.
- BO10 Understand and apply the estimation criteria for the requirements in the Agile context.
- BO11 Communicate the importance and the applications of Agile artifacts and tools for keeping an Agile project under control and supporting requirements development and management.

0.8 Level of Detail

The REQB Certified Professional for Requirements Engineering - Agile Practitioner syllabus is intended to support internationally consistent training and examination. This syllabus comprises the following components to reach this goal:

- General instructional objectives describing the intention of the Agile Practitioner program of REQB certification.
- Learning objectives for each knowledge area describing the cognitive learning outcomes of the course and the qualifications that the participant is achieving.
- A list of information to teach, including a description, and references to additional sources such as accepted technical literature, norms or standards, if required.
- A list of terms that participants must be able to recall and understand. A list of individual terms is described in detail in the REQB "Standard Glossary of Terms used in Requirements Engineering" document.

The syllabus content is not a description of the entire "Requirements Engineering" field of knowledge. It covers the scope and level of detail relevant for Agile Practitioner certification.

© GASQ – Global Association for Software Quality

0.9 Organization of the Syllabus

There are six major chapters and a final reference chapter.

The top-level heading for each chapter shows the topic that is covered within the chapter and specifies the minimum amount of time that an accredited course must spend on the chapter.

Learning objectives to be satisfied by each chapter are listed at the beginning of the chapter.

Within each chapter there are a number of sections. Each section has a minimum amount of time that an accredited course must spend in that section. Subsections that do not have a time associated with them are included within the overall time for the section.

1 Introduction to Agile Software Development (80 minutes)

Terms

Agile Manifesto, Agile software development, Agile Team, continuous integration, continuous delivery, cross-functional team, development Team, Extreme Programming, kanban, incremental model, iteration planning, iterative model, product backlog, product increment, Product Owner, release planning, retrospective, Scrum, Scrum Master, sprint, timebox, , user story, whole-team approach

Learning Objectives for Introduction to Agile Software Development

1.1 The Fundamentals of Agile Software Development (35 minutes)

AR-1.1.1	Recall the basic values and principles of Agile software development based on the Agile Manifesto (K1)
AR-1.1.2	Recall Agile software development approaches (K1)
AR-1.1.3	Understand the advantages of a cross-functional team and the whole-team approach (K2)

1.2 General Aspects of Agile Software Development (45 minutes)

AR-1.2.1	Recall the basics of user stories, release planning and iteration planning (K1)
AR-1.2.2	Recall the benefits of continuous integration (K1)
AR-1.2.3	Recall the benefits of early and frequent feedback (K1)
AR-1.2.4	Recall how retrospectives can be used as a mechanism for process improvement in Agile projects (K1)

1.1 The Fundamentals of Agile Software Development

A requirements engineer on an Agile project will work differently from one working on a traditional project and will play a role that will be different. The requirements development and management processes and practices also will be different than those on a traditional project, but are still necessary to achieve successful results in an Agile project.

Requirements engineers must understand the values and principles that underpin Agile projects, and how requirements engineers are an integral part of a whole-team approach together with developers, testers and business representatives.

© GASQ – Global Association for Software Quality

1.1.1 The Manifesto for Agile Software Development

The roots of Agile software development can be traced back to 2001 when a group of seventeen people with long experience in lightweight models for software development agreed on four values and 12 principles that form the Manifesto for Agile Software Development [Agile Manifesto, 2011].

The Agile Manifesto emphasizes the great importance of four agile values for sustained competitive success and profit:

1. Individuals and interactions **over** processes and tools
2. Working software **over** comprehensive documentation
3. Customer collaboration **over** contract negotiation
4. Responding to change **over** following a plan

The Agile Manifesto also provides twelve Agile principles that are a basic reference for any Agile method:

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.

From the underlined words the strong relationship among the Agile principles and requirements engineering is clear.

The different Agile methods provide rules and practices to put these Agile values and principles into action [Schwaber, 2001], [Beck, Andres, 2004], [Anderson, 2010].

1.1.2 Agile Software Development Methods

There is no single Agile software development approach, but many different methods. Each of these implement the values and principles of the Agile Manifesto in different ways. According to the most recent surveys about Agile methods adoption, the three most popular representatives of Agile methods are Scrum, Extreme Programming and Kanban. Each of these is discussed in this chapter.

Scrum is adopted in more than 50% of cases. After Scrum, XP, Kanban and hybrids between these last two and Scrum account for the other 50%. The other methods are adopted with lower percentages.

© GASQ – Global Association for Software Quality

Scrum

Scrum is an iterative, incremental framework for projects and product or application development.

It structures development into iterations called sprints. These iterations are 1-4 weeks in length, and take place one after the other. The sprints are of fixed duration – they end on a specific date whether the work has been completed or not, and are never extended; they are timeboxed.

At the beginning of each sprint, a cross-functional team selects items (customer requirements) from a prioritized list. The list can change from the previous sprint, enabling faster adaptation to changes. The team commits to complete the items by the end of the sprint.

During the sprint, the chosen items do not change. Every day the team gathers briefly to report to each other on progress, and update simple charts that orient them to the work remaining. This daily meeting is called Daily Scrum, or Stand-up meeting.

The most important Scrum artifact is the product increment. Every sprint produces a product increment. The product increment must be of high enough quality to be given to the users. When the product increment is delivered, it needs to be "done" according to a shared understanding of what “done“ means. This definition is different for every Scrum team, and as the team matures, the “Definition of Done“ will expand and become more stringent. The Definition of Done must always include the notion that the product increment is of high enough quality to be shippable if the team should choose to release it immediately. The product increment includes the functionality of all previous product increments and is fully tested so that all completed items continue to work together.

At the end of the sprint, the team reviews the sprint with the stakeholders and demonstrates what they have built. From this review the team obtains feedback that can be incorporated into the next sprint. Scrum emphasizes a working product at the end of the sprint that is really “done”; in the case of software, this means code that is integrated, fully tested and potentially shippable.

A major theme in Scrum is “inspect and adapt” with retrospective. Since development inevitably involves learning, innovation, and surprises, Scrum emphasizes taking a short step of development, inspecting both the resulting product and the efficacy of the current practices, and then adapting the product goals and process practices.

Scrum is a team process. The Scrum team includes three roles, the Product Owner, the members of the development team, and the Scrum Master.

The product backlog is an essential artifact in Scrum. The product backlog is an ordered list of ideas for the product, kept in the order the team expects to address them according to the priority assigned by business value, customer priority, risk and so forth. It is the single source from which all requirements flow. This means that all work the development team does comes from the product backlog. Every feature idea, enhancement, bug fix, documentation requirement – every bit of work they do – is derived from a product backlog item. Each prioritized item on the product backlog includes a description and an estimate.

The Product Owner represents the customer, and generates, maintains, and prioritizes the product backlog. This person is not the team lead.

The development team is made up of the professionals who do the work of delivering the product increment. The team needs to understand the vision and sprint goals of the Product Owner in order to deliver potentially shippable product increments. They self-organize to accomplish the work.

Scrum requires that the development team is a cross-functional group of people who, among them, have all the necessary skills to deliver each increment of the product.

The sprint backlog is the list of refined product backlog items chosen for development in the current sprint. It reflects the team's forecast of what work can be completed. With the sprint backlog in place, the sprint begins, and the development team develops the new product increment as defined by the sprint backlog.

The Scrum teams do not emphasize any of the traditional roles of software engineering, such as a programmer, designer, tester, or an architect. Everyone on the project is working together to complete the set of work to which they have committed collectively within a sprint.

Note:

In this syllabus the term “Team” (with capital T) indicates the Scrum team.

The Scrum Master has the following responsibilities:

- Ensuring the Scrum Team lives by the values and practices of Scrum
- Coaching and facilitating the Team
- Shielding the Team from external interference
- Controlling the “inspect and adapt” cycles of Scrum
- Teaching the process, challenging habits and encouraging new thinking and behavior
- Helping people outside the Team understand the process, and understand which interactions with the Team are helpful and which are not

The Scrum Masters are not the managers of the Scrum Teams.

Extreme Programming

Extreme Programming (XP) [Beck, Andres, 2004] is an Agile approach to software development described by certain values, principles, and development practices.

XP embraces five values to guide development:

communication, simplicity, respect, feedback, courage

XP describes a set of principles as additional guidelines:

humanity, economics, mutual benefit, self-similarity, improvement, diversity, reflection, flow, opportunity, redundancy, failure, quality, baby steps, accepted responsibility

XP describes thirteen primary practices:

sit together, whole team, informative workspace, energized work, pair programming, stories, weekly cycle, quarterly cycle, slack, ten-minute build, continuous integration, test first programming, incremental design

Key practices of XP include the following.

- A team of five to ten programmers work at one location with customer representation on-site.
- Development occurs in frequent builds or iterations, which may or may not be releasable, and delivers incremental functionality.
- Requirements are specified as user stories, each a chunk of new functionality the user requires.

© GASQ – Global Association for Software Quality

- Programmers work in pairs, follow strict coding standards, and do their own unit testing. Customers participate in acceptance testing.
- Requirements, architecture, and design emerge over the course of the project.

XP is prescriptive in scope and is typically applied in small teams of less than ten developers, where the customer is integral to the team or readily accessible. In addition, as opposed to other methods, XP describes some strict practices for coding that have been shown to produce extremely high-quality output.

Most of the Agile software development approaches in use today are influenced by XP and its values and principles. For example, Agile Teams following Scrum often incorporate XP practices.

Kanban

Inspired by the movement to Lean Development, Kanban [Anderson, 2010] (the word means "signal" in Japanese) is based exclusively on lean principles [Leffingwell, 2011]. It is the label for a way of scheduling and managing software work that is seeing increasing use in the Agile community.

As defined by the Limited WIP (Work-in-Progress) Society, a software Kanban system has the following characteristics.

- Visualizes some unit of value. This unit of value could be a user story, minimal marketable feature, requirement, or something else. This is different from a taskboard, which generally focuses on visualizing the current tasks.
- Manages the flow of these units of value, through the use of WIP limits.
- Deals with these units of value through the whole system, from when they enter a team's control until they leave it.
- By putting these three properties of a Kanban system together, Kanban allows value to flow through the whole system using WIP limits to create a sustainable pipeline of work.
- Further, the WIP limits provide a mechanism for the Kanban system to demonstrate when there is capacity for new work to be added, thereby creating a pull system.
- Finally, the WIP limits can be adjusted and their effect measured as the Kanban system is continuously improved.

The pull system of Kanban tends to quickly expose impediments, blocking issues, and bottlenecks in the flow (which may result from either a capacity constraint or non-instant availability of a resource). The team can then change their process for the better. As compared to a more traditional, prescriptive approach to change, this visible, adaptive approach can lower resistance and accelerate capability improvement.

Kanban utilizes three instruments [Linz, 2014]:

- Kanban Board: The value chain to be managed is visualized by a Kanban board. Each column shows a station, which is a set of related activities, e.g., development, testing. The items to be produced or tasks to be processed are symbolized by tickets moving from left to right across the board through the stations.
- Work-in-Progress Limit: The amount of parallel active tasks is strictly limited. This is controlled by the maximum number of tickets allowed for a station and/or globally for the board. Whenever a station has free capacity, the worker pulls a ticket from the predecessor station.

© GASQ – Global Association for Software Quality

- Lead Time: Kanban is used to optimize the continuous flow of tasks by minimizing the (average) lead time for the complete value stream.

Kanban features have only some similarities to Scrum. In both frameworks, visualizing the active tasks (e.g., on a public whiteboard) provides transparency of content and progress of tasks. Tasks not yet scheduled are waiting in a backlog and moved onto the Kanban board as soon as there is new space (production capacity) available.

Iterations or sprints are optional in Kanban. The Kanban process allows releasing its deliverables item by item, rather than as part of a release. Timeboxing as a synchronizing mechanism, therefore, is optional, unlike in Scrum which synchronizes all tasks within a sprint. Some consider Kanban a useful integration and completion of Scrum, as well as other methods such as XP, hence the growing popularity of 'hybrid' methods as Scrum/XP and “Scrumban”.

Note:

In this syllabus we will refer to Scrum as the primary Agile software development method.

For training companies: speak about commonalities (common Agile properties) and differences (peculiarities of the methods) among the Agile methods described above.

1.1.3 Agile Team Approach

The cross-functional team has sufficient knowledge about different technologies and software components to be able to produce, from a customer point-of-view, meaningful functionality, either in the scope of a whole product or within a development or a requirement area. Ideally, the cross-functional team is truly cross-functional, not only within development, and having a cross-disciplined team is the first step in achieving this. This team involves representatives from the customer and other business stakeholders who determine product features.

The team size is typically five to nine people. Ideally, the whole team shares the same workspace, as co-location strongly facilitates communication and interaction. When the team is distributed and cannot share the same workspace, current communication technologies help in facilitating face-to-face interaction, even when long distances separate the team members.

The cross-functional team approach is supported through the daily stand-up meetings (daily Scrum) involving all members of the team, where work progress is communicated and any impediments to progress are highlighted.

The whole team is responsible for quality in Agile projects. Business representatives, therefore, will work closely with both developers and testers to ensure that the desired quality levels for the requirements and the product increments are achieved. The whole team is involved in any consultations or meetings in which product features are presented, analyzed or estimated [ISTQB_FA_SYL]. The concept of involving testers, developers, and business representatives in all feature discussions has been called the “Power of Three” [Crispin, Gregory, 2008]. This whole-team approach has a number of advantages including:

- Enhances communication and collaboration within the team
- Enables the various skill sets within the team to be leveraged to the benefit of the project
- Makes quality everyone’s responsibility

These advantages promote more effective and efficient team dynamics.

© GASQ – Global Association for Software Quality

1.2 General Aspects of Agile Software Requirements

1.2.1 Release and Iteration Planning

User stories (stories for short) are the general purpose Agile substitute for what traditionally has been referred to as software requirements [REQB_APPROACH], [REQB_FL_SYL].

User stories are used with Agile software development as the basis for defining the functions a business system must provide, and to facilitate requirements management. It captures the 'who', 'what' and 'why' of a requirement in a simple, concise way, often limited in detail by what can be hand-written on a small paper notecard.

The usual form for a user story is: ***As a <role> I want to <action> so that <benefit expected>***

For example: As a beginning photographer I want to see a user rating displayed so that I can better decide which camera to buy by comparing the rating of my alternatives.

User stories are written by or for the business user. A story is a business user's primary way to influence the functionality of the system being developed. User stories may also be written by developers and other stakeholders to express non-functional requirements (e.g., security, performance, quality).

For Agile lifecycles, two kinds of planning occur, release planning and iteration planning.

Release planning looks ahead to the release of a product, often a few months removed from the start of a project. Release planning defines and re-defines the product backlog, and may involve refining larger user stories (sometimes called epics) into a collection of smaller stories.

Release plans are high-level. In release planning, the product owner establishes and prioritizes the user stories for the release, in collaboration with the Team. Based on these user stories, project and quality risks are identified and high-level effort estimation is performed.

After release planning is done, iteration planning for the first iteration starts. Iteration planning occurs for a single iteration and is focused on determining the sprint backlog.

Iteration plans are low-level. In iteration planning, the Team selects user stories from the prioritized release backlog, elaborates the user stories, completes a risk analysis for the user stories, and estimates the work needed for each user story. If a user story is too vague, the Team can refuse to accept it and use the next user story based on priority. The product owner must answer the Team's questions about each story so the Team can understand what they should implement and how to test each story.

The number of stories selected is based on the established Team velocity (the ability of the Team to complete stories within one iteration) and the estimated size of the selected user stories. After the content of the iteration is finalized, the user stories are broken into tasks that are self-assigned to Team members.

Release plans may change as the project proceeds, including changes to individual user stories in the product backlog.

© GASQ – Global Association for Software Quality

1.2.2 Continuous Integration

Delivery of a product increment requires reliable, working, integrated software at the end of every sprint. Continuous integration addresses this challenge by merging all changes made to the software and integrating all changed components regularly, at least once a day. Configuration management, compilation, software build, deployment, and testing are wrapped into a single, automated, repeatable process. Each integrated build is verified by an automated build test that detects integration errors as quickly as possible. Since developers integrate their work constantly, build constantly, and test constantly, errors in the code are detected more quickly.

An effective continuous integration environment is a basic building block of continuous delivery, the software development discipline where software is built in such a way that it can be released to production at any time. Continuous delivery aims to reduce the cost, time, and risk of delivering incremental changes to users [Humble, Farley, 2010]. Continuous delivery is a basic building block of the Agile iterative model.

1.2.3 Early and Frequent Feedback

Agile projects have short iterations enabling the project Team to receive early and continuous feedback on product quality throughout the development lifecycle. One way to provide rapid feedback is by continuous integration and continuous delivery. When non-iterative development approaches are used, the customer often does not see the product until the project is nearly completed. At this point, it is often too late for the development Team to effectively address any issues the customer may have. By getting frequent customer feedback as the project progresses, Agile Teams can incorporate this new information into the product development process. This maintains a focus on the features with the highest business value, or associated risk, and these are delivered to the customer first. Through frequent feedback, the Agile Team also learns about its own capability. For example:

- How much work can we do in a sprint or iteration?
- What could help us go faster?
- What is preventing us from doing so?

The benefits of early and frequent feedback include:

- Avoiding requirements misunderstandings which may not have been detected until later in the development cycle when they are more expensive to fix
- Clarifying customer feature requests early and regularly throughout development, making it more likely that key features will be available for customer use earlier and the product will better reflect what the customer wants
- Discovering (via continuous integration) quality problems early, making these problems more easily isolated and resolved than when found later in the development cycle
- Providing information to the Agile Team regarding its productivity and ability to deliver

These benefits promote consistent project momentum [ISTQB_FA_SYL]. See also 4.4

1.2.4 Retrospectives

As stated in the twelfth principle of the Agile Manifesto, inspect and adapt is a key practice that all Agile organizations and Scrum Teams must adopt during their work. In Agile development, a

© GASQ – Global Association for Software Quality

retrospective is a meeting held at the end of each iteration to discuss what was successful, what could be improved, and how to incorporate the improvements and retain the successes in future iterations. Retrospectives cover topics such as the process, people, organizations, relationships, and tools. Regularly conducted retrospective meetings, when appropriate follow up activities occur, are critical to self-organization and continual improvement [Derby, Larsen, 2006]. The appropriate follow-ups and the retrospective recordings allow the Team to transfer the achieved experiences and the resulting improving actions to the other Teams and to the other stakeholders, in a continuous improvement and adaptation process.

There are many approaches to holding retrospectives and it is good to use different approaches to keep the retrospectives fresh and Team members engaged. At the start of a retrospective, to set the tone and create an environment of trust, it is important to remind Team members to build a healthy climate. The Scrum Master typically facilitates the retrospective. See also 4.4.2.

In successful retrospectives, Teams focus on problems they can address and create action plans to carry out changes. Focusing on problems within their own sphere of influence forces the Team to avoid continually turning the retrospective into a complaint session. Creating an action plan and acting on it right away in the following sprint gives the Team a sense of continuous improvement.

For training companies: provide some examples of retrospectives, possibly from real cases, and explain how to keep the focus on inspecting and adapting.

1.3 Applicability of Standards and Norms to Agile Development

Standards and process norms that are useful for requirements engineering are listed in [REQB_FL_SYL].

Regarding one of the main questions that arose when Agile development became adopted by an increasing number of organizations, it has been clarified during the last decade that:

- Standard frameworks such as ISO 9000 and CMMi are traditionally thought to be based on large amounts of paperwork. Agile development has proven that it can be effectively applied to the structured frameworks by focusing on what is actually done and how the business goals are achieved.It is important that the auditor knows what Agile is and how it usually works with the structured framework to be audited.
- From the point of view of the other Standards and Norms, listed in [REQB_FL_SYL], they usually apply well also to Agile development when they are describing quality models and general concepts for requirements engineering and systems engineering.
- Some Standards and Norms are not applicable to Agile organizations/project teams when those documents are based on specific development models such as sequential models, often referred to as “traditional development models”.

2 Requirements in Agile Context (90 minutes)

Terms

Agile enterprise, Agile requirement, Agile requirements engineering, business level, business value, commitment, criticality, epic, feature, INVEST, portfolio backlog, priority, program level, program backlog, requirement item, risk, Team level, theme, vision

Learning Objectives for Requirements in Agile Context

2.1 The Requirements in the Agile Enterprise (30 minutes)

AR-2.1.1 Understand the software development and delivery process, Teams and organizational units in the Agile enterprise (K2)

AR-2.1.2 Describe the Team level, the program level and the business level in the Agile Requirements Engineering Workflow (K2)

2.2 Attributes of Agile Requirements (15 minutes)

AR-2.2.1 Understand the main attributes of requirements in Agile software development (K2)

2.3 Quality of Agile Requirements (15 minutes)

AR-2.3.1 Explain the most important quality criteria of requirements in Agile software development (INVEST criteria) (K2)

2.4 Requirements Engineering in Agile Software Development (30 minutes)

AR-2.4.1 Understand the differences between requirements engineering in Agile and traditional software development (K2)

AR-2.4.2 Explain when an Agile requirements engineering approach can be beneficial and when it can be problematic (K2)

© GASQ – Global Association for Software Quality

2.1 The Requirements in the Agile Enterprise

The REQB Foundation Level syllabus [REQB_FL_SYL] defines the concepts of a problem, a solution and a product to introduce the classification of requirements. The full list of terms and definitions can be found in [REQB_GLO].

After giving the definition of requirement as from IEEE 610, [REQB_FL_SYL] states that one of the main purposes of requirements is to express the customer’s needs and expectations regarding the planned solution. Requirements also serve as a foundation for assessment, planning, execution and monitoring of project activity and are often a component of service agreements, orders, or contracts. Requirements not only define what is to be done but they establish the solution boundaries, scope of delivery, plan of delivery, and contractual services.

The above summary remains completely valid working in an Agile environment, where care is required for the product requirements, for both functional and non-functional aspects, and constraints, both business and technical. Again there are external and internal requirements, with different abstraction levels including business requirements, solution/system requirements, and product/component requirements.

The purpose of this chapter is to give an overview about how definition, purpose, classification, and abstraction levels of requirements apply to an Agile development organization.

The definition and the concept of a requirement, as well as most of the standards and regulations that define its characteristics and boundaries, remain completely applicable in an Agile development environment.

The realization of an Agile context, however, is not simple and involves many changes, especially if this occurs within an organization accustomed to a traditional model of development. The Agile environment introduces new terms, new roles, existing roles changes, development practices and requirements management changes, and fundamental differences in the way the team works and communicates.

Several authors in agile-related literature [Cohn, 2010], and several leading figures in agile development verified from many business experiences that a number of organizations have completed an evolutionary transition from traditional software development. In these transitions some common patterns for lean and agile software process success have started to emerge. Starting from this important evidence, REQB proposes what can be called "Agile Requirements Engineering Workflow" to describe the process of Requirements Engineering in the new software development and delivery process mechanisms, the new Teams and their role in the organizational units, and some of the key roles in the agile paradigm. REQB proposes a picture that highlights the practices concerning the requirements within this model.

The figure (see Figure 1) has the advantage of highlighting, through its division into three layers, how the model is applicable to any type of organizational structure, from simple groups of a few people who develop simple applications for the consumer market, to complex projects and multi-product organizations typical of large enterprises.

This picture serves as both the organizational and process model for describing Agile requirements definition, roles, practices, and artifacts, which will be detailed later in this syllabus.

© GASQ – Global Association for Software Quality

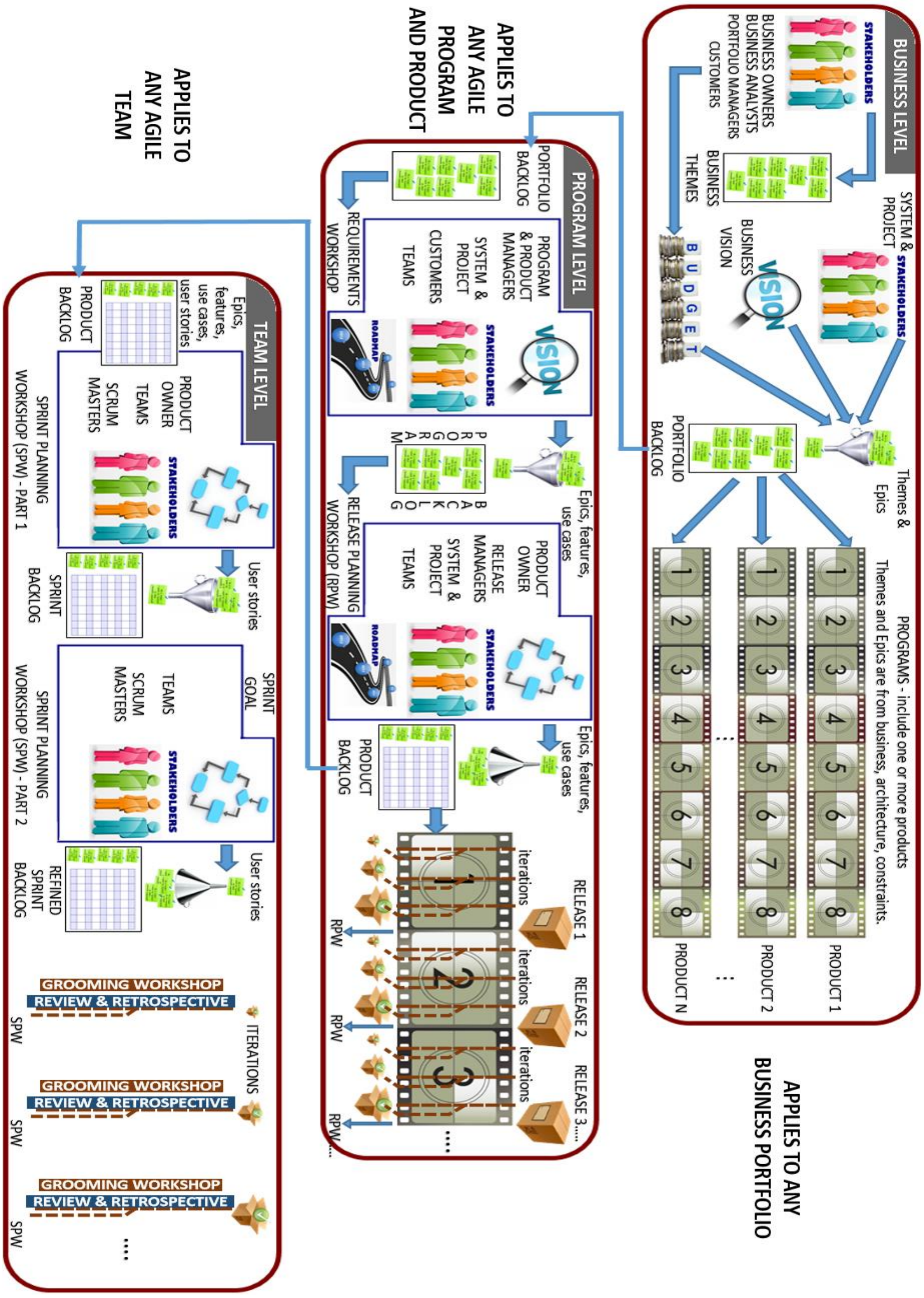


Figure 1: Agile Requirements Engineering Workflow

Figure 1 contains the following elements:

A group of stakeholders	Output of analysis and modeling tasks
Filtering of requirements	Refining of requirements
A sequence of releases	A program/product roadmap
A potentially shippable increment (at the end of an iteration)	A program increment (at the end of a release)

Figure 1 repeats the different abstraction levels of requirements [REQB_APPROACH]. **Figure 2** shows this correspondence, how the requirements are refined and how they progress through the levels of an Agile enterprise while gaining an increasing level of detail.

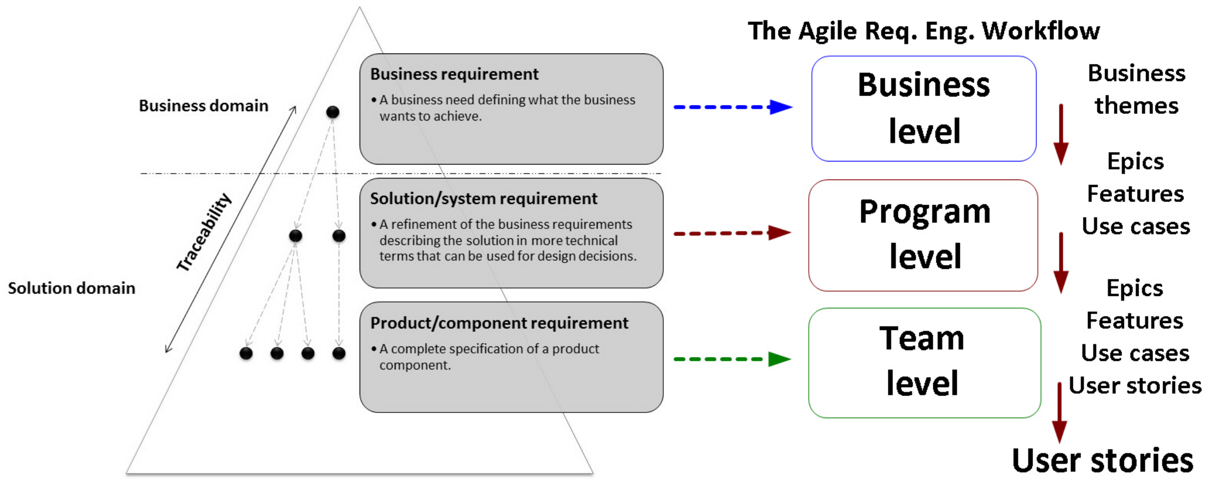

The Agile Req. Eng. Workflow
Business domain
Solution domain
Traceability
Business requirement
• A business need defining what the business wants to achieve.
Solution/system requirement
• A refinement of the business requirements describing the solution in more technical terms that can be used for design decisions.
Product/component requirement
• A complete specification of a product component.
Business level
Program level
Team level
Business themes
Epics
Features
Use cases
Epics
Features
Use cases
User stories
User stories

Figure 2: Abstraction levels of requirements mapping with the Agile Requirements Engineering Workflow

For training companies: provide examples of the applicability of the "Agile Requirements Engineering Workflow" to different sizes of organization.

2.1.1 The Business Level

At the business level, the highest level in the picture, the company's product portfolio is managed. The portfolio management function includes all those roles that deal with managing the investments of the company in accordance with the business analysis, which in turn should be tuned with the needs and prospects of the market and the customers.

The business themes, together with the business vision, can be considered as the primary output of the business analysis process regarding the business needs of the agile enterprise with regards to software development, in other words the business themes at the business level declares the business problems that are input to the requirements engineering process.

The set of business themes establishes the relative investment objectives for the enterprise or business unit and drives the investment priorities and the work necessary for the enterprise to deliver on its chosen business strategy. These themes drive the vision for all programs, and will be expressed as a series of larger, epic-scale initiatives, which will be allocated to various releases. Business themes are key product value propositions that provide marketplace differentiation and competitive advantage.

Epics represent the highest-level expression of a customer need. Epics are development initiatives that are intended to deliver the value of a business theme. Epics are identified, prioritized, estimated, and maintained in the portfolio backlog. Prior to release planning, epics are refined into specific features, which in turn are converted into more detailed user stories for implementation.

Epics may be expressed in user story form, in bullet form, as a sentence or two, in video, in a prototype, or in any form suitable to express the intent of the product initiative. With epics the objective is strategic not specific. In other words, the epic needs only to be described in enough detail to initiate a further discussion about what types of features are implied.

These artifacts are usually championed by portfolio managers, line-of-business owners, product councils, or others who have fiduciary responsibilities to the stakeholders. More details about the requirements engineering flow at the business level are in chapters 4 and 5.

A requirement item has multiple levels, each with an increasing level of detail:

Theme => Epic => Feature => User Story

These span the three levels of a general Agile organization that is developing software products:

- Business level
- Program level
- Team level

All of the requirement items can be conveniently written in user story format; themes, epics and features could be written in other formats too (short descriptions, use cases, and so on).

2.1.2 The Program Level

At the program level, the intermediate level in the picture, the development of larger-scale systems functionality is accomplished via multiple Teams in a synchronized Agile Program Increment. This is a standard cadence of time-boxed (1-to-4 weeks) iterations and milestones that are date-and-quality-fixed, but variable in scope. In larger organizations the program increment should produce releases or potentially shippable increments at frequent, typically fixed, 60-to-120-day intervals.

© GASQ – Global Association for Software Quality

These evaluable increments can be released to the customer, or not, depending on the customer's capacity to absorb new product as well as external events that can influence timing. In smaller organizations or smaller projects, often with a wider audience of customers (e.g., web and mobile applications), releases can be more frequent and iterations shorter.

Within the organization the product management, or possibly program management or business analysis function, is primarily responsible for maintaining a vision of the products, systems, or application in their domain of influence.

The vision answers the big questions about the system, application, or product, including the following:

- Who are the customers?
- What problems does it solve?
- What are the benefits?
- What features does it provide?
- What performance, reliability, and other properties, does it include?
- What platforms, standards, and applications will it support?

The primary content of the vision is a prioritized set of features intended to deliver benefits to the users. In addition, the vision must also contain the various non-functional requirements, such as reliability, performance, usability, compatibility standards, and so on, that are necessary for the system to meet its objectives.

In a manner similar to the product backlog, which primarily contains user stories, the program backlog contains the set of desired and prioritized features that have not yet been implemented. The program backlog should not contain estimates for the features, but should have a priority assignment from both the point of view of the business value and the priority for the customer.

In Agile, and in Scrum in particular, an epic is a large user story. When user stories are in the form of epics, they are too large to be directly implemented and will need to be refined by the Team into smaller stories.

Epics, features and use cases could span several products and releases. The refining process can continue until every User Story has a business value, a priority, acceptance criteria, and an estimate of effort indicating it can be implemented within one iteration. Epics are sometimes treated as use cases or short descriptions (titles) of features.

A theme is a big collection of user stories. Themes are used to express macro functionality, high-level requirements and investment objectives.

2.1.3 The Team Level

At the Team level, the lower level in the picture, Agile Teams of 5 to 9 Team members define, build, and test new features and components aligned with the functional and non-functional requirements which are described in user stories (see section 1.2). The work is divided into tasks and proceeds through a series of iterations and releases.

In the smallest enterprises, there may be only a few such Teams, or just one Team. In some extreme cases, Scrum is adopted by one person who is developing mobile applications. In larger enterprises, groups of Agile Teams work together to support building large pieces of functionality into complete products, features, architectural components, subsystems, and so on. The responsibility for managing the backlog of user stories and other tasks the Team needs to complete belongs to the

© GASQ – Global Association for Software Quality

Team’s Product Owner with support from the Team. As the number of Teams grows, it may be necessary to split the work among multiple product owners.

In its daily work the team is supported by architects, external QA resources, documentation specialists, database specialists, source configuration management (SCM) and build/infrastructure support personnel, internal IT, and whoever else is needed. Sometimes, and this can be beneficial, these supporters are also Team members, often working in more than one Team.

As seen in section 1.2, in an Agile project user stories are the primary objects that carry the customer’s requirements through the value stream—from needs analysis through coding and implementation. The user story often takes the following user-voice form:

As a <role>, I want to <action> so that <benefit (description of the business value)>

With this form, the Team learns to focus on both the user’s role and the value that the new functionality provides. The user story is ready for implementation when the acceptance criteria and tests have been defined and when the priority and size (effort) estimation have been added.

The product backlog is referenced by all Teams involved in the product development. It consists of all the user stories the system and project stakeholders have identified together with the product owner for implementation. A Team always refers to only one product backlog and one product owner who maintains and prioritizes it. In very large-scale projects, the backlog may not define a real product but rather one of its components; for this reason sometimes the backlog is called the Team backlog.

The requirements engineering process in an Agile organization incorporates identifying, prioritizing, elaborating, maintaining, scheduling, implementing, testing, and accepting user stories. At this level the Team works on refining the requirements in the Product Backlog so that each User Story that enters in the next Sprint Backlog can be completely “done” by the end of the iteration.

The three level scheme is applicable not only to large enterprises with large-scale and complex projects, but also to relatively small organizations that develop different product lines, diversify programs through various release roadmaps, and orient products to different categories of markets and customers. There are also cases when the business level and even the program level may not be suitable to describe the development of an Agile project. For example, an internal application with a one-off release and a single internal client would not need the other higher levels.

The next chapters in this syllabus further describe the processes and practices around creating and maintaining user stories.

For training companies: provide examples of the different levels of requirement items, with increasing levels of detail: Theme => Epic => Feature => User Story.

2.2 Attributes of Agile Requirements

The most important attributes, according to [REQB_FL_SYL], for a high-level requirement are the following:

- Commitment
- Priority
- Criticality

© GASQ – Global Association for Software Quality

This is also true for Agile requirements.

Commitment for high level (business) requirements, i.e., themes and epics, is expressed through the typical keywords “shall”, “should”, “may”, “can” [after ISO/IEC/IEEE 29148:2011, previously IEEE 830]. The keywords “should”, “may”, “can”, appear only in the business requirements. As the requirements are refined into features and user stories, the commitment becomes stricter. User stories at the Team level use the format of “I want to” as seen from the point of view of the requirement originator.

The main differences between traditional requirements and Agile requirements include the following:

- When moving from high level requirements to user stories, the Team preserves the perception of the customer view and the benefit that the customer wants to achieve.
- The customer view is never lost when the Agile organization refines the high level requirement from the original theme into the features and stories.

The [REQB_FL_SYL] explains that priority expresses the importance/urgency of a requirement. In Agile the concept remains the same, highlighting the priority as “how much the requirement is requested by a customer or a specific market”. The corresponding attributes are the business value that emphasizes the organization/enterprise view of the requirement in all aspects influencing the business opportunity perspective, and the customer priority that emphasizes the customer view of the urgency to complete the item. Business value is particularly critical if it is difficult to obtain the customer priority.

In Agile requirements, criticality is considered together with all other risk aspects, so not only the evaluation of the risk of the damage that would occur if the requirement were not fulfilled (product risks, risks for business), but also all aspects regarding possible risks during the development (e.g., big changes to existing products, new platforms, third party integration). Criticality is usually referred to as risk in Agile requirements and is refined as the requirement is processed along the model levels down to the Team level.

[ISO/IEC/IEEE 29148:2011, previously IEEE 1233] suggests additional categories of requirements attributes: Identification, Dependency, Source, Rationale, Difficulty and Type. Most of these provide additional information for requirements evaluation that has to be discovered during requirement refinement. The key objective of requirements evaluation is to estimate the business value using the mandatory attributes Commitment, Priority and Criticality. The Agile organization will decide what other attributes are needed in order to better estimate the business value and, later on, the size and the effort for each requirement.

For training companies: explain how the main attributes of high-level requirements are mapped and evolve as the Agile requirements are refined.

2.3 Quality of Agile Requirements

In [REQB_FL_SYL] there is a list of potential issues that can adversely affect the quality of requirements. When comparing this list to the Agile Manifesto it is apparent that applying the stated values and principles should help to minimizing those risks. As with any project, it is still important to formulate and specify the requirements in such as way as to avoid any quality gap.

© GASQ – Global Association for Software Quality

As was discussed, the definition of the requirements in an Agile project goes through a process of progressive refinement crossing the different levels of description of the requirements (i.e., themes, epics, features, user stories). The goal of each refinement is to reach more finite levels of detail that allow the definition of verifiable quality criteria for each requirement. A classification of the quality criteria widely used for Agile requirements is summarized by the acronym **INVEST** [Wake, 2003]:

- Independent - Stories are easier to work with if they are independent. Ideally, they can be scheduled and implemented in any order. One of the characteristics of Agile Methodologies such as Scrum or XP is the ability to move stories around, taking into account their relative priority. This isn't always achievable but when stories are tightly dependent it may make sense to combine them into one story or schedule them in the same or consecutive sprints.

- Negotiable – A good story is negotiable. It is not an explicit contract for features. Instead, the details will be discussed by the product owner/customer representative and the Team members before and even during development. While the story lies in the product backlog, it can be rewritten or even discarded, depending on business, market, and technical issues.
- Valuable - A story needs to be valuable to the customer. This is especially an issue when splitting stories: developers often have an inclination to work on only one layer at a time (and get it "right"), but a full database layer, for example, has little value to the customer if there is no presentation layer. Coming up with technical stories that are really fun to code but bring no value to the end user violates one of the Agile principles, which is to continuously deliver valuable software.
- Estimatable – A good story can be estimated. Being estimatable is partly a function of being negotiated, as it is difficult to estimate a story that is not understood. It is also a function of size; bigger stories are harder to estimate. Finally, it is a function of the Team what is easy to estimate will vary depending on the Team's experience. If a user story's size cannot be estimated, it will never be planned, tasked, and become part of an iteration. Most of the time, when estimation cannot be executed, this is due to the lack of supporting information either in the story description itself or directly from the product owner.
- Sized appropriately – Good stories tend to be small. Stories typically represent at most a few person-weeks of work. (Some Teams restrict them to a few person-days of work.) Above this size, it is difficult to understand what is in the story's scope. Story descriptions can be small too, as tokens promising a future conversation. The details can be elaborated through conversations with the customer, or related stakeholders. Larger stories can be used as starting points, but they must be broken into smaller stories in order to be placed in an iteration backlog.
- Testable – A good story is testable. Actually writing the tests early helps meet this goal. If a story is not testable, it is likely that it is not clearly defined. A story can be "done" only if it has been tested successfully. If one cannot test a story due to lack of information (see "Estimatable" above), the story should not be considered a good candidate to be part of an iteration backlog. This is particularly true for Teams employing test-driven development (TDD).

For training companies: make examples of requirement items (maybe using User stories) that fulfil or not the INVEST criteria.

Quality criteria can be applied not only to a single requirement, but can be used for a requirements specification as well. The major quality criteria for specifications listed in [REQB_FL_SYL] are still valid

© GASQ – Global Association for Software Quality

in an Agile project. What changes is the concept of requirements specification that, in a traditional approach, is a well-structured document. In an Agile project, the requirements specification can take a different form.

Validation and verification still assure the required level of quality for requirements or other products of requirements engineering in an Agile context. Validation and verification benefit from the Agile model's iterations that produce potentially shippable increments. This allows frequent feedback, so validation and verification can be carried out in steps on the delivered product, instead of having a final comparison of the product with its specifications. In Agile projects, common practices such as:

- TDD (Test-Driven Development),
- continuous integration,
- test automation, and
- pair programming

help to continuously verify the quality of the requirements.

For training companies: explain how the above listed practices can be beneficial for verification of the requirements quality.

2.4 Requirements Engineering in Agile Software Development

The general definition of requirements engineering that is provided in [REQB_FL_SYL] is still consistent and valid when working in Agile software development.

When using a sequential development process, there is normally a lengthy, up-front requirements-gathering phase during which the product presumably is fully specified. The idea is that, by thinking longer, harder, and better at the outset of the project, no dark corners will be encountered during the main development phase of the project. If the requirements have been correctly documented, the users will get exactly what they want. That's not necessarily true. At best, users will get exactly what was written down, which may or may not be anything like what they really want.

Agile development focuses on the gradual refinement of a requirement, told in a few words that express the idea, through the collaboration of all parties involved who discuss and define details as more is learned about the product, the users, the competitors, the Team, and so on. In the Foundation syllabus you also find a list of reasons why requirements engineering is neglected. The Agile development methodologies provide a set of practices that should ensure, if properly interpreted and implemented, the requirements engineering process is not neglected in the implementation of the software. These practices include:

- Ongoing communication between the Team and the stakeholders
- Frequent comparison between what has been achieved and what is expected by the customer
- Acceptance of change
- A perpetual cycle of "inspect and adapt" by applying retrospectives
- Assigning responsibilities to the team rather than to individuals

In an Agile project, written documents are no longer the primary tool for requirements specification, but can be a useful support tool to be used only when the Team identifies a real need (such as when you need to transmit knowledge or when customers request such a document). The documents then

© GASQ – Global Association for Software Quality

contain what is actually necessary and will never replace, for the product development, the direct exchange of ideas and face-to-face communication.

2.4.1 Success Factors in Agile Projects

During the last decade Agile methodologies have been applied to all categories of projects and organizations. The Agile software development process has proven most successful when projects and organizations have the following properties:

- The project environment requires development to respond rapidly and effectively to change
- The end goals for the project are not clear at the beginning of the project
- The organization foresees the need for very frequent releases to the end customer, for example mobile applications
- The organization is amenable to change and any resistance can be managed effectively
- The communication and collaboration among the teams is well-managed and effective even with large and distributed teams

2.4.2 Risk Factors in Agile Projects

In most cases Agile requirements engineering does not work if Agile development values and principles are simply disregarded, including inadequate training and poor preparation for key roles. Independently of this, there are some situations where even a well-executed Agile development project can be problematic or less beneficial. These situations include the following:

- There are strong contract constraints from a requirements and solutions point of view. When the margin of negotiation and discussion with the customer is limited, these constraints can undermine the benefits of an Agile project (see section 4.1.3).
- Organizations where compliance to legal and regulatory requirements at fixed milestones is a recurrent goal. This is a problem particularly when there are fixed, stable requirements at fixed dates requiring significant written documentation for compliance.
- Organizations where it is difficult to get the stakeholders together and communicating consistently and reliably. For example, this can occur when Agile is implemented in the development Teams but the customer-related stakeholders are relying on formal requirements documents. This can also happen when the product owner is not adequately representing the end user community (see also section 3.2).
- Organizations with a well-established traditional development process where strong resistance is expected or experienced against a transition to Agile development, or when the transition to Agile development is not sponsored by the top management.
- Projects with distributed development, concentrated competence within a site and development team, a high level of competition between sites and development teams, and poor integration and collaboration among teams.

For training companies: provide examples of situations that are beneficial for an Agile requirements engineering approach and others that are problematic.

© GASQ – Global Association for Software Quality

3 Roles for Agile Requirements Engineering (65 minutes)

Terms

development Team, product owner, project stakeholder, requirements manager, Scrum Master, stakeholder, system stakeholder, Team member

Learning Objectives for Roles of Agile Requirements Engineering

3.1 The Stakeholders (20 minutes)

AR-3.1.1 Identify the stakeholders for Agile requirements engineering and explain their roles (K3)

3.2 The Product Owner (15 minutes)

AR-3.2.1 Understand the role of the Product Owner in Agile requirements engineering (K2)

3.3 The Development Team (15 minutes)

AR-3.3.1 Understand the role of the development Team in Agile requirements engineering and the main contributions (K2)

3.4 The Scrum Master (15 minutes)

AR-3.4.1 Understand the role of the Scrum Master in Agile requirements engineering (K2)

3.1 The Stakeholders

Stakeholders play a key role for requirements engineering in the Agile enterprise. They provide the necessary input for the requirements engineering that occurs throughout the activities of an Agile project [REQB_FL_SYL]:

- Requirements elicitation
- Requirements analysis
- Requirements specification
- Requirements validation and verification
- Requirements traceability
- Configuration and change management
- Quality assurance

According to [Leffingwell, 2011] a project has system stakeholders and project stakeholders.

System stakeholders are those that:

© GASQ – Global Association for Software Quality

- Directly use the system
- Work with the results from those who use the system
- Will be impacted by the deployment and operation of the system

These stakeholders include the following:

- Users and operators, as well as users of reports, data, signals, and other outputs of the system
- Managers, purchasers, and administrators for the users
- Support and help-desk staff
- Developers working on other systems that integrate or interact with the delivered system
- Installation and maintenance professionals
- Any others who interact with or depend on the system

These stakeholders are the primary drivers for the requirements of the system, and, as such, they will be the main subjects of the requirements discovery activities.

In addition to the system stakeholders, it is important to identify everyone who may have substantial, vested interest in the project that is developing the system, in other words the project stakeholders that:

- Want to solve a business problem with a business solution
- Have a vested interest in the budget and schedule
- Have a vested interest in the developed product/system/solution
- Will be involved in marketing, selling, installing, or maintaining the system

These system stakeholders include the project sponsors, business management, project management, portfolio management, executives, financial governance staff, configuration control managers, IT managers and so on.

In order to succeed, the Team must first understand, and then synthesize, their requirements into one cohesive vision. The product owner works to help each stakeholder in finding common ground so they can accept a combined – and from each individual perspective, a potentially compromised – solution. When necessary, the product owner makes decisions for the stakeholders.

System and project stakeholders should be identified when a project starts and are sometimes added as the project progresses. The diverse stakeholders have different degrees of involvement:

- Some stakeholders should be kept informed; they simply need to know the status of the project and be informed of decisions that impact it. They may or may not have input on those decisions but may influence those that do.
- Some stakeholders should be consulted as they have a specific area of expertise that aids in the definition or building of the product. They should be involved in decisions within their area of expertise. These stakeholders include subject matter experts, marketing analysts, architects, and user interface designers.
- Some stakeholders are partners in development or in the process. These may include other product or business owners, other development Teams, business or requirements analysts, and providers of solutions with which the system will interact.
- Some stakeholders are in control of outcomes; they make final decisions for the solution. These may include executives, release managers, business owners, and key customers who will be using the solution.

© GASQ – Global Association for Software Quality

The set of stakeholders should be identified among:

- Who will be directly using the system
- Who will be using the results of those who use the system
- Who will be responsible for supporting the system
- Who is involved with other systems with which developed system will interact
- Who will use the interfaces
- Who can provide guidance on the functionality and system qualities (usability, reliability, performance, and supportability) for the system
- Who needs to be consulted on the scope of the project
- Who has input to the budget and schedule
- Who ultimately manages the business relationship between the Teams and the customer
- Who will determine how and when the system is released to customers
- Who can support or harm the project politically
- Who are the partners who are dependent on the system
- Who cares about the process used to develop the system

For training companies: provide examples of system and project stakeholders, and provide an exercise about how to identify them and their degree of involvement, possibly from real cases.

3.2 The Product Owner

According to [REQB_FL_SYL] a requirements manager is a person with overall responsibility for documenting, analyzing, tracing, prioritizing and coordinating the agreement to the requirements and then controlling changes and communicating requirements to relevant stakeholders. Working in an Agile context, the activities of the requirements manager correspond to the role of the product owner.

In an Agile project, and in particular in Agile requirements engineering, assuming Scrum as the reference method, all activities concerning the management and development of the requirements are the result of intense collaboration and communication among the stakeholders, the product owner and the development Team. When the role of the product owner was introduced in 1.1.2, it was noted that the product owner, through the product backlog (see 6.1.1), manages and tracks all the information about the requirements, prioritizes the requirements, coordinates discussion activities, assesses and reviews the requirements through workshops, and manages changes via continuous feedback.

Regarding requirements engineering, the product owner is responsible for the following:

- Managing and controlling the product backlog
- Representing stakeholders to the Team. Stakeholders are often not available when needed. In most cases their presence is limited to key collaborative events (e.g., workshops and sprint reviews). The product owner acts as the voice of the stakeholders in this case.
- Leading requirements definition
- Setting priorities
- Collaborating closely with the Team
- Evaluating and inspecting the delivered product

© GASQ – Global Association for Software Quality

The product owner should have the skills listed in [REQB_FL_SYL] for a good requirements engineer. The most important skills for a product owner are as follows:

- Communication skills
- Facilitating skills
- Good sense of business
- Good technical knowledge of the domain
- Abililty to make decisions
- Trustworthiness

The most serious hazards for the success of a product owner are as follows:

- Only devoting part time to the project
- Participating as the product owner on multiple teams (depending on the activity level and state of the projects)
- Not adhering to the principles of Agile development Team roles, such as when the line manager of the developers is also serving as the product owner
- Not adequately representing the customer and other stakeholders due to lack of knowledge or isolated areas of expertise (e.g., coming from strong technical IT experience without customer exposure)
- Having more than one product owner for a Team (this is not allowed in Scrum)

Identifying the right person to play this role is fundamental for the overall success of the Agile enterprise. Especially in big companies, with a large number of Teams that might be working in a distributed environment with outsourced and offshored Teams, successfully identifying several product owners who are able to network and cooperate with many stakeholders and distributed Teams determines how successful the Agile project will be.

For training companies: provide some examples of real situations where a product owner correctly interpreted his role for requirements engineering and some others where he was unsuccessful and explain why.

3.3 The Development Team

In Agile development, according to [Leffingwell, 2011] the entire development Team is integrally involved in defining requirements, optimizing requirements and design trade-offs, implementing, testing, integrating into a new baseline, and then ensuring that the fulfilled requirements are delivered to the customers. That is the sole purpose of the Team.

The basic unit of work for the Team is the user story. Each user story is completed within an iteration through a sequence of defining, building and testing actions until it is “done”.

Before starting the development for an iteration, the Team members are involved in all the activities that make up the process of defining the requirements. This consists of workshops that realize the elicitation, analysis and specification of the requirements.

The presence of the development Team in this phase is critical to:

- Understand, thanks to the contribution of stakeholders, the origin, content and rationale of a requirement
- Help identify the constraints related to the requirement

© GASQ – Global Association for Software Quality

- Contribute to the emergence of the non-functional requirements along with the functional requirements
- Provide an initial estimate of the size, in terms of the relative amount of development and testing work and the complexity of each user story
- Help identify the risks associated specifically with the technical aspects, related both to the development and testing

The Team is the main actor in the development of requirements. The presence and weight of members in the events of requirements elicitation, analysis, specification and validation increases as the requirements take shape and user stories are refined to be implemented entirely within an iteration, assuming the INVEST properties.

For training companies: provide some examples of real situations where members of the development team are involved in requirements development (e.g., involvement of expert developers, or testers).

3.4 The Scrum Master

The Scrum Master is responsible for:

- Facilitating the Team's progress toward the goal
- Leading the Team's efforts in continuous improvement
- Enforcing the rules of the Agile process
- Eliminating impediments
- Avoiding changes to the backlog during the sprint execution when the changes will change the sprint goal, will add risk to the achievement of the goal, and are not motivated by strong business reasons

Given the responsibilities of the Scrum Master, it is clear that this role is important in requirements engineering, for example protecting the consistency between the sprint backlog and the sprint goal. The Scrum Master helps the Team develop greater communication skills, both internal and with other stakeholders, supports the Team by addressing the right people to obstacles, and prevents the Team losing sight of its fundamental capability. This is done by pooling the knowledge and experience of all members to refine, assess, and divide each user story into elementary tasks for the implementation and testing.

The same support is given to the product owner, so, from a methodology point of view, the Scrum Master is a key support role for both requirements management and development in an Agile environment.

A Scrum Master should possess the following skills:

- Methodological competence (i.e., practical knowledge of the Scrum method, techniques and tools and how they apply to requirements engineering)
- Moderation and negotiation skills
- Ability to argue and convince
- Language and communication skills

For training companies: provide some examples of real situations where a Scrum Master correctly interprets his role for requirements engineering and some others where he is unsuccessful and explain why.

© GASQ – Global Association for Software Quality

4 Agile Requirements Development (240 minutes)

Terms

3C concept, 4C+R criteria, acceptance criteria, analysis workshop, business value, conditions of satisfaction, contract, customer priority, effort estimation, grooming workshop, implementation story, infrastructure story, model, non-functional requirements, portfolio backlog, relative size and complexity, release planning workshop, requirements elicitation, requirements specification, requirements workshop, retrospective, spike, sprint planning workshop, sprint review, use case, user experience, vision, workshop

Learning Objectives for Agile Requirements Development

4.1 Requirements Elicitation (65 minutes)

AR-4.1.1	Explain requirements elicitation and its common techniques in the Agile enterprise (K2)
AR-4.1.2	Practice the requirements workshop (K3)
AR-4.1.3	Understand the implications of requirements in contracts and the impact on Agile organizations (K2)
AR-4.1.4	Explain how to extract user stories from use cases and non-functional requirements (K2)

4.2 Requirements Analysis (70 minutes)

AR-4.2.1	Practice the release planning workshop, the analysis workshop and the spike (K3)
AR-4.2.2	Explain infrastructure and implementation stories (K2)
AR-4.2.3	Summarize the Agile requirements acceptance (K2)
AR-4.2.4	Practice the sprint planning workshop and the grooming workshop (K3)

4.3 Requirements Specification (55 minutes)

AR-4.3.1	Practice the 3C concept and the process of refining a user story until it fulfills the INVEST criteria (K3)
AR-4.3.2	Assign and apply conditions of satisfaction and acceptance criteria to refined user stories (K3)
AR-4.3.3	Explain why and how to dedicate time for the user experience within a user story (K2)

© GASQ – Global Association for Software Quality

4.4 Requirements Validation and Verification (30 minutes)

AR-4.4.1 Explain validation and verification of the requirements in the sprint review (K2)

AR-4.4.2 Understand how the retrospective applies for inspecting and adapting in requirements development (K2)

4.5 Estimating User Stories (20 minutes)

AR-4.5.1 Estimate the business value, the customer priority, the relative size and complexity, the risk, and the effort of a requirement and explain how to assign these values during requirement refining (K3)

4.1 Requirements Elicitation

According to [REQB_FL_SYL] regarding requirements development, requirements elicitation includes:

- Collecting requirements from stakeholders and, based on the initial scope definition and business needs, formulating business requirements and identifying any limitations and assumptions
- Detailing known high-level requirements
- Excluding unnecessary features

When working in an Agile context, these are the activities that transform a portfolio backlog into a program backlog. This is done by selecting the business themes and refining those to create a set of user stories, or at least epics, which later will be the basis for analysis and specification.

Most of the common techniques for requirements elicitation [REQB_FL_SYL] are used in the Agile context, including the following:

- Requirements workshops
- Interviews and questionnaires
- Self-recording
- Solicitation of information from representatives of the customer on site
- Identification on the basis of existing documents
- Reuse (i.e., reusing the specification from another project)
- Brainstorming
- Field observation and apprenticing
- Use cases

In addition, the following are also used:

- Competitive analysis
- Community participation

The requirements workshop applies all the above techniques in order to share requirements, discuss related outputs and help hidden requirements emerge and become known to all the stakeholders.

Some of the above techniques are used in subsequent workshops as well. Workshops are not meetings and shouldn't be confused as such. According to [Gottesdiener, 2002] a workshop:

- Is facilitated, not run or led by a manager
- Requires pre-work by participants
- Makes decisions based on collaboration or rules-based input
- Is focused on discovery and creation
- Requires active participation by all attendees
- Produces specific deliverables

The following should be addressed when preparing for a workshop:

- Create a shared purpose for the workshop
    - Address the scope and focus on the work products
- Establish the principles for participation
    - Set the basic ground rules – respect, openness, time
    - Determine the decision making process and rules
- In case a stakeholder cannot participate that person must nominate a substitute who can play an active role

Note:

All the workshop outputs and artifacts must be recorded, saved, and made available to everybody involved.

For training companies: provide some examples of the differences between a meeting and a workshop to explain why the second one must always be a collaborative event.

4.1.1 The Requirements Workshop

The requirements workshop brings together all the key stakeholders to forge an agreement with the product owners and the development Team representatives regarding the business value for each identified requirement. The requirements workshop main goals are:

- To identify the requirements for the program starting from the portfolio backlog and looking to the portfolio vision
- To highlight the architectural aspects of the requirements (at a high level)
- To identify the timeframe for the next release from the market point of view and any intermediate dependencies
- To identify the user stories (i.e., start refining the themes and epics, split them into new epics, features, user stories)
- To estimate the business value of the user stories from the stakeholders' point of view
- To identify the conditions of satisfaction for each user story from the stakeholders' point of view

The workshop typically involves the following functions/roles:

- Product owner
- All key stakeholders
- Team representatives (e.g., development, testing, configuration management)
- System architects

© GASQ – Global Association for Software Quality

The first step in a successful workshop is to select the necessary stakeholders, according to the criteria listed in section 3.1.

After the stakeholders have been selected, the next step is to collect the necessary input for supporting the requirements elicitation (e.g., results from interviews and questionnaires, self-recordings, use cases). The workshop can then start based on the following main agenda topics:

- Provide an overview of the portfolio backlog, the vision, the market, and the competitors
- Present results of user interviews, questionnaires and so on
- Start a brainstorming session where the stakeholders identify the items to put in the program (e.g., epics, features, use cases, user stories)
- Make the first divisions of the epics, features, use cases, and possibly the user stories
- Make an estimate of the business value for each identified requirement
- Consolidate and prioritize the resulting requirements

The requirement items here must be identified with a "customer-centric" view, not with a development, testing, architecture or organization view, even if developers, testers, or architects participate. It is a good practice to start with a small number of customer-centric items for the release. Requirements items that are too general or are not valuable from a business point of view will have to be split into more detailed items.

The business value is a key parameter to determine whether or not to implement a user story, when to implement it, and the expected return on investment. Stakeholders assign a business value that reflects their vision of the user story. This value can be guided by any of the following factors:

- New business: every feature that will potentially bring new customers or new markets and, potentially, a fresh flow of money
- Up sell: every feature that will potentially bring money from existing customers and could be sold as an add-on, upgrade or plug-in
- Retainment: every feature that will avoid losing customers and the corresponding loss of revenue
- Cost efficiency: every feature that will allow the company to save money (costs) for material, equipment, people, and time
- Operational efficiency: every feature that will allow the company to save money (costs) given a potential increase in any operation (e.g., installation, configuration, customization)
- Incremental investment capability
- Increased expertise
- Compliance

Neither the effort estimation nor the development complexity are in focus for this workshop and should never influence the business value estimation.

The product owner, or an external coach, acts as the moderator. Stakeholders may ask each other questions and clarify the requirement items without going into too much detail. For each requirement the stakeholders will have to agree on a business value. A common tool to support this activity is planning poker (see section 6.2.1). After reaching a common agreement about the business value, the value is assigned to the related requirement item and that item is then ready to join the product backlog.

The timeboxed duration of the workshop depends on the complexity and size of the project.

© GASQ – Global Association for Software Quality

For training companies: provide some examples of real situations where, during a requirements workshop, a business value is correctly evaluated and some others where the estimation is based on improper parameters such as effort and complexity.

4.1.2 Vision

Another important source that expresses the customer's primary needs is the vision. Since traditional documents may no longer exist to specify intended system behavior, communicating the vision directly to the Agile development Teams is more critical. This potential problem is mitigated or completely removed thanks to the communicative and shared approach of the workshop, where stakeholders sit together.

For each product, the vision must be set anew. A vision is realized by goals to be met by a specific program/project. Goals should express the business objectives of the product and should be S.M.A.R.T. [REQB_FL_SYL]. A vision also summarizes the business risks related to that product.

Independently of the communication methods, the vision communicates the strategic intent for the program/project and should answer these questions:

- Why are we building this product, system, or application?
- What problem will it solve?
- What features and benefits will it provide?
- For whom does it provide these features and benefits?
- What risks will we face for our business?
- What performance, reliability, and scalability must it deliver?
- What platforms, standards, applications, and so on, will it support?

When all stakeholders, business goals and the vision are known, it is possible to start with detailed requirements elicitation.

4.1.3 Contracts

In addition to the deadlines for development and delivery of a product, a contract usually also contains the following:

- The list of prioritized requirements
- A short description of the planned solution
- The acceptance criteria for each requirement
- The list of deliverables (e.g., documentation, code, working software)
- Other needs and expectations such as preferred technology to be used, resource requirements, etc.

Contracts can be a key source for requirements elicitation. This can be constraining in an Agile project.

4.1.3.1 List of Prioritized Requirements

Having a list of prioritized requirements in the contract has pros and cons for both the customer and the supplier. Some of the contracted requirements may never be needed due to changes in the customer needs and priorities during the contract duration (e.g., change of market conditions, new competitors). Usually a lawyer will enforce (via the contract language) the customer's wishes to

© GASQ – Global Association for Software Quality

articulate every possible need and the supplier will strive to meet each anticipated requirement. When an Agile approach is used, the requirements should be articulated in an iterative and evolutionary manner to minimize the risk or spending time and money in developing software for requirements that are not ultimately needed. This may be contradictory to what is contained in a traditional contract.

Ideally, the list of prioritized requirements should be less important and less specific when a contract is used in an Agile development project. This provides advantages to both the customer and the supplier by allowing adaptation to changing needs.

4.1.3.2 Short Description of the Planned Solution

Any description of the planned solution(s) should be avoided in the contracts. It is better to include a summary of the scope, vision, and business motivation of the project or product in the contract preamble, together with the price, and payment model. The supplier should refer to the scope and vision when determining the requirements. This will result in fewer constraints on the contracted solutions and will allow more room for discussing the problems with the customer and reaching consensus.

4.1.3.3 Acceptance Criteria for Each Requirement

Acceptance criteria in a contract can be crucial in outsourced project work. Ambiguity around such areas is a possible source of conflict and litigation. On the other hand a detailed description of the acceptance criteria for each requirement and perhaps a planned solution is a potential pitfall for both the customer and the supplier. When this occurs, the customer will give up the flexibility to change the requirements and the supplier may have constraints that will decrease software quality and communication with the customer.

In an Agile project, it is better to define a framework for acceptance in the contract such as the following:

- When using an iterative approach, for each iteration, acceptance can be based on conformance to a prior agreed-on set of acceptance tests.
- When using Scrum, acceptance can be based on conformance to the Definition of Done that includes the use of test automation.

4.1.3.4 List of Deliverables

Contracted outsourced work is often assumed to involve low amounts of transparency and trust. It is also common for the working software to be delivered after a long period of time during which the development team has “gone quiet” providing little visibility into their activities. Contracts are often written to require a conventional quality management plan or deliverables list that defines a long checklist of documentation to be provided. The concept here is that more documentation will ensure the customer is getting what they want.

A more practical approach is to limit the list of deliverables to the ones that are truly necessary, concentrating on those that provide real value to the customer. This allows the supplier to concentrate on delivering more value and optimizing the time for requirements engineering within the project.

4.1.3.5 Other Needs and Expectations

Most of these needs and expectations are the non-functional requirements and should be treated as any other requirement.

The product owner should play an active part in any negotiation regarding the contract that can lead to an improvement in the contract model. Everything in the contract about the requirements should be treated so as to bring benefit to the customer first, through a greater overall flexibility, while allowing the supplier to focus on the value of the delivered software. This is not always possible because the discussion on the contract is often limited. Where there is room to negotiate, the role of the product owner becomes important to support the business functions that deal with the customer.

For training companies: propose examples of an evolution of traditional contracts towards contracts that support effective requirements engineering in the Agile context.

4.1.4 Use Cases

Use cases are an alternative method for expressing the functionality of a system. If an Agile organization is good with use cases, there may be no reason to change. However, use cases are intended to be much larger in scope than is common for a user story. Scrum Teams typically work more effectively with units of work that are smaller than a typical use case. In general a single use case would not meet the INVEST criteria, in particular the “sized appropriately” property, because it describes more functionality than can be implemented in a single sprint.

Refined user stories, i.e., those ready for inclusion in a sprint, are very granular and often do not give the designers a context to work from—when is the user doing this, what is the context of their operation, and what is their larger goal at this moment?[Cockburn, 2007]

Use cases scan be helpful in an Agile development project by playing the role of (or replacing) system epics, features and large stories. These use cases will need to be refined into “sized appropriately” user stories. In this way, use cases can provide the context and the user stories provide the final backlog items to be implemented.

[Cockburn, 2007] suggests some tips for applying use cases in Agile projects:

- Keep them lightweight—no design details, GUI specifications, and so on
- Don’t treat them as fixed requirements. Like user stories, they are merely statements of intended system behavior
- Don’t worry about maintaining them; they are primarily thinking tools
- Model them informally—use whiteboards, lightweight tools, and so on

When use cases are used, user stories can be extracted by:

- Implementing a subset of the scenarios
    - Primary path first, then exception paths
- Implementing a subset of a scenario’s actions
    - Key actions first, then secondary
- Separating out common sub-goals from a set of use cases
    - Implement each sub-goal as a building block

For training companies: provide examples of use cases and user stories that can be extracted from these use cases.

4.1.5 Non-Functional Requirements

Non-functional requirements (NFRs) describe the non-functional quality attributes of the whole system or its specific components or functions. Non-functional requirements can describe different aspects of the solution. According to ISO/IEC 25000 (previously ISO 9126), specific types of non-functional quality characteristics are the following:

- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Each of these has sub-characteristics.

Non-functional requirements can be considered to be constraints on the functionality as they may limit the design of the solutions to ensure the expected level of performance, to guarantee compatibility with specific platforms, to allow the users to be able to use the functionality within a limited time, to ensure an expected number of transactions can be processed in a limited time, and so on.

One NFR may apply to one or more stories and a story may be constrained by one or more NFRs.

An NFR itself can be written in the form of a user story. In general, an NFR that is written as a simple statement can be described in user story format. This can be a beneficial way to help orient the development team to the requirement and to facilitate estimation and implementation.

The following is an example of an NFR in the traditional format:

"All open source software used in the project must be approved by the customer"

User story format 1: *As a customer representative, I want to approve any open source software used in the project, so we don't have license exposure.*

This first user story adds some information regarding the reason for this requirement. The story could also be stated as:

User story format 2: *As a customer representative, I want to approve any open source software used in the project, so that the reuse of already tested and reliable open source software is maximized.*

Depending on the user story motivation, the development team can identify different tasks for the same NFR. In the first case the selection of the open source software will be based on the license policy; in the second case it will be based on reusing reliable software.

An NFR may be persistent, i.e., it remains in place for a long time across different releases or applies to similar independent products. Often these requirements, especially as they relate to performance, are verified through automated testing and will be reverified throughout the life of the product to ensure the NFR is consistently met. In some cases NFRs are incorporated into the Definition of Done and become applicable to each story or for an entire iteration.

For training companies: provide examples of NFRs and user stories that can be extracted from these NFRs.

4.2 Requirements Analysis

In requirements development, requirements analysis includes [REQB_FL_SYL]:

- Elaborating business requirements after elicitation into system/solution requirements and down to product/component requirements, based on organizational assets
- Resolving conflicts between requirements with support from the stakeholders
- Establishing the boundaries for the solution
- Formulating agreed requirements with scope definition
- Developing models of the business solution

When working in an Agile context, requirements analysis means all activities that transform a program backlog into a product backlog. These activities include selecting the requirements items that are in the form of high level epics, features, use cases, and big user stories, and refining them to create the prioritized user stories in the product backlog. Once refined, these stories are ready to be estimated. These activities take place during the release planning workshop (refer to Figure 2 chapter 2), the analysis workshops and the spikes.

4.2.1 The Release Planning Workshop

The release planning workshop typically involves the following functions/roles:

- Product owner
- Workshop facilitator (can be the product owner)
- A selection of stakeholders who can actively contribute by providing information for user story refinement and risk estimation. They provide a first verification and validation to ensure that the requirements are correctly understood and approved. Participants could include:
    - Stakeholders representing the customers (e.g., product management)
    - Representatives of other projects with dependencies
- Teams representatives (development, testing, configuration management)
- System architect
- Scrum Masters

The main goals of the workshop are:

- To present and agree on a ranking of all items (epics, features, use cases, user stories) that are candidates for the product backlog, based on the customer priority and business value. The items were prepared by the product owner as a result of the requirements workshop.
- To identify and plan a detailed analysis workshop if necessary
- To complete the user story refinement for the items in ranked order
- To assign a risk value to all items in the product backlog
- To estimate the size of the user stories, from the Team and product owner point of view
- To verify the release timeboxing in order to identify the Scrum Team members and build the Teams
- To decide the focus for the sprint planning workshop of the first sprint

Employing a traditional planning approach and trying to capture all requirements in detail in the product backlog is error-prone when dealing with innovative products. It is also undesirable in an

© GASQ – Global Association for Software Quality

Agile context where the product's detailed functionality is discovered during the development through early and frequent customer and user feedback. This does not mean that the Team cannot make an informed investment decision at an early point in time in an Agile project. The launch date and project budget frequently are decided before the first sprint.

Based on this first draft of the product backlog prepared by the product owner, the participants can assess if further analysis is required in order to proceed with the next steps: user story refinement, risk value estimation and user story size estimation. If further analysis is required, the release planning workshop is postponed and an analysis workshop is scheduled. The release planning workshop resumes when the analysis results are good enough to proceed.

After completing the refinement of the user stories (see 4.3.1.1) with the highest priority and business value in the product backlog, the team should proceed as follows:

- Estimate the risk value. The release planning workshop is the right place to estimate the risk because the necessary people are attending. The risk value includes all risks related with the analyzed requirement, including risk for the overall product, risk for the business (see 4.1.2), and technical risk. The value is a number that summarize the results from the risk analysis for a requirement. A risk value can be used as additional information to prioritize the user stories for the sprints with the goal being to address the higher risk stories early.
- Estimate the effort or complexity of the stories. In this workshop an estimation of complexity is highly recommended to provide a preliminary rough idea of how complex it will be to implement the story. A more refined estimate of effort should be created during the sprint planning meeting. It is important to remember that these are size estimates, not business value estimates. The estimate is usually made in a measurement called story points. Planning poker is often used to determine this estimate. Planning poker is discussed in section 6.2.1.

Once again, the user stories that are too big have to be split into more detailed ones. This can be done splitting the story based on different criteria as explained in 4.3.1.1.

It is important to note that the criteria for identifying the contents of the first sprint are not just numbers. While it is logical to prioritize the product backlog based on the user stories that have the maximum business value, highest priority and risk values, that is not the only way to define the content of a sprint. Good judgment must be used and stories must be evaluated to determine what preconditions are necessary. It may also makes sense to move some lower value stories to the top of the product backlog because of project opportunities or needs (e.g., infrastructure and implementation stories as discussed in section 4.2.2). This logical prioritization can be done without compromising the sprint goal.

4.2.2 Infrastructure and Implementation Stories

The requirements analysis provides an additional step toward the completion of the product backlog. Two important categories of requirements must be considered and described in user story format: the implementation stories (or capability stories) and the infrastructure stories.

An implementation story is an additional item, conveniently written in the form of a user story, which may be required when new capabilities must be added in order to be able to implement feature items. For example an implementation story would be needed for a new driver that is required to allow the software of several user stories that runs on a specific platform to be compatible with an

© GASQ – Global Association for Software Quality

additional platform. Usually this is an item on the product backlog that could provide low or even null business value, but is so important that it must be included in order to be able to ship the business value of related user stories.

An infrastructure story is an item in the product backlog that is required to create the necessary infrastructure in order for the sprint to achieve the status of "done". This is usually a high priority item that must be managed within a sprint like any other user story. An example of an infrastructure story would be one that describes setting up the continuous integration environment or one that explans how the test equipment or the test automation framework must be set up. It can be written in the form of a user story or not.

The introduction of these items into the product backlog can be done during the release planning meeting as soon as they become known. This is useful because they are estimated as any other user story (but may have no business value).

At the end of the release planning, the product backlog includes a variety of items:

- New customer features ("enable all users to place a book in a shopping cart")
- Engineering improvement goals ("rework the transaction processing module to make it scalable")
- Exploratory or research work ("investigate solutions for speeding up credit card validation")
- Known defects ("diagnose and fix the order processing script errors")

The refinement of the requirements can be complex to complete in several cases, in particular when:

- The product content is innovative
- Customers are not able to define the requirement unambiguously
- Several customers have a different and incompatible view of the same requirement
- Conditions of satisfaction cannot be agreed to by the various stakeholders
- The INVEST level of refinement of a requirement cannot be reached due to poor knowledge of some functional or technological aspects, or because interfaces to other products, systems or infrastructures need to be investigated

In all these cases it is necessary for further analysis to reach the appropriate level of understanding of the requirement, both from the point of view of business value and from a technical point of view. This analysis should make it possible to complete the refinement of the requirements, clarifying any aspects that are still not clear.

There are two primary methods for conducting this analysis:

- The analysis workshop
- The spike

In both cases the goal of the analysis is that, at the end of the activity, each analyzed requirement will acquire the properties expressed by the "4C+R" criteria:

- Correct: accurate, reflects actual needs
- Clear: only one interpretation, unambiguous
- Consistent: no conflict with other requirements
- Complete: no missing elements
- Relevant: necessary to achieve business goals and objectives

© GASQ – Global Association for Software Quality

4.2.3 The Analysis Workshop

The analysis workshop is necessary in the following cases:

- The Team may need to conduct more in depth analysis of the implied behavior, because the customers and the business stakeholders did not provide enough detail about this behavior.
- Several customers view the same requirement with substantial differences in functional content and priority. It is necessary to understand if these differences are compatible and possibly make decisions that will maximize the business value and minimize the product risk.
- It is not completely clear how the requirement will impact existing products or subsystems.
- It is unclear if the requirement implies significant impact to the project infrastructure. This must be clarified to provide a reliable estimate.

The main goals of the analysis workshop are to review all the following aspects of a requirement:

- Product functionality: user stories, scenarios or use cases and related analysis models
- Quality attributes: constraints on functionality (e.g., usability, maintainability, reliability, security, performance, safety)
- External interfaces: integrating / combining two subsystems or system components
- Technology or tools: proving technical feasibility to meet product requirements, incorporating a new tool into project development to satisfy design or implementation constraints
- To define the conditions of satisfaction for the requirements (user stories) if still missing

The key aim of the analysis workshop is not to complete an exhaustive, preliminary analysis, as in the sequential models, but rather to clarify the requirements to the point where they can be accurately estimated during the release planning workshop. For higher priority items, it is important to refine them to achieve the INVEST characteristics, to estimate the effort and to plan the individual tasks that will be targeted for the next sprint.

It is important to understand during the analysis workshop if any specific analysis/design tasks will have to be planned during the next sprints.

Once again communication and collaboration are key factors; during the workshop it is important to draw schemes, diagrams and whatever is necessary on flipcharts and whiteboards, making sure they are clearly visible to everyone. Individual notebooks should not be used unless strictly necessary.

The output of the workshop must always be recorded, using readable photos of the room walls or whatever else is useful, to be stored for future access. The participants do not leave the room until all the outputs have been recorded.

4.2.4 The Spike

Referring to [Leffingwell, 2010], spikes, another invention of XP, are a special type of story that is used to drive out risk and uncertainty in a user story or other project facet. Spikes may be used for a number of reasons:

- The Team may not have knowledge of a new domain, and spikes may be used for basic research to familiarize the Team with a new technology or domain.
- The story may contain significant technical risk, and the Team may have to do some research or prototyping to gain confidence in a technological approach that will allow them to commit the user story to some future timebox.

© GASQ – Global Association for Software Quality

- The story may contain significant functional risk, in that while the intent of the story may be understood, it is not clear how the system needs to interact with the user to achieve the implied benefit.
- It is clear that a specific interface or a specific layer will suffer significant impact, and the Team needs to reach a reasonable knowledge about this impact.

Spikes may be classified as technical spikes or functional spikes.

Technical spikes may be used in the following cases:

- When analysis workshops are used to research various technical approaches in the solution domain, technical spikes are used to quickly check the same identified technical approaches.
- When analysis workshops are used to determine a build vs. buy decision, a technical spike may be used to quickly check and validate the buy options.
- Technical spikes can be used to evaluate the potential performance or load impact of a new user story.
- Technical spikes can be used for any reason when the Team needs to develop a more confident understanding of a desired approach before committing new functionality to a timebox.

Functional spikes may be used in the following cases:

- Whenever there is significant uncertainty as to how a user might interact with the system, a functional spike may be used to allow the team to do more research.
- Functional spikes are often best executed through some level of prototyping, whether it be user interface mockups, wireframes, page flows, or whatever technique is best suited to get feedback from the customer or stakeholders (see also 4.3.3).

For training companies: provide practical examples, possibly from real cases, when analysis workshops rather than spikes should be done.

4.2.5 Requirements Acceptance

According to [REQB_FL_SYL], requirements acceptance (also called agreeing on requirements or signoff) is the next element of requirements analysis. It is a formal agreement that the content and scope of the requirements are accurate and complete.

In an Agile context:

- This is partially achieved in the requirements workshop, where agreement on the high-level requirements (business requirements) has been obtained.
- This is partially achieved during the release planning workshop, where the requirements have been accepted at different levels of the solution development as formal agreement for inclusion in the product backlog.
- This is partially achieved by the agreement on the detailed requirements (solution/system requirements and component/function requirements) with the development Team in the sprint planning workshop (see 4.2.5.1).

In an Agile project, the purpose of requirements acceptance is not ensuring the requirements are stable and that any changes are managed via formal change requests. Requirements acceptance in Agile means to agree on the product vision, the business requirements, the content of the product backlog and the content of each sprint. Changes are always possible to the product backlog content,

© GASQ – Global Association for Software Quality

prioritization, and estimations with recursive workshops. Only the content of the sprint in progress is protected from changes that might change the sprint goal.

The risk of misunderstanding between the customer and the supplier regarding the project's scope is minimized through the feedback opportunities at each sprint review.

At the end of every workshop the output is the (new) agreement among the participant stakeholders.

4.2.5.1 The Sprint Planning Workshop

Each sprint begins with a timeboxed event called the sprint planning workshop. The Scrum Team collaborates to select and understand the work to be done in the upcoming sprint. The entire Team attends the sprint planning meeting. Working from the ordered product backlog, the product owner and the development Team members discuss each item and come to a shared understanding of that item and what is required to complete it consistent with the current Definition of Done.

In Scrum, the sprint planning meeting is described as having two parts:

Part 1) Determine what work will be completed in the sprint.

- The product owner informs the Team(s) about the product vision and prioritized product backlog, and defines with the Team the sprint goal.
- The product owner and the Team(s) discuss the proposal for the Definition of Done and, after any agreed change, commit to it.
- The Team estimates the effort for each story proposed by the product owner to be part of the sprint.
- The Team checks the estimated effort, the sprint duration and the Team availability. Stories are added or removed from the set selected for the sprint backlog in order to meet the team velocity.
- The Team(s) assign themselves stories until each story is assigned.
- The Team confirms the sprint schedule.

Invited participants for Part 1:

- All Team members
- Scrum Master
- Product owner (owner of the meeting)
- Other stakeholders are optional

Part 2) Determine how the work will be accomplished.

- Team members define tasks for each user story in the sprint.
- Task owners estimate the effort for each task. If a task effort is bigger than one day it should be split into smaller tasks.
- The Team verifies that every piece of work is taken into account (based on the Definition of Done), including additional analysis and design, meetings, learning new technologies, infrastructure work.
- If the Team believes that the sprint backlog is too large, backlog items are removed in agreement with the product owner.

© GASQ – Global Association for Software Quality

- If the Team believes that the sprint backlog is too small, the most important backlog items are moved from the product backlog to the sprint backlog based on agreement with the product owner. The sprint goal is adjusted if necessary.
- The Team commits to the sprint goal.

Invited participants for Part 2:

- All Team members
- Scrum Master (owner of the meeting)
- Product owner (has to be reachable for questions)

For estimations in part 1 and 2 planning poker can be used (see 6.2.1).

For training companies: provide an exercise for a simplified application of the sprint planning workshop.

4.2.6 The Grooming Workshop

The beginning of each sprint has a product backlog that collects "at the top" the more refined items, those that will be included in the sprint and inserted into the sprint backlog.

At the end of each sprint, the items that are "done" and are accepted as completed, are then marked with a status of "done". Those not completed or not approved in the sprint review must be rescheduled, following the procedure already described for creating the sprint backlog for the next sprint.

During a sprint, it is necessary to reiterate the activities carried out before the sprint planning workshop, which consist of refining a subset of the stories in the product backlog (using INVEST). These refined stories will form the foundation for the discussion of the content of the new sprint in the next sprint planning workshop.

This activity is carried out in the grooming workshop, which incorporates part of the structure of the release planning workshop, although participation will be restricted to those who can help in refining the part of the product backlog to be targeted for the next sprint as identified by the product owner.

Participation in this grooming workshop is required from:

- Product owner
- Scrum Master
- Development Team
- Stakeholders involved in the items under discussion

The methods of preparation and development will be substantially similar to those seen for the release planning workshops, analysis workshops, and spikes.

For training companies: explain, with real examples, why the grooming workshop is one of the key collaborative events in Scrum.

© GASQ – Global Association for Software Quality

4.2.7 Solution Modeling in Agile

Solution modeling as described in [REQB_FL_SYL] is still valid in the Agile context. Model types include context diagrams, scenarios, data models, event-response tables, state diagrams and business rules.

In an Agile project, the modeling is used for conversation, design, and creating understanding and agreement between the Team and the stakeholders, before using it for documenting. In this sense model diagrams are hand-drawn on a flip chart or white board before being drawn with a tool for inclusion in a document. The document may be needed to deliver business value, but should not be used if it does not provide value and is just part of the traditional documentation that is created.

4.3 Requirements Specification

In requirements development, requirements specification includes producing the requirements specification documents.

In an Agile project, the team should "favor working software over comprehensive documentation," but that does not mean that documentation is not needed. The team should strive to do the simplest thing that can possibly work, and when something should be documented, it makes sense to do it.

When working in an Agile context, requirements specification does not mean to produce a comprehensive paper document, but rather record, preserve and make available all evidence and artifacts created during conversations among the participants in the workshops. Paper documents are welcome limited to what the Teams consider necessary to allow implementation. Other paper documents are welcome when needed to document what has been done (not what will be done), for example for customer documentation or to respond to obligations. Requirements specification is a continuous process that starts with requirements workshops and proceeds before and during sprints.

In case a customer provides the supplier with a requirements specification, this is a good source of information for all the requirements engineering processes. Even if this document is provided, the Agile requirements development process should function as described in this section.

4.3.1 Requirement Specification through Collaborative User Story Creation and Refinement

When people convert from a traditional methodology to Agile, they want to know how to get all the information needed to understand a requirement and the user's need. They also want to know how to specify, in a sufficiently complete manner, the solution that is to be implemented when the requirement is expressed in only a few words in a user story. It often seems that the Agile requirements have considerably less information than traditional requirements.

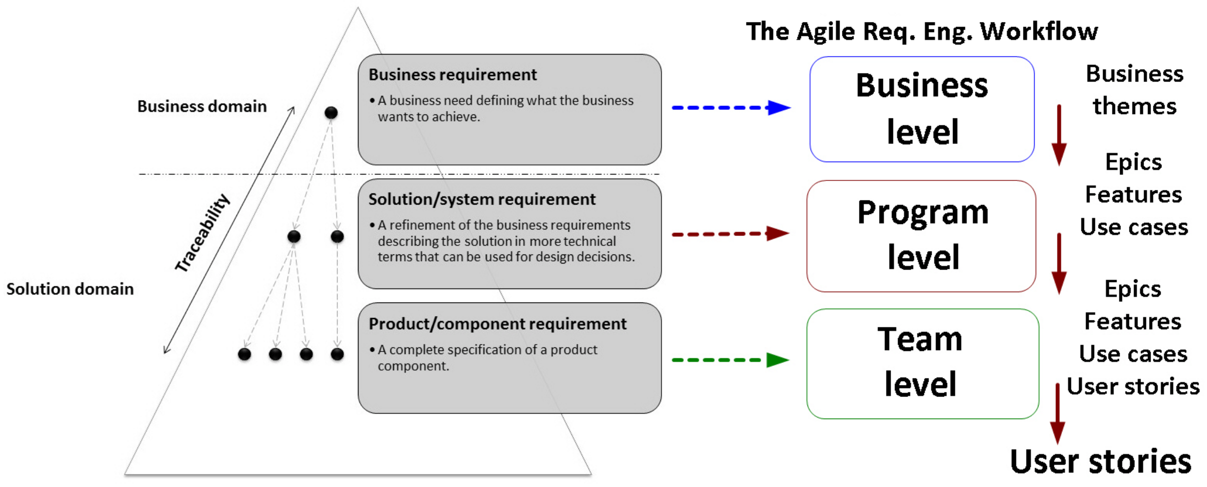
In reality, according to the 3C concept [Jeffries, 2001], a user story is made of three elements:

- Card: The card is the physical media describing the story and its benefits (Who, What and Why). It also identifies the requirement, its criticality (business value, priority, risk), and the expected effort for making it done. The information has to be accurate enough to be used in the product backlog.

© GASQ – Global Association for Software Quality

- Conversation: The conversation element of the user story allows for continuous dialogue around the related requirement, to define how it will be implemented, tested, how the related software will be used, how the expected non-functional aspects will be guaranteed, what technical solutions will be used, and which constraints must be managed. Independent from how the conversation results are documented and recorded, all this information is the fundamental component of the requirements specification.
- Confirmation: The acceptance criteria, discussed in the conversation, are used to confirm that the story is done. These acceptance criteria may span multiple user stories. Both positive and negative tests should be used to cover the criteria. During confirmation, various participants play the role of a tester. These can include developers as well as specialists focused on performance, security, interoperability, and other quality characteristics. To confirm a story as done, the defined acceptance criteria must be tested and shown to be satisfied.

Agile Teams vary in terms of how they document user stories. Whatever the approach, the documentation should be concise, sufficient, and necessary.


Business domain
Solution domain
Traceability
Business requirement
• A business need defining what the business wants to achieve.
Solution/system requirement
• A refinement of the business requirements describing the solution in more technical terms that can be used for design decisions.
Product/component requirement
• A complete specification of a product component.
The Agile Req. Eng. Workflow
Business level
Program level
Team level
Business themes
Epics
Features
Use cases
Epics
Features
Use cases
User stories
User stories

In chapter 2 we saw (**Figure 2**, shown again above) how the requirements are entered into a product backlog at the end of a refinement process that starts from the definition of a portfolio. The refinement process includes understanding the general business themes, selecting the appropriate parameters (as shown in 2.1.2) to determine which features a product release must contain, and then breaking the requirements down into user stories of various sizes.

The user stories that reach this maximum level of refinement (summarized in the acronym INVEST) are moved to the upper part of the product backlog and become target content for the first sprint planned.

Requirement items that do not meet all the INVEST criteria, especially for the size, must be further refined. This process may take more than one step of refinement because the resulting user stories may themselves be epics, to be refined further.

Examples of refinement of requirement items can be found in [Cohn, 2004], [Cohn, 2010] chapter 13, and [Cohn, 2012].

During requirements elicitation and requirements analysis, the process of refinement is based on conversations between the product owner, stakeholders representing clients either directly or indirectly, the development Teams and other stakeholders representing business functions in support of the development Teams. This refinement of user stories usually takes place in the context of workshops involving all those who have the knowledge to contribute. The aim of these workshops is to reach a level of refinement for a sufficient number of user stories to enable the next step.

All that emerges from these conversations starts to form the requirements specifications. These specifications are not necessarily written in a document on paper or electronically, but will be collected in any format deemed suitable. Suitable formats provide the following capabilities:

- Storage for what has been said and agreed, so that everybody can easily find the reference and understand the content
- Ability to change these specifications each time the conditions require it
- Traceability from the requirement to the actual need

During the workshops the output of the discussions on the requirements can be collected in several ways:

- On large sheets of paper hanging on the walls, where, for example, models are drawn, which are stored at the end of the workshop or simply photographed to be easily distributed
- On patterns drawn on a whiteboard, with a printed copy or a photo is taken as a record
- On sticky notes displayed on a wall to represent each user story
- With a tool that enables the sharing of ideas with a distributed Team
- With other tools or media that encourage the collaboration and sharing of information

It is important to note a fundamental difference between a requirements specification document and the techniques of requirements specification used in an Agile project.

The specification document tries to collect everything needed for understanding the requirements in a single document, which will be submitted for review and approval. With this approach the quality of the review of the document is not under the control of the author and a subjective interpretation is likely to occur when there is no sharing of the background information.

In an Agile project, the requirements specification is solidly based on this sharing, the details are formed as part of the conversations during the workshops and the step by step details are identified and shared as needed for the next step.

For training companies: make examples, possibly from real cases, about requirements documentation in an Agile environment (wallpapers, wikis, etc.), and show how a user story is made by the 3C elements.

4.3.1.1 Common Patterns for Splitting a User Story

The following is a list of common patterns that may indicate a story should be split into multiple stories:

1. Workflow steps – A complete workflow is composed of multiple steps. The Team may identify specific steps that a user takes to accomplish a specific workflow, and then make a refined user story for each step.
2. Business rule variations - At first glance, some stories seem fairly simple. However, sometimes the business rules are more complex or extensive than the first glance revealed. In this case, it may be useful to break the story into several stories to handle the business rule complexity.

© GASQ – Global Association for Software Quality

3. Major effort - Sometimes a story can be split into several parts where most of the effort will go toward implementing the first part. In many cases, processing infrastructure should be built to support the first story; adding more functionality should be relatively trivial later.
4. Simple/Complex - When the Team is discussing a story and the story seems to be getting larger and larger ("What about this? Have you considered that?"), stop and ask, "What's the simplest version that can possibly work?" Capture that simple version as its own story, and then break out all the variations and complexities into their own stories.
5. Variations in data - Data variations and data sources are another source of scope and complexity. Stories may be isolated to handle only certain types of data.
6. Data entry methods - Sometimes complexity is in the user interface rather than the functionality itself. In that case, it may make sense to split the story to build it with the simplest possible UI, and then build the richer UI later.
7. Defer system qualities - Sometimes, the initial implementation is not difficult, and the major part of the effort is in making it fast or reliable or more precise or more scalable. However, the Team can learn a lot from the base implementation, and it should have some value to a user who would not otherwise be able to use the functionality at all. In this case, it may make sense to break the story into successive "ilities."
8. Operations - Words such as "manage" or "control" are a giveaway that the story covers multiple operations, which can offer a natural way to split the story.
9. Use case scenarios - If use cases have been developed to represent complex user-to-system or system-to-system interactions, then the story can often be split according to the individual scenarios of the use case.
10. Break out a spike - In some cases, a story may be too large or overly complex, or perhaps the implementation is poorly understood. In that case, the Team should build a technical or functional spike to figure it out and then split the story based on that result. (See 4.2.4)

4.3.2 Assign Conditions of Satisfaction to Refined User Stories

In the specification of requirements in Agile development, a key input is to assign to each user story the conditions of satisfaction of the requirement. This anticipates and enables the definition of the acceptance criteria, providing the conditions according to which the user story can be considered satisfied from the point of view of the user.

During the discussion of the conditions of satisfaction, often the criteria that must not be satisfied, because they are not required, will also be analyzed. Keeping track of these issues will reduce the risk of implementing parts of functionality that are not needed (waste), and will help the team focus on what is actually required. This is a benefit that is not usually seen when the specification is written in a formal document.

It is important to have the conditions of satisfaction already defined to support a user story in reaching its INVEST properties. These conditions should be discussed and collected during a requirements workshop, a release planning workshop, and sometimes during a grooming workshop.

For training companies: provide examples of assigning conditions of satisfaction, required and not required. Later on this will be referenced in showing the difference between a condition of satisfaction and acceptance criteria (4.3.4).

4.3.3 Dedicate Time for the User Experience

A common problem in Agile software development is determining how to incorporate the visual design of the product into the rapid iteration structure. When teams attempt to resolve complex user interactions while trying to code and test the system at the same time, they can often end up "churning" through many iterations. This creates waste because the result of this process is delayed feedback loops that complicate productivity and introduce delays in delivering the value. [Leffingwell, 2011]

A number of practices for addressing these design elements leverage the iterations to drive out uncertainty and risk through rapid feedback:

- Provide paper prototypes, simple HTML prototypes, for the user to review
- Conduct user experience story spikes that can be used to develop alternative user interfaces and test them through actual or representative users
- Create a small, centralized user experience design authority who provides the basic design standards and preliminary mock-ups for each UI. In this case sprint Teams will have Team-based UI implementation experts for the implementation that are sourced from the centralized team. The centralized authority will provide HTML designs, CSS style sheets, brand control, mock-ups, usability guidelines, and other artifacts that provide conceptual integrity of the UX across the entire solution.

Paper prototypes, outputs of user experience story spikes and centralized user experience design artifacts become part of the requirements specification.

4.3.4 Assign Acceptance Criteria

User story acceptance criteria form the basis for the functional tests that are intended to assure the implementation of each new user story delivers the intended behavior and ultimately satisfies the needs of the users and business owners. Generally, the following is true [Leffingwell, 2011] regarding acceptance criteria:

- They are written in the language of the business domain.
- They are developed in a conversation between the developers, testers, and product owner. The conversation output is recorded and saved.
- Although anyone can write tests, the product owner, as business owner/customer proxy, is the primary owner of the acceptance tests.
- They are black-box tests in that they verify only that the outputs of the system meet the conditions of satisfaction, without concern for how the result is achieved.
- Acceptance tests are implemented during the course of the iteration in which the story itself is implemented.

This means that new acceptance tests are developed for every new story. If a story does not pass its tests, the Teams get no credit for the story. The story is carried over into the next iteration, where the code or the test, or both, are reworked until the tests pass.

For training companies: provide examples of assigning acceptance criteria. Showing the difference between a condition of satisfaction (4.3.2) and acceptance criteria.

© GASQ – Global Association for Software Quality

4.4 Requirements Validation and Verification

In requirements development, requirements validation and verification includes checking the quality of the requirements and the requirements specification documents.

According to [REQB_FL_SYL], defects resulting from low quality requirements are more expensive to fix in later phases of the project than other types of defects. In addition, the later defects are detected, the higher is the cost to fix them. Therefore the use of verification ("are we producing the product correctly") and validation ("are we producing the right product") of the requirements are key activities.

In Agile software development, requirements validation and verification mainly rely on the continuous feedback mechanism that is intrinsic within sprints, because in Agile when using Scrum, each sprint is required to deliver a potentially shippable product increment. Thinking in lean and Agile terms it is important to take a systemic and holistic view. The stories (requirements), implementation (code), and validation (acceptance tests, unit tests, and others) are not separate activities but rather a continuous refinement of a much deeper understanding.

The software must be routinely regression tested to ensure it continues to work. This is done by persisting and automating (wherever possible) acceptance tests. Many Teams write the story acceptance test first, before developing the code. This is called acceptance test-driven development. The acceptance tests serve to record the decisions made in the conversation so that the Team understands the specifics of the behavior the user story card represents. The code follows logically thereafter. [ISTQB_FA_SYL]

4.4.1 The Sprint Review

At the end of each sprint, the Team has produced a coded, tested and usable piece of software that is demonstrated at the sprint review meeting. During this meeting, the Scrum Team shows what they accomplished during the sprint. Typically this takes the form of a live demo of the new features to the product owner, the Teams, the Scrum Master, interested stakeholders and customer representatives. The sprint review is intentionally informal, often forbidding the use of slide presentations.

During the sprint review, the project is assessed against the sprint goal that was determined during the sprint planning meeting. Ideally, the Team has completed each product backlog item brought into the sprint, but it is more important that they achieve the overall goal of the sprint. For each story in the sprint backlog, the Definition of Done is checked in order to verify that the story has achieved Done status, before the meeting starts.

The output of the sprint review must be always recorded. The participants do not leave the room until all the outputs have been recorded.

4.4.2 The Retrospective

As introduced in section 1.2.4, the retrospective is the inspect and adapt event that happens at the end of each sprint. After the sprint review, when the Team knows if the sprint has been successfully done or not, i.e., approved or not by the product owner and the stakeholders, the Teams meet together to look at what went well and what needed improvement during the sprint, try to

© GASQ – Global Association for Software Quality

understand the reasons for good and bad things, and try to enforce good practices and correct mistakes.

A reasonable part of this meeting should be dedicated to requirements, for example:

- Looking at the sprint result, did we implement the requirement as expected by the stakeholders?
- If not, what did we do wrong, and in particular did we refine the requirement correctly?
- If not, when and how did we refine it incorrectly, and in particular did we analyze the requirement correctly?
- If not, why and where did we analyze incorrectly, and in particular did we understand the requirement correctly?

This helps the team determine the strong and weak points of the requirements development. The goal is to identify corrective actions and to put them in place with an urgency that corresponds to the severity of the problem. If necessary, new items in the product backlog are raised.

Invited participants to the retrospective include:

- All Team members
- Scrum Master (owner of the meeting)
- Product owner (optional, but recommended if the sprint goal was not reached)

In case more than one Team was involved, it is useful to have the first part of the retrospective as a single team meeting, then a second part where all Teams can sit together (maybe involving just Team representatives) to discuss common issues with the product owner.

For training companies: provide some examples about how to identify problems related to poor requirements engineering during retrospectives and how to put in place corrective actions.

4.5 Estimating User Stories

In the previous sections several values have been discussed for use in estimating a user story. This section retraces the characteristics of each of these estimations, including when to hold it and when to re-evaluate based on refinement of the story.

A user story, or more generally a requirement, can be estimated with respect to:

- Business value
- Customer priority
- Complexity
- Relative size
- Risk
- Effort

4.5.1 Business Value

The business value is the first criterion for requirement evaluation. Business stakeholders evaluate the advantage to the business that will be gained by the implementation of the requirement. This evaluation will be based on the stakeholder's own vision and visibility into the business.

The business value is estimated during the requirements workshop, according to the evaluation criteria agreed among the participants. The method of assessment should be a relative, not absolute, value, so if two requirements are evaluated respectively with business value of 5 and 10, it is understood that the second has double the value of the first, without giving a precise amount to the two numbers. It is sometimes useful to start the evaluation with a requirement that the stakeholders consider to be of great value – for example, the value BIG using the voting technique of planning poker (6.2.1) – then evaluate the subsequent requirements in relation to the first and to those that follow.

When a requirement is refined, the business value should be distributed among all items derived by refinement. There are two possible situations:

- The requirement has been refined into several user stories that are carried out in any order, but there is no business value until the last user story is "Done". In this case, the entire business value is assigned to the last user story, while the others will get a business value of 0.
- The requirement has been refined into several user stories; each of them delivers business value once completed, perhaps in a certain sequence. In this case, the initial business value must be spread among the refined user stories, keeping the sum as the initial value.

Real cases are usually a mix of these two. When a requirement implies the realization of its refinements in sequence for providing value, often the term "saga" indicates this sequence.

4.5.2 Customer Priority

The customer priority tells us how important a requirement is for the customer. The customer representatives will evaluate the advantage to the business upon the implementation of the requirement, according to their own vision and visibility, and will translate this into a priority.

The customer priority is estimated during the requirements workshop, according to the evaluation criteria agreed among the participants. As with the business value, the method of assessment should produce a relative, not absolute, value.

In case several customer representatives will be present and a different priority evaluation is expected from them, it is better to move this estimation before the workshop, keep the different estimations and avoid this discussion here. The product owner will weight the customer priorities, together with the business colleagues if possible, in order to find a key to interpreting the different priorities from the point of view of business opportunities and pitfalls.

The user story priority is not necessarily the same as the customer priority. The user story priority can be a combination of the business value, the customer priority, and the risk, depending on the strategy.

When a requirement is refined, the customer priority is the same for all items derived by refinement.

4.5.3 Relative Size and Complexity

An estimation of the relative size is usually done during the release planning meeting. That estimate is used to provide an initial understanding of the development effort that will be required. The method of assessment should be a relative, not absolute, value.

Estimating the "size" of a user story usually includes an effort estimation from the development Team. It is sometimes easier to discuss the size in terms of complexity so non-developer stakeholders gain a better understanding of the estimate. Voting techniques such as planning poker provide the opportunity for each estimator to explain their estimate to the others.

When a requirement is refined, the size value should be distributed among all items derived by refinement, each of them having a size value. The sum of the size value of the user stories derived from refinement could be different from the size value of the original item before refinement. This is because after refinement a new estimation is possible.

4.5.4 Risk

A risk estimation is usually done during the release planning meeting and sometimes during the sprint planning meeting. A risk estimation during the requirements workshop is usually premature because there is not enough information available about the requirement. When the risk estimation is being made, it is important that the right stakeholders are present. The product owner has to understand when to conduct a risk estimation depending on the audience to involve. Each user story has to have its own risk value. When a story is refined, the refined story must have an updated risk value.

Risk identification and analysis is part of requirements management (section 5.1).

The method of assessment can be a relative value or an absolute value, according to criteria defined prior to the workshop.

4.5.5 Effort

Estimating effort is more complex than estimating the other values. The sprint content depends on this estimation and the ability of the team to achieve the sprint goal depends on the reliability of the effort numbers. Agile helps this estimation in the following ways:

- Estimations are made for short durations of effort. User stories have to be implemented within one sprint; the individual tasks that make up a user story typically require hours rather than days.
- At any time, the progress for each activity is known. The Team quickly adapts to more accurate estimates.
- Statistical data about the velocity of the development Team is available in a short time. This data can be used as a reference for subsequent estimates. The velocity is expressed in story points of completed tasks per unit of time. Once the story points for a set of stories is known, that information is used to project the time required to complete the tasks.
- The mechanism of inspect and adapt with the retrospective allows the Team to refine its effort estimations.

The estimation unit is Ideal Developer Days (hours for very short iterations). In first sprints it will be based on the time expected to complete the items for the sprint. Later on it will be based on the Team velocity and the value in story points.

For training companies: provide examples of effort estimations, based on Ideal Developer days/hours, and show examples of stories that must be refined due to the effort being too high. Show how the Team velocity can be used after some sprints.

© GASQ – Global Association for Software Quality

5 Agile Requirements Management (70 minutes)

Terms

change, change management, change request, configuration management, continued collaboration, continuous feedback, metric, product risk, project risk, quality assurance, quality control, risk, risk management, risk mitigation, traceability

Learning Objectives for Agile Requirements Management

5.1 Project and Risk Management (15 minutes)

AR-5.1.1 Understand the common problems, project risks and product risks related to Agile requirements engineering (K2)

5.2 Traceability of Agile Requirements (15 minutes)

AR-5.2.1 Understand how traceability of requirements can be achieved in Agile projects and when specific traceability practices should be added (K2)

5.3 Configuration and Change Management (15 minutes)

AR-5.3.1 Explain how change management works in Agile projects (K2)

5.4 Quality Assurance (25 minutes)

AR-5.4.1 Recall the factors that influence the quality of the requirements engineering process and products (K1)

AR-5.4.2 Understand the criteria for using metrics in Agile requirements engineering. (K2)

According to [REQB_FL_SYL], requirements management is mainly a collection of managing and supporting activities that are needed to assure that the requirements development process is executed properly during the product lifecycle.

5.1 Project and Risk Management

A weak requirements engineering process is an intrinsic risk for any development model as this can result in requirements that are not precise, are contradictory, do not fulfill the criteria, and do not satisfy stakeholders' needs [REQB_FL_SYL].

© GASQ – Global Association for Software Quality

Agile development, Scrum in particular, organizes the activities before, during and after the development through a series of events and practices, focused on requirements development. Without a systematic approach to requirements development an Agile project cannot even start.

The most common problems related to Agile requirements engineering are the following:

- Unclear business vision and a lack of business goals to be achieved within the solution
- Poor involvement from the stakeholders during requirements elicitation and refinement
- Contradictory requirements from different customers where priority cannot be established on the basis of clear business objectives
- Insufficient requirements refinement
- Lack of analysis, often resulting in imprecise estimates of the impact of requirements and their changes on the other areas of the product under development
- Size or effort estimates that are too far from reality (often resulting from unclear requirements) resulting in sprint goals not being achieved

The potential for the problems listed above can be perceived as a risk. To deal with those risks – and other risks as well – a risk management process is necessary.

In an Agile project, risks are identified, estimated and assigned to the related user stories during the workshops and then managed during the sprints.

[REQB_FL_SYL] classifies two main types of risk: product risk and project risk.

Project risks are the risks that surround the project's capability to deliver its objectives. Risks that are specifically pertinent in Agile organization include:

- Lack of confidence with Agile methods and practices
- Resistance of the organization to the transition to Agile from traditional approach
- Poor experience of the product owner and the Scrum Master
- Cross-functional areas not covered adequately by the Team
- Teams allocated only part time
- Continuous integration not implemented
- Neglected Definition of Done
- Sprint timeboxes extended
- Weak retrospective practices including skipping retrospectives

Product risks are potential failure areas (adverse future events or hazards) in the software or system. These are risks to the quality of the product. Risks that are specifically related with Agile practices include:

- Poor test automation and insufficient maintenance of testing environment
- Refactoring never done
- No participation from the stakeholders or participation by the wrong stakeholders in the sprint review

Risk mitigation is usually managed as any other activity in Agile. It can be translated into additional user stories to be added to the product backlog, items to be included into the impediment backlog, systematic checks to be included in the Definition of Done, practices to be enforced during the sprint and so on. In this way any risk can be managed and checked through the Scrum mechanisms.

© GASQ – Global Association for Software Quality

5.2 Traceability of Agile Requirements

Traceability is the ability to relate aspects of project artifacts to one another. A requirements traceability matrix is the artifact that is often created to record these relationships. It starts with the individual requirements and traces them through any analysis model, architecture model, design model, source code, or test cases that are maintained.

The benefit of having such a matrix is that it makes it easier to perform an impact analysis pertaining to a changed requirement because the potentially affected aspects of the system are identified.

In a requirements traceability matrix, the traceability information is often manually maintained. The creation of a traceability matrix, as any other artifact or document in Agile development, should be justified by providing real added value. Real value is determined by balancing the following:

- Benefits
- Risk of not having it considering the complexity of the context
- Total cost of ownership
- Contribution to traceability that artifacts and tools already provide

Regarding this last bullet it is important to note that in an Agile process, requirements are refined in place, not translated. In fact, for purposes of forward and backward traceability, the backlog owner literally builds out a requirements matrix from the beginning, and develops it in a way that keeps traceability intact.

As the project proceeds:

- Traceability does not need to be maintained as a separate activity, because each system feature is being built out from general description into detailed implementation, and approved as "complete" when it meets pre-agreed acceptance criteria.
- If requirements need to change to remove functionality, new stories are introduced to override behavior that is not needed any more.

It is important to verify if the standard artifacts from the Agile project can fulfill the traceability needs for the product and the supplier's organization. If so, additional traceability matrices are not needed and would not add value.

In simple situations, which many Agile Teams find themselves in, traceability matrices are highly overrated because the total cost of ownership to maintain such matrices, even using specific tools to do so, far outweigh the benefits. Making the stakeholders aware of the real costs and benefits allows the team to make the right decisions regarding the amount of documentation needed to track traceability.

There are cases when maintaining a traceability matrix makes sense. Some of these include:

- Automated tool support exists. Some development tools will automatically provide traceability as a side effect of normal development activities. This may not supply full traceability, but a lot of the traceability work can and should be automated. With automated tooling the total cost of ownership of traceability drops, increasing the chance that it will provide real value to the Team effort.
- Complex domains. The need for traceability is much greater to help deal with the complexity of the domain.
- Large Teams or geographically distributed Teams. Although Team size is typically motivated by greater complexity – domain complexity, technical complexity, or organizational

© GASQ – Global Association for Software Quality

complexity – there is often a greater need for traceability as well. When a large Team is involved it becomes difficult for even the most experienced Team members to comprehend the detailed nuances of the solution as the knowledge may be spread across several people. The traceability information provides the insight necessary to effectively assess the impact of proposed changes. Note that large Team size and geographic distribution have a tendency to go hand-in-hand.

- Regulatory compliance. Traceability may be required by regulations. If so, it must be provided.

5.3 Configuration and Change Management

Configuration management is a key practice in Agile development. It is a basic requirement for continuous integration (see also 1.2.2) and test automation. Specifically regarding requirements engineering, configuration management is applied to:

- Requirements as backlog artifacts and individual artifacts
- Workshops results
- Requirements analysis output (see 4.2.3)
- Models

Change management in an Agile project is not substantially different from the traditional process with regard to activities to manage changes in requirements [REQB_FL_SYL].

Changes to release plans may be triggered by internal or external factors. Internal factors include delivery capabilities, velocity, and technical issues. External factors include the discovery of new markets and opportunities, new competitors, or business threats that may change release objectives and/or target dates. In addition, iteration plans may change during an iteration. For example, a particular user story might prove more complex than expected during estimation [ISTQB_FA_SYL].

Change management in Agile includes the following activities:

- Identifying or discovering a need for change
- Putting it in the parking lot until it is analyzed
- Estimating any change in terms of the business value, then moving it as a new item in the product backlog, linking it with the original requirement
- During the next sprint planning or next grooming workshop, evaluating the change (including assessing the risk, cost, and effort of the change) in order to make a decision whether and when to implement the change
- Planning the change (if the change has been approved for implementation)
- Implementing the change and following it through as part of the sprint goal

The big difference in the process of change management in Agile development compared to traditional development lies in the fact that the mechanism of the change request is not mandatory and the Change Control Board is not necessary. An Agile project is designed to accommodate changes through roles and events that have already been defined. When a change request arrives for the product owner, it is handled by applying the usual assessment of the value and priority of the new request with respect to what is already in the backlog. In case the change affects a requirement that is already done or a new requirement, the positioning of the change in the backlog could imply the infeasibility of another backlog item being implemented in time for the release. A change of a

© GASQ – Global Association for Software Quality

requirement that is not yet done may result in the simple replacement or a change in the estimates, but this could result in an impact to the existing backlog items.

For training companies: show with examples the difference between the change management process in traditional projects (with change request and Change Control Board) and the change management in an Agile context.

5.4 Quality Assurance

[REQB_FL_SYL] lists some factors that can affect the desired quality level of the product. Such factors are valid for Agile projects. They include:

- The customer's quality criteria or expectations related to the QA process
- The type of product that is being produced (e.g., the complexity or target audience of the product – a product intended for a small audience might have a lower level of some quality attributes than a mass market product)
- The environment in which the product is developed (e.g., the technical environment might limit the ability to achieve the desired level of quality attributes)
- Domain (e.g., complexity of the business domain, the level of innovation, the frequency of changes in the business)
- Legal, safety and environmental factors (e.g., regulations resulting in a higher level of quality to be achieved)
- Time and cost pressures that reduce the ability to execute the requirements engineering processes

The approach to quality control in an Agile environment has some fundamental differences compared with traditional development. The traditional approach assumes that the product is realized with successive phases, requiring handover of work from one phase to another and that the release of the final product can take place only at the end of the handover sequence. When products in the chain are modified, there is a potential disruption that must be handled with extreme caution. In addition, the quality control of everything that is produced during requirements engineering must rely on pre-established quality criteria for documents based on templates and formal reviews at the various stages of progress. Until the quality of the final product can be verified, it is necessary to rely on the quality of the documentation.

In an Agile project there are no specific standards, templates, or reviews of documents to lay the groundwork toward a certain level of expected quality. The Agile project bases quality on the fundamental practices of Agile development that originate from the 12 agile principles (1.1.1):

- Continuous collaboration between the development Team (including testers) and stakeholders, based on direct conversation rather than on the exchange of documents, defining the conditions of satisfaction of the requirements and acceptance criteria of incremental deliveries to ensure the testability of the requirements.
- Continuous integration, during iterations, of continuous deliveries by the development Team. The use of automated testing and the use of test-driven development aims to maintain a high quality level in each incremental delivery.
- Continuous feedback, at the end and possibly also during the iteration, by the product owner and other stakeholders, as the main mechanism to verify and validate requirements and the quality of the emerging product.

© GASQ – Global Association for Software Quality

With these practices in place, in order to ensure the required level of quality, in the Agile context verification and validation are planned and executed from the beginning of the project through to the end.

An essential collaborative event that drives improvement in Agile is the retrospective described in section 4.4.2. Continuous improvement and adaptation is managed through appropriate follow-ups to retrospective findings and the retrospective recordings. These allow the Team to transfer the achieved experiences and the resulting improved actions to the other Teams and to the other stakeholders.

Agile development conceives the use of metrics for the sole purpose of keeping all stakeholders, in addition to the development Team, informed about the progress of the release and of each sprint through burndown charts. Metrics are also tracked for any impediments that are encountered and defects that are found and resolved.

The metrics are never a pure instrument of control defined and standardized by a process or a quality plan. In Agile the metrics are provided to all or part of stakeholders in order to improve the transparency and efficiency of the work. The adopted metrics may then be changed according to needs, the status of release, and the actual usefulness recognized.

In (section 2.3) the INVEST properties were introduced as the key checklist for common quality attributes for requirements to be used during user story refinement. In parallel, the following questions should also be asked when evaluating the quality of requirements during a quality checkpoint:

- Is the requirement unambiguous?
- Is the requirement feasible?
- Is the requirement understood?
- Is the requirement originator identifiable?
- Is the requirement testable?

© GASQ – Global Association for Software Quality

6 Tools and Artifacts in Agile Requirements Engineering (80 minutes)

Terms

emerging requirements, parking lot, planning poker, Pomodoro Technique®, release burndown chart, timeboxing

Learning Objectives for Tools and Artifacts in Agile Requirements Engineering

6.1 Artifacts Supporting Agile Requirements Engineering (35 minutes)

AR-6.1.1	Use the product backlog, the release burndown chart to provide general progress view, several metrics and support activities related to requirements engineering (K3)
AR-6.1.2	Explain how to support emerging requirements with the parking lot (K2)

6.2 Tools Supporting Agile Requirements Engineering (45 minutes)

AR-6.2.1	Use planning poker to vote in an Agile environment (K3)
AR-6.2.2	Explain timeboxing and the use of timers (K2)
AR-6.2.3	Understand the purpose of tools supporting communication and explain important factors to be considered when selecting a tool for Agile requirements engineering (K2)

6.1 Artifacts Supporting Agile Requirements Engineering

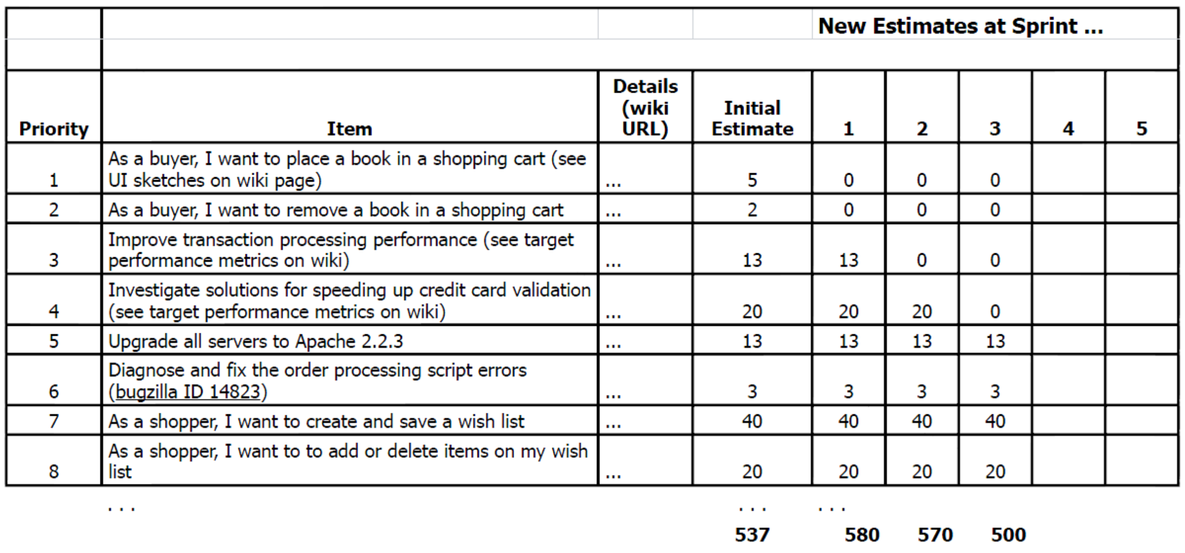
6.1.1 The Product Backlog

As seen in previous chapters, the first basic step in Scrum for the product owner is to articulate a prioritized list of features called the product backlog. According to [ScrumGuide, 2013] the product backlog is an ordered list of everything that might be needed in the product and is the single source of requirements for any changes to be made to the product. The product owner is responsible for the product backlog, including its content, availability, and ordering. This ordering is based on the following criteria:

- Identify the most important stakeholder values
- Minimize risk by putting higher risks earlier
- Minimize waste of funds and resources
- Determing the best financial performance
- Make the incremental model effective by delivering value in steps

© GASQ – Global Association for Software Quality

The product backlog evolves as the product and the environment in which it will be used evolves. The product backlog constantly changes to identify what the product needs in order to be appropriate, competitive, and useful.



				New Estimates at Sprint ...				
Priority	**Item**	**Details (wiki URL)**	**Initial Estimate**	**1**	**2**	**3**	**4**	**5**
1	As a buyer, I want to place a book in a shopping cart (see UI sketches on wiki page)	...	5	0	0	0		
2	As a buyer, I want to remove a book in a shopping cart	...	2	0	0	0		
3	Improve transaction processing performance (see target performance metrics on wiki)	...	13	13	0	0		
4	Investigate solutions for speeding up credit card validation (see target performance metrics on wiki)	...	20	20	20	0		
5	Upgrade all servers to Apache 2.2.3	...	13	13	13	13		
6	Diagnose and fix the order processing script errors (bugzilla ID 14823)	...	3	3	3	3		
7	As a shopper, I want to create and save a wish list	...	40	40	40	40		
8	As a shopper, I want to to add or delete items on my wish list	...	20	20	20	20		
	. . .		. . .	. . .				
			537	**580**	**570**	**500**		

Figure 3: Example of product backlog [ScrumPrimer 2.0]

At the beginning of each sprint, the remaining work for each item in the backlog is estimated and written in the column related to that sprint. The completed (done) items have zero estimation for all remaining sprints, in progress items have an estimate for the remaining work, and those items that are not yet started will show the total estimate which may be revised based on further refinement and last analysis activities.

Each row in the backlog can include additional information, for example:

- The epic / feature / use case (requirement item) that generated the user story / item of the backlog
- Links to artifacts, documents, workshop output, whatever provides additional details to the item. A very common tool used to store, organize and exchange data is a simple Wiki page.
- Business value and the priority
- Risk value
- Notes
- Status

The product backlog provides a means to have an immediate overview of the product progress that is always evolving, changing and under refinement. The information related to each item allows the team to track changes, specification, and analysis output.

Only a single product backlog exists per owner; this means the product owner is required to make prioritization decisions across the entire spectrum of the requirements.

A spreadsheet is often used to manage the product backlog. Teams can make their own spreadsheets and customize them as needed, download a spreadsheet from the Internet or use one of the product backlog features available as part of many commercial packages.

© GASQ – Global Association for Software Quality

6.1.2 The Release Burndown Chart

A release burndown chart, owned by the product owner, shows the amount of work remaining at the start of each iteration.

The horizontal axis of the release burndown chart shows the sprints; the vertical left axis shows the amount of work remaining. In the product backlog each backlog item contains the amount of work remaining in story points. These values are updated continuously. The burndown chart plots this backlog across time starting from the amount of work at the beginning of the release down to zero when the release is completed.

The vertical right axis shows the business value added by each sprint. This metric provides useful indications about the remaining work, and also provides good indication about where the business value is going to be distributed during the sprints and how much of the requirements value is done.

Several metrics are incorporated in the release burndown chart, or can be easily produced if necessary. For example:

- Business value done at each sprint (compared with the sprint goal)
- Release business value change at each sprint
- Deviation of sprint value (in story points)

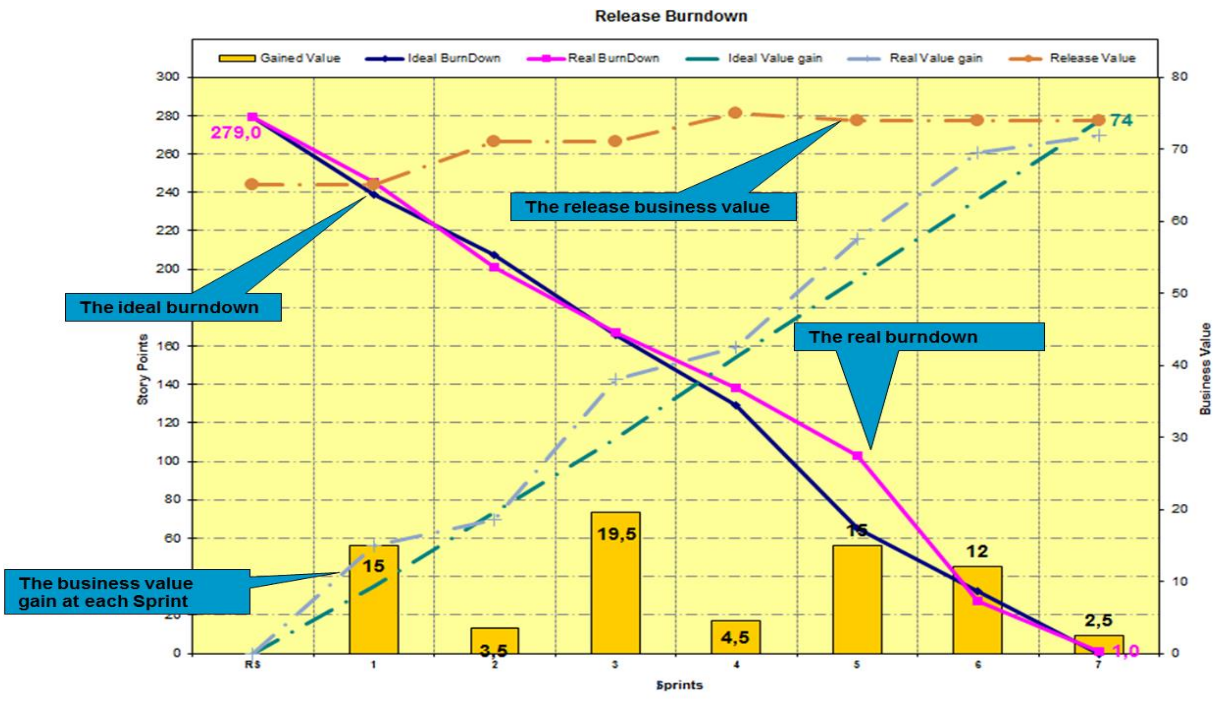


Figure 4: Example of Release Burndown Chart

For training companies: show with practical examples how to use the product backlog, the release burndown chart to provide general progress view, several metrics and support activities related to requirements engineering.

6.1.3 Emerging Requirements: The Parking Lot

There are certain requirements, functional and non-functional, that emerge during the development of a product. This can occur for many reasons, for example:

- During a sprint review, looking at what has been done, the customer or his representative becomes aware of a non- functional aspect that must be considered and implemented
- During the construction of a user story, the Team realizes an interdependence from another feature that requires an additional feature
- During the construction of a user story, the Team realizes an interdependence from another feature that implies a new non-functional requirement, such as usability

These emerging requirements, once approved for the product development, must be managed as any other: entered into the product backlog, refined through analysis by the Team, and finally included into a sprint. The identification of emerging requirements can be effectively supported by the parking lot, which, as the name suggests, is a temporary repository where the emerging requirements are "parked" waiting to be discussed, approved and incorporated into the product backlog.

The parking lot helps to track important items, ideas and issues that need more investigation or would be better discussed at a later time.

An example of this would be an idea that comes up during a daily Scrum that needs a discussion with another Team to understand if it helps, or a new idea during a product backlog refinement that the product owner wants to consider. The Team can add this to a parking lot. The parking lot is a simple facilitation tool usually done with a poster and some sticky notes. It allows a meeting to continue efficiently while still noting items that will require follow-up outside the meeting.

At the end of a meeting, 10 to 15 minutes should be spent reviewing the items put in the parking lot to identify:

- Items that must be addressed now
- Items that need to be addressed but not right now (so they remain in the parking lot)
- Items that no longer need to be addressed (remove from the parking lot or move to a done state)

6.2 Tools Supporting Agile Requirements Engineering

6.2.1 Voting: The Planning Poker

Planning poker is the tool most widely used to vote in an Agile environment. Normally, if a group of stakeholders is asked to provide an estimate, usually the person who has a clearer view of the user story and of the related business will be the first to express an estimate. Unfortunately, this will strongly influence the estimates of all the others. Planning poker is an excellent technique to avoid this problem because all voting team members display their estimates simultaneously. Differences between estimates are then discussed and re-voting occurs. A first description and other links explaining planning poker can be found in [Planning Poker].

A technique such as planning poker is extremely useful when voting for estimates related to Agile requirements such as:

- Business value and priority: higher estimations may promote the related requirement to a higher position in the product backlog which means it will be addressed in the earlier sprints. Lower estimations may demote the requirement in the product backlog which may cause it to be withdrawn altogether.
- Risk: higher estimations of risk position the related requirement to be discussed further, maybe causing it to be refined and split, and then moved to a higher position in the product backlog. Lower estimations (low risk) may demote the requirement to the last rows of the product backlog.
- Complexity, size and effort: higher estimations may cause the related requirement to be furtherly discussed, refined and split. Lower estimations may promote the requirement to a higher position in the product backlog.

For training companies: based on an example of a product backlog, provide an exercise for a voting session using planning poker.

6.2.2 Timers

Timeboxing is a key issue in Agile development. A timebox is a previously agreed period of time during which a person or a Team works steadily toward completion of some goal. Rather than allow work to continue until the goal is reached, and evaluating the time taken, the timebox approach consists of stopping work when the time limit is reached and evaluating what has been accomplished.

In traditional development it is common to see a prolonged time until a certain amount of work is completed resulting in schedule slips. In Agile, the time is no longer a variable; the rule of timeboxed work is that work must stop at the end of the previously agreed time; more time cannot be allocated. When the limit has been reached progress is reviewed: has the goal been met, or partially met if it included multiple tasks? Only completed work is considered done. This concept is applied to virtually all activities in an Agile project: a meeting, a workshop, an iteration. All key practices of requirements engineering, such as workshops, analysis events, spikes and sprint reviews, are timeboxed and at the end of the time what has been achieved is evaluated and a new schedule is created as needed.

According to human ultradian rhythms, there are 90- to 120-minute cycles during which the body moves between high and low energy states. Every hour and a half or so a human needs to take a rest break, brief but regular, to keep high energy and attention. At the start of each time slot all distractions, such as e-mail and phone, should be turned off. A sand timer should be turned over. At the end of time the Team can decide to turn the timer immediately back over if the attention is holding, or the Team can decide to take five or ten minutes to check e-mail, return a call, or just look outside.

In the Pomodoro Technique® [Cirillo, 2007], Team members work in 30-minute increments. At the start of each increment, a tomato-shaped kitchen timer is set for 25 minutes. The Team works diligently during that time with no distractions from e-mail, the phone, and so on. When the timer goes off, Team members take a five-minute break. During these five minutes they can walk around, stretch, share stories, and so on but are discouraged from doing things such as talking about work or checking e-mail. Every fourth pomodoro, a longer break of from 15 to 30 minutes should be taken.

© GASQ – Global Association for Software Quality

This technique or any other similar technique using differently-shaped kitchen timers, different times between two breaks, but keeping a maximum of one hour of concentration, is a good way to govern timeboxed events with a high level of attention and energy.

6.2.3 Tools Supporting Communication

Effective communication between the Team and the product owner, between the product owner and the stakeholders, the Team and the stakeholders, and among different Teams within a project or between multiple projects with dependencies, is a pillar of any Agile environment.

The mechanisms of communication become much more complex as development complexity grows and a large organization is needed. In recent years the rise of geographically distributed organizations, offshore Teams and outsourced activities, has increased the complexity of communication, raising questions about the feasibility of face-to-face conversation. The organization of workshops bringing together all participants in one room is often impossible and the usual tools – wallpapers, whiteboard shots, sticky notes, photos of the walls – that allow a natural documentation of the work in a group, may become inadequate when everyone is not present or when the analysis becomes complex or regards mission-critical applications. The risk of misinterpretation and misunderstanding increases when the face-to-face communication decreases.

There are two categories of tools supporting communication: tools bringing the involved people together and making them communicate, such as audio-video conference systems, and tools allowing the involved people to make available to their colleagues the results of their discussions and conversations about requirements in a simple, shared and understandable way for everyone.

There are a number of techniques that can be implemented and made available by tools designed to facilitate and ease the communication in the most problematic situations for Agile, distributing the results of the analysis without sacrificing the discussion in favor of the documentation. Some of the most common techniques include:

- Activity diagrams (flowcharts)
- Sample reports
- Pseudocode
- Decision tables and decision trees
- Finite state machines
- Message sequence diagrams
- Entity-relationship diagrams
- Use cases

There are several tools, both commercial and open source, that support Agile development and encourage communication over documentation even in geographically distributed organizations. Whatever the techniques chosen by the Team to exchange information, it will be very important to choose the most suitable tools in order to avoid the tool replacing communication. The tool should not become only a means to write a detailed technical document. The conversation between stakeholders must remain the main focus, and techniques provided by these tools must be one way to promote the understanding and dissemination of communication between people.

© GASQ – Global Association for Software Quality

6.2.4 Other Tools Supporting Agile Requirements Engineering

[REQB_FL_SYL] lists some categories of tools that support requirements engineering activities.

In traditional software development, in many cases, these tools are used to support the documentation. They may also support a few specialists that play, separately for the most part, the roles of analysts, architects, and designers while drawing up detailed specifications of requirements and solutions to implement them. In an Agile environment specific tools should be used only when deemed necessary by the Team and other stakeholders.

The product backlog is already the main tool in Agile for requirements elicitation, requirements management and, together with the sprint backlog, project management. In fact, the backlogs in Agile can be tracked on simple spreadsheets, but spreadsheets can also be structured and full of information to manage the traceability of requirements and resources. In the case of geographically distributed Teams, Teams often use management platforms that enable an easy distribution, under configuration control, of information such as the product backlog and other artifacts typical of an Agile project [Larman, Vodde, 2010].

Even the modeling tools can be useful in an Agile environment, but the utility is not in the automatic production of a specification document from a model, but in using the tool to create a model that is more clear, understandable and distributable compared to the wallpaper on which the same pattern was drawn during the discussion in an analysis workshop.

In any case, a tool should never replace the direct conversation and sharing of information between the Team and other stakeholders. It is a task of the Scrum Master to make sure that the use of a tool is functional both to support and facilitate communication, not to replace it.

© GASQ – Global Association for Software Quality

7 References

7.1 Tables/Figures

Tables

Figures

7.2 Standards

See Standards listed in REQB Foundation Level Syllabus

7.3 REQB Documents

[REQB_APPROACH] REQB® Approach to Requirements Engineering, Version 1.0

[REQB_FL_SYL] REQB® Foundation Level Syllabus, Version 2.1

[REQB_GLO] REQB® Standard glossary of terms used in Requirements Engineering, Version 1.3

7.4 Books and Publications

[Anderson, 2010] David Anderson, “Kanban: Successful Evolutionary Change For Your Technology Business,” Blue Hole Press, 2010.

[Beck, Andres, 2004] Kent Beck and Cynthia Andres, “Extreme Programming Explained: Embrace Change, 2nd Edition”, Addison-Wesley The XP Series, 2004

[Cohn, 2004] Mike Cohn, “User Stories Applied: For Agile Software Development”, Addison-Wesley Professional, 2004

[Cohn, 2010] Mike Cohn, “Succeeding With Agile – Software Development Using Scrum”, Addison-Wesley, 2010

© GASQ – Global Association for Software Quality

[Crispin, Gregory, 2008] Lisa Crispin and Janet Gregory, "Agile Testing: A Practical Guide for Testers and Agile Teams," Addison-Wesley Professional, 2008.

[Derby, Larsen, 2006] Esther Derby and Diana Larsen, "Agile Retrospectives – Making Good Teams Great", The Pragmatic Bookshelf, 2006

[Gottesdiener, 2002] Ellen Gottesdiener, "Requirements by Collaboration: Workshops for Defining Needs", Addison-Wesley, 2002

[Humble, Farley, 2010] Jez Humble and David Farley, "Continuous Delivery", Addison-Wesley, 2010

[ISTQB_FA_SYL] ISTQB Foundation Level Agile Tester Syllabus, Version 1.0

[Larman, Vodde, 2010] Craig Larman and Bas Vodde, "Practices for Scaling Lean and Agile Development - Large, Multisite, and Offshore Product Development with Large-Scale Scrum", Addison-Wesley, 2010

[Leffingwell, 2011] Dean Leffingwell, "Agile Software Requirements. Lean Requirements Practices for Teams, Programs, and the Enterprise", Addison-Wesley Agile Software Development Series, 2011

[Linz, 2014] Tilo Linz, "Testing in Scrum: A Guide for Software Quality Assurance in the Agile World," Rocky Nook, 2014.

[Schwaber, 2001] Ken Schwaber and Mike Beedle, "Agile Software Development with Scrum," Prentice Hall, 2001.

7.5 Other References

The following references point to information available on the Internet and elsewhere. Even though these references were checked at the time of publication of this syllabus, the REQB cannot be held responsible if the references are not available anymore.

[Agile Manifesto, 2011] Various contributors, Manifesto for Agile Software Development, 2001, www.agilemanifesto.org

[Cirillo, 2007] Francesco Cirillo, The Pomodoro Technique®, 2007
http://pomodorotechnique.com

[Cockburn, 2007] Alistair Cockburn, A user story is to a use case as a gazelle is to a gazebo, 2007
http://alistair.cockburn.us/A+user+story+is+to+a+use+case+as+a+gazelle+is+to+a+gazebo

[Cohn, 2012] Mike Cohn, Two Examples of Splitting Epics, 2012
http://www.mountaingoatsoftware.com/blog/two-examples-of-splitting-epics

[Jeffries, 2001] Ron Jeffries, Essential XP: Card, Conversation, Confirmation, 2001
http://xprogramming.com/xpmag/expCardConversationConfirmation

[Konrad, Over, 2005] Mike Konrad and James W. Over, Agile CMMI: No Oxymoron, 2005,
http://www.drdobbs.com/agile-cmmi-no-oxymoron/184415287

[Leffingwell, 2010] Dean Leffingwell, The user story primer, 2010
http://scalingsoftwareagilityblog.com/wp-content/uploads/2010/01/user-story-primer_1.pdf

© GASQ – Global Association for Software Quality

[Planning Poker] Wikipedia, Planning Poker, 2014, http://en.wikipedia.org/wiki/Planning_poker

[ScrumGuide, 2013] Ken Schwaber, Jeff Sutherland, The Scrum Guide – The Definitive Guide to Scrum, 2013, https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide.pdf#zoom=100

[ScrumPrimer 2.0] Pete Deemer, Gabrielle Benefield, Craig Larman, Bas Vodde, The Scrum Primer 2.0, 2012, www.scrumprimer.org

[Wake, 2003] Bill Wake, INVEST in Good Stories, and SMART Tasks, 2003 http://xp123.com/articles/invest-in-good-stories-and-smart-tasks

© GASQ – Global Association for Software Quality

Index

© GASQ – Global Association for Software Quality

© GASQ – Global Association for Software Quality