Requirements Engineering and Agile Development
- collaborative, just enough, just in time, sustainable -

By Rainer Grau and Kim Lauenroth,
with support from
Bogdan Bereza, Erik van Veenendaal and Sven van der Zee.
English Review by Gareth Rogers.

Content

1 Introduction

1.1 Requirements Engineering as Craftsmanship

Organizations working in an agile environment are proven to deliver products or services that are close to the needs and expectations of the customer. Important benefits from working agile are higher flexibility, faster time to market and increased productivity. These benefits do not come for free. Organizations that have transformed in this direction deeply invested in the development of people in the form of training and coaching; in methods and techniques; and in tooling that supports these methods and techniques (see [1]).

A very good example for such investment is the movement around (acceptance) test-driven-development and continuous delivery. Behind these two terms lies a large set of techniques and methods, complemented by non-trivial investments in environment and tools, investments into people development and last but not least a changed mindset regarding collaboration. Organizations that succeeded in going down this long road of continuous improvement in testing and delivery would never go back (see [6]).

When it comes to the discipline of requirements engineering, the agile community is unhappy with traditional approaches that create upfront documents of sometimes even hundreds of pages creating “Specifications” that are outdated the time the first line of code is under development. This unhappiness is even codified in the manifesto for agile software development:

“Working software over comprehensive documentation“

Unfortunately, this has resulted in the perception of the agile community when looking into “old-fashioned” descriptions of the discipline requirements engineering as: “old stuff, useless, we have to do it differently”.

This is where we - as the representatives of the International Requirements Engineering Board (IREB) - have a strong opinion: Requirements Engineering is a discipline that is independent from any particular process model or software development approach. The IREB-Definition of requirements engineering [2] says:

> Requirements engineering is a systematic and disciplined approach to the specification and management of requirements with the following goals:
>
> (1) Knowing the relevant requirements, achieving a consensus among the stakeholders about these requirements, documenting them according to given standards, and managing them systematically
> (2) Understanding and documenting the stakeholders’ desires and needs
> (3) Specifying and managing requirements to minimize the risk of delivering a system that does not meet the stakeholders’ desires and needs

Moreover, we believe that the discipline of requirements engineering constitutes a core competence in every software development process since it is responsible for continuous communication and confirmation of what needs to be done to satisfy stakeholder and user needs and why this is the right thing to do. Requirements engineering is an essential craftsmanship needed in the complex world of development of products and services.

Our answer to the claim “old stuff, useless, we have to do it different” is: “old stuff? Yes; useless? No!; we have to do it differently: Yes, for sure!“. Many of the IREB members are engaged in the agile movement to increase flexibility and productivity and to deliver in faster time to market sharing the mindset of agile organizations.

1.2 About this Article

This article discusses the discipline of requirements engineering under the view of different life cycle models. The article points out that requirements engineering is an essential success factor in any life cycle model, whether waterfall or agile.

The article reflects in section 2 why many organizations change from sequential life cycle models towards iterative and agile ones. The goal of this change is to establish a continuous flow of requirements from user and customer side through the organization. Uncertainty is a major aspect in this reflection.

Section 3 discusses the application of requirements engineering activities in agile environments. We will see that the body of knowledge in requirements engineering surprisingly is as relevant in waterfall life cycle models as in agile life cycle models. Section 4 focuses on the differences of application, sequence and timing of the requirements engineering related activities of waterfall and agile life cycle models.

With section 5 this article considers if requirements engineering, seen as craftsmanship, is compliant with the agile mindset by checking the IREB body of knowledge against the agile manifesto. This validation is essential to determine whether the IREB body of knowledge is comprehensive in agile environments.

But, let’s start with the root cause of most of our troubles in software engineering: uncertainty.

2 Uncertainty is a Fact

2.1 Uncertainty is an Attribute of modern customer-oriented Markets

The demand for increased flexibility and productivity has its roots in dramatically shortened product cycles and in high market pressure to deliver products and services that delight customers. Investigations show that product cycles shortened dramatically from an average of six years in the 1980s down to six months in 2010 [7].

This has had a drastic impact on organizations. As innovation circles are so fast, time to market pressure is high and long term planning horizons are of no relevance because of a dynamically changing market and economy. Reliability of mid- and long-term planning is low because of increased uncertainty in the market development. A successful organization is able to adapt its business model and product and service portfolio fast in response to the changing market drivers.

The agile and lean movement is one approach that supports performing successfully under conditions of increased uncertainty. Companies that perform successfully under these conditions often adopt agile and other iterative methods and techniques throughout the whole organization. In fast changing and consumer-oriented markets agile and iterative life cycle models have proved to increase the success rate of projects [1].

2.2 Why Organizations stick to Waterfall

Given the evidence that an agile life cycle model increases the success rate in fast changing markets – why do many organizations still stick to classic approaches of large up-front specification phases resulting in huge specification and ending up in waterfall-like processes?

Among others the most important reasons are:

The illusion of predictability: Classical management treats uncertainty as a personal issue. The self-perception of management is “management has to exactly know where to go”. In this mindset management defines the goals. As result, typically goals, plan and budget are fixed. Management treats changes to goals, plan and budget as issues and risks in reaching the goals instead of the “normal” course of life under the conditions of a dynamic market. Goal, plan, budget driven management see agile life cycle models as an additional risk in reaching committed goals instead of an smart mechanism to adapt successfully to changes.

(Business) projects have fixed milestones: Organizations that deliver products and services do not only have to deal with changing markets. As important is compliance to legal and other regulations. Examples are banks, insurances, medical, environmental services, energy and others. If a regulation changes the organization has to be compliant with this regulation at a fixed milestone. So the goal (fulfilled compliance) and plan component (at this milestone) are set from outside. In organizations where major investments go into compliance and regulations the goals, plan and budget mindset is predominant. The dynamic and market oriented part of the investment budget is – often for reasons of standardization – handled with the same mindset: goal, plan, budget.

Organizations with a different history or environment take the chance to act differently. Success stories like Amazon, Facebook or Google demonstrate that fast adoption to changing markets based on user feedback is a valid approach to manage uncertainty. Many successful Google services started with a beta version and emerged using the feedback of the users. Users appreciate this behavior. Management in these organizations is of the type: "We do not know what the goal really is, actually our users know; but we know how to get there as fast as possible". This is a very different approach from classical organizations; an approach that incorporates agile and lean principles.

3 Requirements Engineering and Agility

Following the idea of incremental innovation by iterative development in small steps, like Amazon, Facebook or Google, implies fundamental changes in the operating model of an organization. Things previously done in sequence need to be executed in parallel. This applies to all different aspects in product and service development, from strategic level down to operation.

3.1 Agility fosters a continuous Information flow in Organizations

The classical sequential information flow, starting with the definition of a strategy, breaking it top down to a portfolio definition and development plan for a set of products with a fixed roadmap will not work in these scenarios. The construct of a fixed one to five years business program plan is gone.

Instead, a continuous workflow of highly parallel character through all levels of an organization supports mutual influence and feedback. Information flow from strategic level down to operation is as important as feedback flow from users over the product and portfolio level back into the strategy definition activities reflecting current market changes.

Individuals in an organization work in parallel on the adoption of the strategy (based on bottom up feedback); on the development of the product and service portfolio (as alignment to the new strategy and to emerge the existing portfolio based on customer feedback); and in development and operation (the delivery of the next product version).

3.2 Impact on the Discipline Requirements Engineering

Waterfall life cycle models order the activities sequentially with predefined roles. Every process step has predefined inputs and outputs, and lasts for a defined period. Industrial experience[1] shows that such a sequential process typically falls short, since the development team and the stakeholder improve their understanding of the system throughout the whole development process. Lessons learned because of better understanding the problem domain has to flow back into the project in form of feedback.

[1] This experience was already presented by Winston W. Royce in his original paper "Managing the Development of Large System Systems" (often referred to as the original "waterfall paper").

Waterfall life cycle models address this by a change management process as explicit measures to manage the flow of information back into the projects based on lessons learned, issues and external changes within the context of work. In many organizations, the change management process is heavyweight with high impact costs and effort.

The timely order of activities and the way of collaboration is different in the agile life cycle model. At all times throughout the course of the project, the team works closely with stakeholders and users on requirements and develops the productive system incrementally. The core objective of the agile life cycle model is to get fast and direct feedback from stakeholders and users, ideally by using the system. Direct feedback from using the system has proved to show the highest evidence that system development is going in the direction as intended by the system's objectives. In the discipline of requirements engineering agile teams works continuously with stakeholders and users:

- In elicitation of business requirements, i.e. in identifying and specifying new epics and user stories aligned to stakeholder and user objectives or in backlog refinement activities.
- On adding details to epics and user stories up to a confirmed definition of ready and by specifying acceptance criteria so the team has all information required to develop these requirements into a productive part of the system that delivers the expected values to the business
- In design, implementation during the development of the requirements to meet additional implicit non-functional requirements coming from areas such as usability (attractiveness, learnability, understandability)
- On testing in verification and validation of requirements engineering executed on the so far developed parts of the system in potential shippable quality and status.

The team executes all these activities in parallel. Every single team member is active in more than one role. When discussing a user story and defining acceptance criteria with a stakeholder, a team member acts in the role of a requirements engineer, when implementing the user story, hopefully following an acceptance test driven design process, the team member acts as developer and tester.

The difference between the two life cycle models is how roles are assigned to people. Organizations preferring the waterfall model often assign one person to exactly one role. The effect of this role lock is that many individuals work only in the context of the discipline on a feature of the product. Responsibility for product features is spread over many individuals. Upfront paper work replaces direct communication and feedback to stakeholders thus resulting in assumptions that might show up as wrong and will end up in change requests. A project organization like this acts in kind of a Chinese whisper game through the roles participating in development of a requirement. In organizations preferring the agile life cycle model, every team member is acting in more than one role and switches roles as the work requires. Direct communication with stakeholders and fast feedback creates a current and active team knowledge that reduces the need for documentation.

3.3 The Impact of Assumptions

Another important factor in fast changing markets is the reliability and half-life of assumptions. A requirement that neither is a pre-defined goal nor a fixed constraint nor is already validated by the user can be considered as an assumption. Such a requirement describes a product or system feature that will come into existence in the future. Nevertheless, in the end we must confess that all these statements are assumptions as long as the developed product does not validate them. Assumptions may be valid and may be wrong. Failures based on wrong assumptions occur in waterfall and agile projects and have to be handled. Important is to understand the implications of the life cycle model when handling invalid assumptions:

- The period between making the assumption and validation of the assumption by the developed system is much longer in waterfall projects than in agile projects. In waterfall projects, the assumptions are validated within an explicit integration and user acceptance test phase. Agile projects with short iteration cycle validate assumptions in every iteration based on user acceptance tests and in review sessions with stakeholders.
- The longer the period is to validate an assumption against reality the higher is the probability of changes, especially in environments where the context around the project itself changes. A correct assumption at the time when discovered will more likely become invalid during the longer cycle time within a waterfall project than within the short cycle time of a sprint in an agile project.
- The desire of waterfall projects to create a "complete" requirements specification upfront increases the number of assumptions made. This makes the aforementioned issues when working with assumptions even worse. Abstract requirements are decomposed to a set of more concrete requirements. In case one abstract requirement turns out to be based on an invalid assumption, the complete derived chain of requirements is invalid as well. Such dependency chains can dramatically increase the impact of a false assumption. The potential rework in waterfalls projects is higher compared to agile projects resulting in more change requests related to invalid assumptions.

Agile life cycle models try to avoid this by restricting requirements engineering effort on a single requirement to the minimal amount required to understand the requirements down to a level of detail that is needed at that point in time. If the requirement is subject of today's implementation, the level of detail is high. If the requirement is planned for implementation within 3 months, the most abstract level of detail to understand core attributes of the requirement might be sufficient. Agile release planning and roadmapping activities, often based on backlogs, are responsible for orchestrating and steering the investment of effort in requirements engineering.

3.4 Manage Uncertainty by Feedback and Learning

For example: Let's assume an organization decides to offer location-based cloud services that are accessible for users by a set of mobile applications on any mobile device.

- The classical approach is sequential: market research, customer segmentation, service feature definition, feature development, start with marketing, launch – and probably fails as the world turned meanwhile and customer segments and the demanded feature set have changed.
- An agile life cycle model is iterative: have a vision, start marketing, develop the minimal viable product (welcome lean startup), launch very fast, get feedback, emerge by the next minimal set of demanded features, launch again very fast, get feedback…and so on.

The difference between the two approaches is the fact that a limited group of individuals is not capable of representing or anticipating the demand of a heterogeneous crowd of users. Speaking in terms of cognitive abilities, a fixed group of people only has limited capabilities to predict future developments (cf., e.g., [3]). The only way to identify the demand is to get real feedback. Users give feedback either direct or service built-in analytics measure user behavior. Feedback is a type of communication with the crowd. In terms of requirements engineering this is an elicitation activity. The app rating mechanism in an app store like iTunes in fact is feedback mechanisms and a transparent form of analytics.

The qualified feedback from the crowd supports organizational learning. Learning outcomes are:

- We get real feedback from real customers and not our own (potentially false) beliefs of what the customer might think
- We identify the customer segments more precisely
- We understand the needs of the customer segment more precisely
- We can learn to predict the future to some extent: anticipate the next incremental innovation (features) that will improve our product

This approach implies an iterative and incremental development process. The cycle time between delivery milestones of an emerging product depends on the market. In fast changing markets this could be the next day; in other markets four deployments per year might be sufficient. The important fact is that the cycle time is becoming shorter and shorter in every single market. If it is six months today, it might be six weeks tomorrow.

4 Requirements Engineering as Discipline is independent from the Process Model.

The fundamental finding is, no matter whether the project to develop a cloud service uses a waterfall process or an agile process, the discipline of requirements engineering is as important in either case. Nevertheless in the example of developing a cloud service, the agile process most probably creates a results that better meets user needs and with this the objectives of the organization developing the service.

4.1 The Application of the Discipline is different

From a high-level point of view, the discipline requirements engineering in the waterfall and agile life cycle model are comparable. To demonstrate this we analyze the core activities as defined in the IREB syllabus [5, page 6]:

- Elicitation [IREB citation]: *During requirements elicitation, different techniques are used to obtain requirements from stakeholders and other sources and to refine the requirements in greater detail.*
  Waterfall projects as well as agile projects perform this activity. The methods and techniques used are identical. Examples for methods and techniques are interview, apprenticing, storyboarding. Responsible to draw out elicitation is a person acting in the role requirements engineer. Agile projects apply this activity continuously. In waterfall style projects, this activity is applied mainly in a requirements analysis phase in the beginning of a project.
- Documentation [IREB citation]: *During requirements documentation, the elicited requirements are described adequately. Different techniques are used to document the requirements by using natural language or conceptual models.*
  This activity is the same in waterfall as in agile projects. The methods and techniques to create documentation are identical or similar. The quality criteria in the IREB syllabus for requirements [5, page 13] can be mapped to the SMART and INVEST principles of user stories [8]. From the point of view of the IREB syllabus [5, page 11], epics and user stories fall in the category of natural language requirements documentation of scenarios. Responsible for developing epics and user stories is a person acting in the role requirements engineer. Within an agile life cycle model, the order of activities is completely different to the documentation of requirements within a waterfall life cycle model. Waterfall projects decompose all relevant requirements in the “concept Phase” down to a fine-grained detail level. Agile projects start with an agreed vision represented by a small set of high-level requirements. The team decomposes a requirement at that point in time when pulled into development. High-level requirements, in agile projects often called “product features” [9], are decomposed into epics and user stories. The requirements specification in an agile project is a living document as long as development is active. Comparing the effort invested in a requirement specification in both life cycle models would be an interesting investigation. Probably agile projects invest more effort into requirements engineering than waterfall projects, aware that the effort results in high value for the project [1].

- Validation and Negotiation [IREB citation]: *In order to guarantee that the predefined quality criteria are met, documented requirements must be validated and verified early on. Furthermore, the stakeholder agreement is a prerequisite for the acceptance of the new product/system, therefore requirements have to be negotiated.*
  In waterfall projects, this activity is confined to the requirements related phases. Requirements are validated and negotiated based on documents (e.g. the requirements specification). Once these activities are finished, an error or a conflicting requirement initiates a change process that reactivates these activities to correct the error or to resolve the conflict. The agile life cycle model extends validation and negotiation of requirements from first idea until delivery. Validation and negotiation are not performed on documented requirements only. In agile projects this is an intrinsic principle implemented by a set of means as there are: fast feedback by short iterations, definition of ready; acceptance criteria; definition of done; test driven design, review sessions with stakeholders and many more. IREB codifies this approach as the fifth principle of validation: validation based on the construction of development artefacts [5].
- Management [IREB citation]: *Requirements management is orthogonal to all other activities and comprises any measures that are necessary to structure requirements, to prepare them so that they can be used by different roles, to maintain consistency after changes, and to ensure their implementation.*
  This activity is executed in waterfall as well as in agile projects. In waterfall projects, this activity is typically assigned to a dedicated role (project manager, requirements engineer, …) responsible for managing the requirements documents. In agile projects, requirements management is incorporated into various activities and distributed over the team members. A team executing a backlog refinement meeting with stakeholders manages requirements by decomposing requirements. Prioritization is an important activity and requires management activities around requirements. In Scrum the Product Owner is responsible to prioritize requirements from different sources such as stakeholders, support organizations or system architecture. Another example of requirements management is the reference of user stories in the source code (production code as well as acceptance test code) to show which part of the software implements a user story, i.e. traceability or the progress tracking realized by updating the release burn-up.

From a high-level point of view, the disciplines of requirements engineering in the waterfall and agile life cycle models are comparable. When going into the details of how the discipline of requirements engineering is applied, the waterfall and agile life cycle models are very different. Both life cycle models are relevant for different scenarios. For developing a cloud service as mentioned above the agile life cycle model most probably is the better choice. The relocation of a data center or the construction of a train tunnel might always require a sequential and waterfall-like life cycle model.

As a reflection on the discussion so far, the core differences in the application of the requirements engineering discipline in waterfall and agile life cycle models are:

- The order of the activities of requirements engineering
- The assignment of the role "requirements engineer" to team members in the sense of a multifunctional team, with one person acting in more than one role
- The greater focus on continuous collaboration and communication between team members of agile teams with stakeholders and users to receive feedback on what has been developed so far and what is under development.

So the core difference of the waterfall and the agile life cycle model is the application of the requirements engineering discipline, not the body of knowledge.

4.2 Requirements Engineering in complex and large Agile Projects

The typical unit for structuring agile projects is the sprint, which is defined in terms of weeks, e.g. 2-4 weeks. The idea and concept of an agile team as defined in Scrum working on a backlog in sprints is sufficient for small projects. If a project requires more work to be done, by two, three or ten teams in parallel, the concept of Scrum needs to scale. Beyond Scrum, there are several approaches on the market that address this. One for example is the scaled agile framework SAFe [9] to manage longer product roadmaps, larger projects with up to several hundred developers, and whole organizations from marketing and sales to delivery and operations. In the essence, such approaches scale the concept of cadences, rhythms and ceremonies from team level to organizational level and increase the level of abstraction of artefacts. It is interesting to see the implications on the discipline of requirements engineering [9].

- User stories / short term sprint planning / from today up to three sprints into the future: The team plans the next sprint based on the product backlog / level of detail that can be implemented by the team.
- Product feature (epics) / medium term planning / up to 3-6 months: more coarse grained features of the product. Reflects the product release planning. Decomposing of features (epics) requires additional effort, the so called backlog refinement.
- Objectives and goals / long term planning / 3 months to 1 year: describe the business cases of the product / company and reflects the roadmapping activities.

From the perspective of the discipline requirements engineering, such an approach utilizes predefined abstraction levels to structure the development and discussion of specification concepts. The application of abstraction levels is by no means limited to agile projects. Abstraction levels are a well-known and well established concept to decompose information. Alistair Cockburn for example defines several abstraction levels for use cases (cloud level, kite level, sea level …) [10]. Another example is the requirements abstraction model [11]. These concepts are valid and used as well in waterfall as in agile projects. Even name and concept mappings between these abstraction levels on requirements exist, as used in classical waterfall projects and agile projects [15].

With this we can state that scaling agility from team level to organizational level reuses requirements engineering concepts under new names that are already well established and ceremonies in the agile world.

4.3 The Evolution of Requirements Engineering Documentation Formats

What is not that different – surprisingly – are the documentation formats used in waterfall and agile environments in the discipline of requirements engineering. Under documentation format, we consider the type of artifact used to document requirements. For example, a user story is one type of a documentation format that combines a set of information units in a very specific combination. Of less significance is the media used to persist a document in a specific documentation format. An agile team might decide to pin physical cards to a board or to use a software tool like Greenhopper to persist user stories. From the point of view of the discipline requirements engineering the documentation format is “user story”.

Of interest is the observation that these so radically different looking documentation formats claimed as agile inventions as, for example, “user stories” are in fact a smooth and continuous evolution of well-known requirements engineering concepts. Scenarios have been an important element in requirements engineering for over twenty years. Scenarios substantiate abstract goals by example [16]. Typical information units of scenarios are persons or systems that interact; the goals that a scenario substantiates; concrete or virtual locations where a scenario takes place. If we investigate user stories, we discover the same information units, a person (“as a <persona>”) acting in a very concrete environment (the location) with the interest to achieve a goal (“so that <the rational>”). Even in the definition of quality attributes for requirements, we see a continuous evolution. The SMART and INVEST quality criteria definitions are acronyms for quality attribute specifications for user stories.

In many complex and large agile projects as discussed in the previous section agile teams still work with document formats well established before agile life cycle models became that popular. Many large-scale agile projects rely on the documentation formats of a use case model, a data model, business rule specifications, features and a written requirements specification holding quality requirements and constraints as well as detailed technical requirements such as for example the definition of an mathematical algorithm. SAFe [9], for example, uses the documentation formats of epics, vision, features and user stories. These concepts map to the concepts of goals and scenarios in requirements engineering [16]. The concept of Scrum with a sprint goal and user stories maps as well to the concept of goals and scenarios in requirements engineering. Therefore, the understanding of requirements engineering concepts deepens the understanding of the mechanisms of action behind the agile equivalents and evolutions. The documentation formats used in agile life cycle models therefore represent an evolution of goal models and scenarios. The fundamental and game-changing difference is that the agile community introduced new names for concrete evolutions of concepts such as user story as an evolution of use cases. This is positive as new names transport an intent, in this case the intent to use these documentation formats in agile life cycle models in a specific combination and purpose and driven by a different mind-set.

With the focus on continuous information flow and the lean approach to detail requirements only when needed, a requirements specification is rather an emerging documentation then the result of an upfront activity. With this, an agile specification holds information only that is useful for the team and stakeholders. Often the specification is on the more abstract level of goals and objectives. This information supports stakeholders and the team to align the mid-term development with the strategic direction, to draw out fundamental decisions about the systems development roadmap and release planning and to document important core knowledge (such as algorithms) that might get lost over time otherwise. This documentation exists in form of artifacts independent from a backlog.

Under the more abstract view of what documentation is, implemented and automated acceptance tests are as well part of the requirements specification. In classic style requirement specifications a chapter or document “acceptance tests” existed. Today an executable specification or an acceptance test driven testing suite may replace this (chapter in a) document. Nevertheless, it is important to note that the capability to understand the requirements of the system and to “write” high quality acceptance tests is required. A person writing these acceptance tests is acting in the roles of a requirements engineer and tester.

An interesting point is that the team creates this documentation continuously and not as upfront work. The requirements specification in an agile environment is a living thing, represented partially in different containers, even as an executable acceptance test suite. This different way to carry out requirements engineering activities in an agile environment results in a living requirement specification that is up-to-date, correct and useful. A disciplined and mature agile team that owns all required capabilities develops the requirements documentation of the system as the system itself is developed. This type of requirement specification contains only information that is

- required by the team to build the system aligned to a product roadmap
- required by new team members to build up the shared knowledge and personal capabilities to be productive as fast as possible
- required by the team to maintain and support the operational capability of the system
- required by the organization to continuously improve the product aligned to the organizational strategy, i.e. to develop a transparent product roadmap and release planning.

5 The Agile Manifesto from the Requirements Engineering Perspective

Until now, we have discussed the values of requirements engineering from the perspective of the agile community. We discovered that the body of knowledge in requirements engineering is as important and relevant in waterfall as in agile life cycle models. We even discovered that many elements that apparently are inventions of the agile community in fact are evolutions of existing concepts. Yes, there are fundamental differences in how to apply the activities between these life cycle models. Nevertheless, we are convinced that a better understanding of the base concepts in requirements engineering deepens the understanding of the mechanisms in agile projects.

Now, in a final retrospective we want to reflect the body of knowledge as defined by IREB against agile values. The goal is to verify that the statements of IREB do not conflict with agile principles. Therefore, we investigate whether the IREB definition of Requirements Engineering does violate the Manifesto for Agile Software Development:

Requirements Engineering: A systematic and disciplined approach to the specification and management of requirements with the following goals: (1) (a) Knowing the relevant requirements, (b) achieving consensus among the stakeholders about these requirements, (c) documenting them according to given standards, (d) and managing them systematically, (2) (a) Understanding, (b) and documenting the stakeholder's desires and needs (3) (a) Specifying and managing requirements, (b) to minimize the risk of delivering a system that does not meet the stakeholders desires and needs. IREB glossary	We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value: Individuals and interactions over processes and tools Working software over comprehensive documentation Customer collaboration over contract negotiation Responding to change over following a plan That is, while there is value in the items on the right, we value the items on the left more. http://agilemanifesto.org

Individuals and interactions over processes and tools

Stakeholders (and users are stakeholders) are the main source of requirements. A core objective of requirements engineering is to understand the desires and needs (2a) of stakeholders and to derive requirements (1a). This cognitive process happens through intensive communication and interaction between the stakeholders and the team. Processes and tools are important to handle complicated content and support this cognitive process. Nevertheless, processes and tools are not an end in themselves; they have to be introduced with reason and care, and must be used with care and pragmatism.

Working software over comprehensive documentation

Requirements engineering, implementation and testing can be seen as the craftsmanship in software development. Requirements engineering is responsible for understanding user needs (3a) and for transporting this understanding to the development team, so that the team builds what the user requires (3b). Documentation is not an end in itself. The core value is the knowledge about stakeholder and user needs (1a). Interaction with and between team members sharing knowledge is one way to preserve knowledge. Compliance and regulation constraints define specification standards for specific domains (e.g. medical, aerospace or automotive industry) that require a written documentation in relation to requirements. According to IREB, a team member, taking the role as requirement engineer, is responsible to know about and comply with standards and regulations that apply within the development of a product (1c) and to create the documentation to successfully meet compliance and regulation constraints.

Customer collaboration over contract negotiation

The IREB body of knowledge as in [2] discusses customer collaboration in the chapter of requirements elicitation and consolidation. IREB offers an explicit advanced level module [12] dealing with elicitation and consolidation of requirements with stakeholders and users. This advanced level module represents the emphasis of IREB into agile values. Learning objectives of this module cover important principles and elements used in agile projects, such as direct communication, personas, creativity methods, contextual inquiry, user stories, the CCC (card, communication, confirmation) principle [13] and others. The IREB advanced level module elicitation and consolidation can be seen as a core module in covering agile techniques and methods – as other IREB modules cover principles that are valid and important as well in waterfall as in agile projects.

Responding to change over following a plan

Agile methods like Scrum or Kanban address responding to change as one of the core agile values by specific methods, techniques and ceremonies. Responding to change is not an attribute of the requirements engineering discipline. Requirements engineering offers specific methods and techniques that support responding to change. Agile life cycle models give guidance on how and when to apply these methods and techniques.

In [2] IREB change management is an element of the body of knowledge. This way to apply change management is clearly for waterfall-like processes or for regulated environments that require an explicit change management like the development of medical software that is under FDA regulations. Therefore even the chapters of change management might be relevant within agile projects. A team that develops a medical software system under FDA regulations may refer to this chapter to understand how to execute change management compliant to FDA regulations as part of the projects deliverables. The software development process itself could be Scrum.

6 Summary

This article demonstrates that we, i.e. the International Requirements Engineering Board (IREB), are convinced that our body of knowledge is not outdated under the view of agile development. Instead we are convinced that the methods, techniques fostered by IREB are a natural part of the daily work in agile environments – probably applied differently in sequence and emphases.

We see requirements engineering as craftsmanship that uses a set of appropriate "tools", the requirements engineering toolbox. The "tools" in the requirements engineering toolbox are methods, techniques and (concrete) tools to execute the activities. Projects of all types shall benefit from this toolbox and take the subset of tools that leads to success in their specific context. Many of the tools lead to success as well in agile projects as in classical waterfall projects. Some are restricted to agile projects only, some to waterfall projects only.

We are convinced that the understanding of all these methods and techniques is of benefit for every professional involved in the development of new products and services, no matter what life cycle model a development team applies. One core objective of IREB is to foster the discipline of requirements engineering so that engineers developing products and services act as highly skilled professionals building up and using good practices that are based on commonly agreed and valuable methods and techniques.

References

[1] Chaos Manifest 2011, Standish Group International

[2] Requirements Engineering Fundamentals; 1. Edition; Klaus Pohl, Chris Rupp; Rocky Nook Inc. (April 2011); English, 184 pages Paperback; ISBN-13: 978-1933952819

[3] Kahnemann, D.: Thinking, Fast and Slow. PENGIUN Psychology, 2011.

[5] IREB Certified Professional for Requirements Engineering, Foundation Level, Syllabus Version 2.1, 1st March 2011

[6] Supporting Agile Teams of Teams via Test Driven Design; Kris Read; Thesis submitted to the faculty of graduate studies in partial fulfillment of the requirements for the degree of Master of Science; department of computer science; Calgary University Alberta; Feb 2005

[7] Creative Process and Product Life Cycle of High-Tech Firms – Creativity and Innovation, key drivers for success; Verna Lu and Cédric Marjot; Master Thesis; Baltic Business School, University of Kalmar; Sweden; June 2008

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

[9] The Scaled Agile Framework SAFe, www.scaledagileframework.com, and book: Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise; Dean Leffingwell; Addison-Wesley Professional; 1 edition (January 6, 2011); ISBN-13: 978-0321635846

[10] Writing Effective Use Cases; Alistair Cockburn; Addison-Wesley Professional; 1 edition (October 15, 2000); ISBN-13: 978-0201702255

[11] Requirements Abstraction Model; T. Gorschek and C. Wohlin; Requirements Engineering Journal, Vol. 11, No. 1, pp. 79-101, 2006;

[12] Syllabus CPRE AL Requirements Elicitation and Consolidation; version 1.0; Dec 2012; see: http://www.ireb.org/fileadmin/IREB/Lehrplaene/CPRE_Elicitation_and_Consolidation_Syllabus_Version_1.0.pdf

[13] Three C's; Ron Jeffries; Aug 30, 2001; card, conversation, confirmation; see: http://xprogramming.com/articles/expcardconversationconfirmation/

[14] Extreme Programming Explained: Embrace Change; Kent Beck; Addison-Wesley; 2nd edition (November 26, 2004); ISBN-13: 978-0321278654

[15] Agile Requirements Abstraction Model – Requirements Engineering in a Scrum Environment; Richard Berntsson Svensson, Sigurdur Örn Birgisson, Christian Hedin, Björn Regnell; Paper published at the Department of Computer Science, Lund University, Sweden

[16] Requirements Engineering, Fundamentals, Principles, and Techniques; Pohl, Klaus; 2010, XVIII, 814 pages, ISBN 978-3-642-12577-5