Don’t Put it in the Backlog

How Separating Requirements from the Backlog Can Help to Work Agile in a Regulated Environment

Frederic Siepmann <frederic.siepmann@brainlab.com>; Benedikt von Unold <benedikt.von-unold@corpuls.com>

Contents

Summary

- Agile work in a regulated environment requires a clear separation of requirements and backlog items to combine legal requirements and agile principles. A clean separation prevents misunderstandings and ensures that implementation changes do not constantly affect the requirements.
- The product backlog includes all necessary tasks to achieve the goal, while requirements describe the desired product features, which is particularly necessary for highly regulated industries.
- Refinement meetings are crucial for the continuous identification and validation of requirements as well as the adjustments of the product backlog, so that agile teams can work efficiently. Close collaboration in refinements promotes the development of a common language among all participants.
- Iterative approaches and the involvement of all relevant stakeholders are essential for successful agile product development.
- Agile methods such as Behavior-Driven Development (BDD) and Test-Driven Development (TDD) support the early testing to ensure the alignment of product requirements and implementation.

The term "agile" is increasingly subsuming more topics: From incremental software development, rapid prototyping, and continuous integration, to self-organized teams, new work, and agile leadership. A perfect breeding ground for misunderstandings. Hence, for working agile, especially in a highly regulated work context, we want to offer a recommendation here: The distinction and separation of the Product Backlog from the Requirements.

The separation of these concepts is an important basis for working agile and helpful for the work of any company. This is particularly true in our experience in regulated industries such as medical technology, because the documentation of requirements is subject to some legal requirements. This regulation and the associated documentation and seamless traceability of requirements, which is sometimes even used as an argument against working in an agile way, causes considerable additional effort when developing a product. At the same time, however, this regulation emphasizes the distinction between requirements for the product and the according implementation of them.

Requirement & Product Backlog: Strict Separation for More Agility

As the guideline for the use of agile practices in the development of medical device software - AAMI TIR45 [Aami12] as of 2023 - already states, a Product Backlog should include all the work that a team has to do. The emphasis here is on all work, which for medical products also includes the work on the documentation of the product and the documentation of the requirements. Other specifications, such as those for a comprehensive QM system according to ISO 13485, of course remain unaffected by this and must also be established in an agile working model. A common denominator that positively influences both the agile way of working and the development of requirements is the Product Backlog.

Even when developing medical products, business principles cannot be overridden, whatever agile working methods are applied, but it is well known that the focus in the project management triangle shifts from time to the scope of development. The other variables of the triangle, which in this case explicitly include quality, are considered immovable. This also means of course that the development progress of the product must be continuously monitored and “managed” - which inevitably places greater emphasis on the possible scope and its respective value contribution. And this is precisely where the Product Backlog comes into play as the tool that an agile way of working uses to keep the scope adaptable.

Agile working with requirements: Discover the product through a backlog

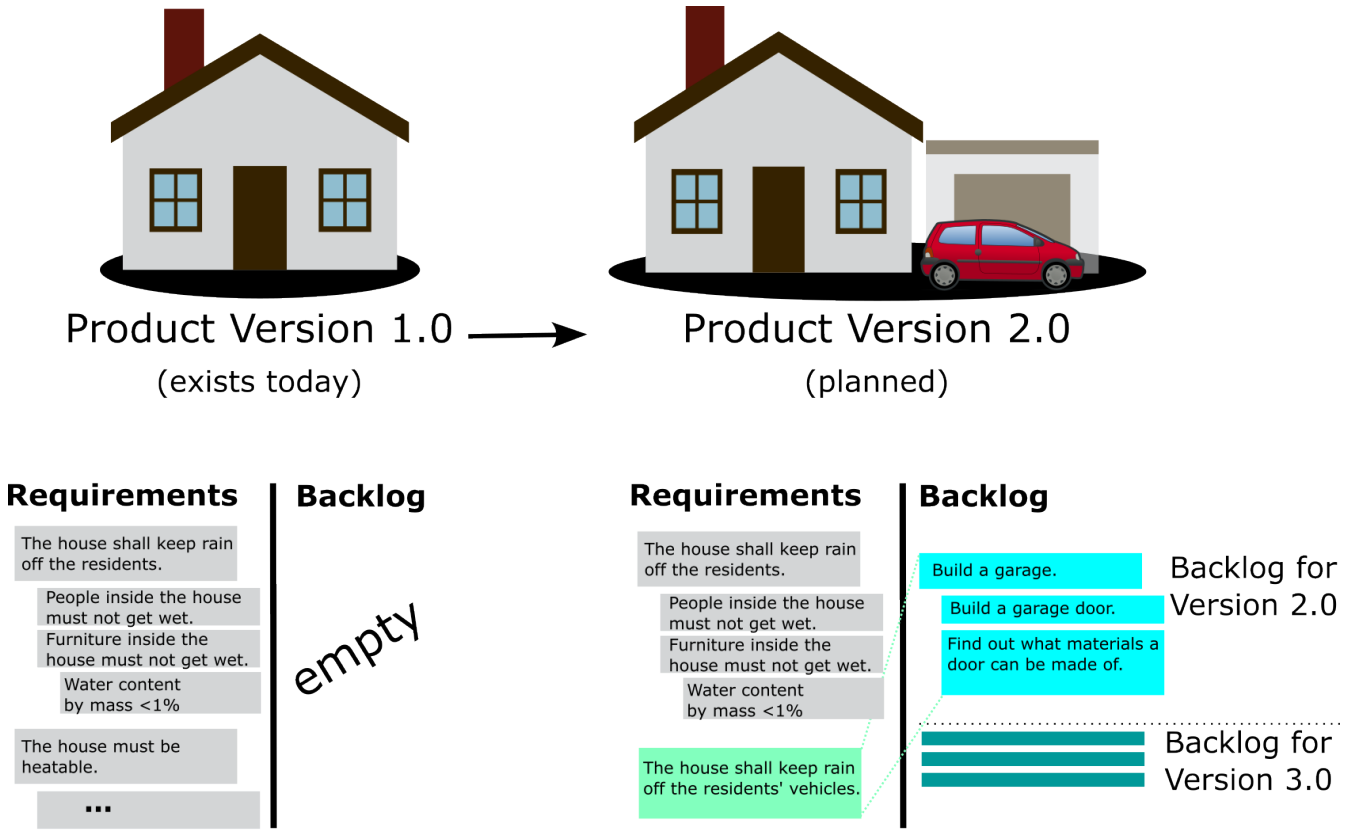

Figure 1. Example of a new product version (right) with new requirements and a resulting product backlog.

As the name suggests, the Product Backlog is fully focused on the common goal: The product. For the example in the image above, we have chosen a house as the product. This is deliberately not medical or complex, but comprehensible for everyone and reduced to the core of our statement.

The product version 1.0, which is already on the market, fulfills the requirements connected to the product. The Product Backlog is empty, as we no longer need to implement anything for the product in order to fulfill the necessary requirements. During the development of version 1.0, there was a backlog that was worked on accordingly by the teams. In our example, the house successfully keeps the rain off the residents and furniture and therefore meets the requirements.

A planned product version 2.0, as also shown in the image above on the right, should now address further needs and solve the user's problems. The requirements therefore make a statement about how the characteristics of a product (house with garage) must be so that the user achieves their goal or their needs are met. The requirements and their documentation therefore exist as long as the associated product exists and are expanded with new versions of the product. Even when the product is finished and on the market (and the backlog with its tasks has been completed), the requirements are part of the permanent documentation (in medical technology: TD or technical documentation). The requirements therefore serve as a mental model of all the criteria that the product should fulfill. In medical technology, this documentation is simply part of the product because it is a regulatory requirement for approval and thus the license for sale.

Separate problem and solution space

This abstract formulation of the requirements, without already addressing the possible solutions (problem area), is more difficult in practice than in our example - especially for engineers whose job it is to find specific solutions. A high level of attention and discipline is required to ensure that the requirements for our company are not too closely tied to the implementation. For example, formulations such as "The house needs a garage" or "The car needs a roof" may sound very similar and obvious, but they are already linked to a specific solution and significantly reduce our scope for development. It is essential to first understand the customer's perspective before thinking about the details of a solution.

In our view, the question of "How should our product behave **after** we have delivered it?" is particularly important with regard to this refinement of requirements. This conscious separation between "problem space" and "solution space" is an elementary concept that was described more than 60 years ago (Newell et al., 1962). It is an empirically proven, abstract procedure on which, among other things, the steps in design thinking are based (its double diamond consists of the problem and solution phases). Requirements, especially at a high level of abstraction, should therefore be as solution-neutral as possible - for example, they should not yet describe a specific implementation.

Guiding questions for requirements

We recommend that you allow sufficient time for the discovery of requirements throughout the entire development process. In an agile working model, this time would typically be scheduled in the form of Product Backlog Refinements. This also answers the question of where requirements are typically identified in an agile working model.

In our experience, teams often spend way too little time on refinement and therefore also on requirements. You should therefore track how much time is actually spent on this together with the teams. If, for example, you have a two-hour meeting for your refinement every sprint in which you go through the electronic backlog, you can say with a fair degree of certainty that neither the time invested nor the format is sufficient.

Work with guiding questions during refinement

Ultimately, our goal for the teams involved is to have a common understanding of the requirements. Especially with complex products, guiding questions during the refinement can not only promote the understanding, but also help to focus on important aspects of the product. The guiding questions also ensure better clarity in the discussions and thus avoid misunderstandings. When identifying requirements, for example, the following key question can help: "What is the simplest system we can build to get feedback on <product characteristic>?". In relation to our house example, a product characteristic would be the roof of a garage. The answer to our key question, "What is the simplest system we can build to get feedback on the garage roof?", would be a very simple carport that we can test and that can therefore answer our questions about the product characteristics. Other guiding questions that we can use in the refinement would be, for example: "What are the most important business objectives that we want to achieve with the product?" and "What criteria must be met for <product characteristic> to be considered successful?". The above-mentioned question of how our product should behave after we have delivered it can also be used as a guiding question in a refinement.

Coming back to the Product Backlog: As already mentioned, this is where we find tasks that need to be worked on in order to achieve the common goal. The higher up in the backlog we find the items, the more concrete, precisely defined and solution-specific they are. In contrast to a traditional project plan based on the waterfall model, not all tasks are recorded in the backlog at the beginning via a work-breakdown structure and their duration is shown on an axis (a.k.a. Gantt chart). The backlog focuses on the importance of the tasks in achieving the goal, i.e. their respective priority. Completeness is worked out iteratively in the aforementioned refinements.

Among other things, we recommend that items in the Product Backlog always start with a verb, because someone (e.g. a team) has to take action here (see image above: “Find out”, “Build”, etc.). The requirements, on the other hand, express characteristics of the product, i.e. the desired goal or target state.

The common misconception mentioned at the beginning is that requirements are located in the backlog. One of the consequences of this is that even tool manufacturers have mapped this way of thinking as a standard configuration in their solutions, for example work items are referred to as “features”. Apart from being in the wrong place, this also often results in formulations of requirements that are too closely connected to implementation.

One indicator that this separation has not been carried out properly would be, for example, if the requirements also have to be regularly adapted after an adjustment in the implementation was made in order to still match the new implementation. In such a situation, we recommend adapting the naming in the tool used in order to maintain the characteristics of a Product Backlog. For example, a “Feature” could be renamed to a “Deliverable”.

Identifying Requirements: Focus on Agile Events

Even without this distinction between the requirements and our plan for implementation, the people involved are faced with the crucial question: How do we get to the content? Or more specifically: What do I have to write in my Product Backlog and what ends up in the Requirements? We are deliberately leaving out the discussion about the respective tools in this article, as there is a large selection for both with their respective advantages and disadvantages. The only recommendation at this point is not to use the same tool for Requirements and Product Backlog. In our experience, there is currently no tool on the market that covers both aspects equally well and can separate the concepts and associated working methods well.

The Refinement

To adequately fill the Product Backlog and identify the Requirements, a procedure is needed that allows us to generate the necessary information for both. The agile way of working usually speaks of a “refinement”, i.e., an activity in which the development teams work together with all necessary knowledge carriers (including customers) to determine what the requirements for the product are and how they can best be implemented with the teams. The refinement takes place continuously in each sprint and focuses on topics that should be worked on by the development teams in the upcoming or in the next few sprints.

We often observed that too little time was allocated to the refinement over the whole development cycle of a product. As a mental support, it might help to think of a refinery when thinking of the term “refinement”: Here too, considerable effort must be made to turn a raw material (oil) into what we need for further use (gasoline).

An overview of the process and activities during a refinement can be found on the website of the LeSS Framework (https://less.works/). Anyone who wants to dive deeper into the methodology should start with the book “Large-Scale Scrum: More with LeSS” [LaVo17]. In chapter 11 you can find a description of how to run a refinement session and what to look out for.

For a regularly occurring refinement session (e.g., in each sprint), we recommend a list of standard participants with whom the topics can be broken down together. This list should consist of those who are responsible for the product (usually Product Owner/Area Product Owner) and the development teams and can be expanded as needed to include appropriate knowledge carriers. Each team that is potentially involved in the implementation of a topic should provide representatives in the associated refinement. The customer side in a refinement is typically represented by product managers and service or support staff. Whenever possible, real customers should also be involved in refinements at an early stage.

A refinement meeting follows a recurring pattern: At the beginning, there is first an idea or a "problem statement" that roughly describes the problem to be solved. Different information can be generated based on a scaffold that helps the refinement participants to understand important aspects and provides relevant information for the development teams. This refinement scaffold contains at least

- a description of how the product should behave for the customer after completion (story),
- an embedding in the existing system, e.g. at system component level, and
- many examples that illustrate the customer’s various interactions with the product.

These examples should always consist of a user, an executed action and the expected system behaviour. Based on our product definition for the garage, an example could look like this:

```
User: resident
Action: gets into the parked car in the rain.
Expected Outcome: The resident's clothes stay dry.
```

At the end of the refinement, the corresponding items are created in the backlog from the information generated. and the identified requirements are systematically documented in the corresponding tools.

Establishing a Common Language

Another recommendation, derived from the refinement activity, is to establish a common language for the refinement sessions so that all participants can make themselves easier understood. The more the language during the refinement resembles the customer’s language, the better. This common language quickly enables complicated interactions with the product to be presented in an understandable way. In relation to our garage example, the common language should enable simple representation of the weather conditions or where the garage is located in relation to the house (right, left, in front, behind…).

At the same time, we strongly advise against creating additional roles in the organization that do some kind of preparatory work for the development teams in the refinement. It is essential that the teams themselves understand the topics and break them down accordingly in order to be able to work on them. “Requirements Engineer”, “Feature Owner” or “Function Developer” are flavors that we have encountered over the years for such “water carrier roles”.

Complexity and Iteration: Working methods for creating knowledge

The complexity of the systems forces us to take an iterative approach, as by definition parts of the requirements (or at least feedback effects) on such a system are not apparent at the beginning of development and should be worked out in an interdisciplinary group. This dynamic in the elaboration is hindered by the extra roles mentioned above. Therefore, it is so important during the course of development that requirements are continuously identified (emerging requirements) and teams learn to integrate this into their regular activities.

To bring together as many perspectives as possible during the refinement, we need to bring together as many knowledge carriers as possible - even if this may feel inefficient. Typically, service and production requirements are integrated into the development too late (because they are not or late involved) and this often leads to high costs at the end of the product development process.

Since the format of the refinements, similar to a workshop, is based on the quick feedback between the participants, we recommend doing it “live” and continuously documenting requirements and activities. Since we want to identify requirements during the course of development, in addition to development tasks, tasks must also be in the backlog that aim at pure information or data procurement ("research activities"). These are even very important in order to generate requirements and validate their necessity in practice.

Applying it to the Example

In relation to our example with the house, it could look like this: Our first product version did not include a garage, and we have never built a garage or a garage door. To enable us to identify appropriate requirements for a garage door, it is helpful for these "research activities" to briefly jump to the solution level and look at what materials such a door could be made from (quickly iterate through problem and solution areas, see Double Diamond). Straw? Styrofoam? Sheet metal? It is not yet a question of building a product or, to put it another way, implementing a development request, but rather of first finding out as quickly and cheaply as possible what requirements there are for a garage door in our product context.

Why do we need a garage door at all to keep rain off the car of the residents? To investigate this hypothesis, we could build a relatively simple prototype and gather feedback from customers. After that, the garage door may remain relevant (e.g. for theft protection) or we may have found out that a roof between the garage and the house is more important, as the house should continue to keep the rain off the residents also on the way from the car into the house.

A product-related example would be requirements for the performance of an entire system. In order to define these in a meaningful way, it may be necessary to determine certain framework conditions at the customer’s site (network, server infrastructure, etc.). A research task in the backlog should therefore be to obtain this information. This can then be used, for example, to improve the test infrastructure - which can be used to validate changes in terms of performance close to the customer’s scenario. From this, meaningful values for the performance requirements can be derived.

Testing is Essential

What we want to make clear here: The backlog does not contain requirements, but the (development) work. Especially with complex systems, it can sometimes take a lot of effort to identify the necessary requirements. So if we have two different locations (tools), the question arises as to how we deal with such requirements and, above all, how we make them understandable to the development teams. Figure 2 illustrates the connection between the examples from the refinement framework, the tests that the development teams write and the requirements.

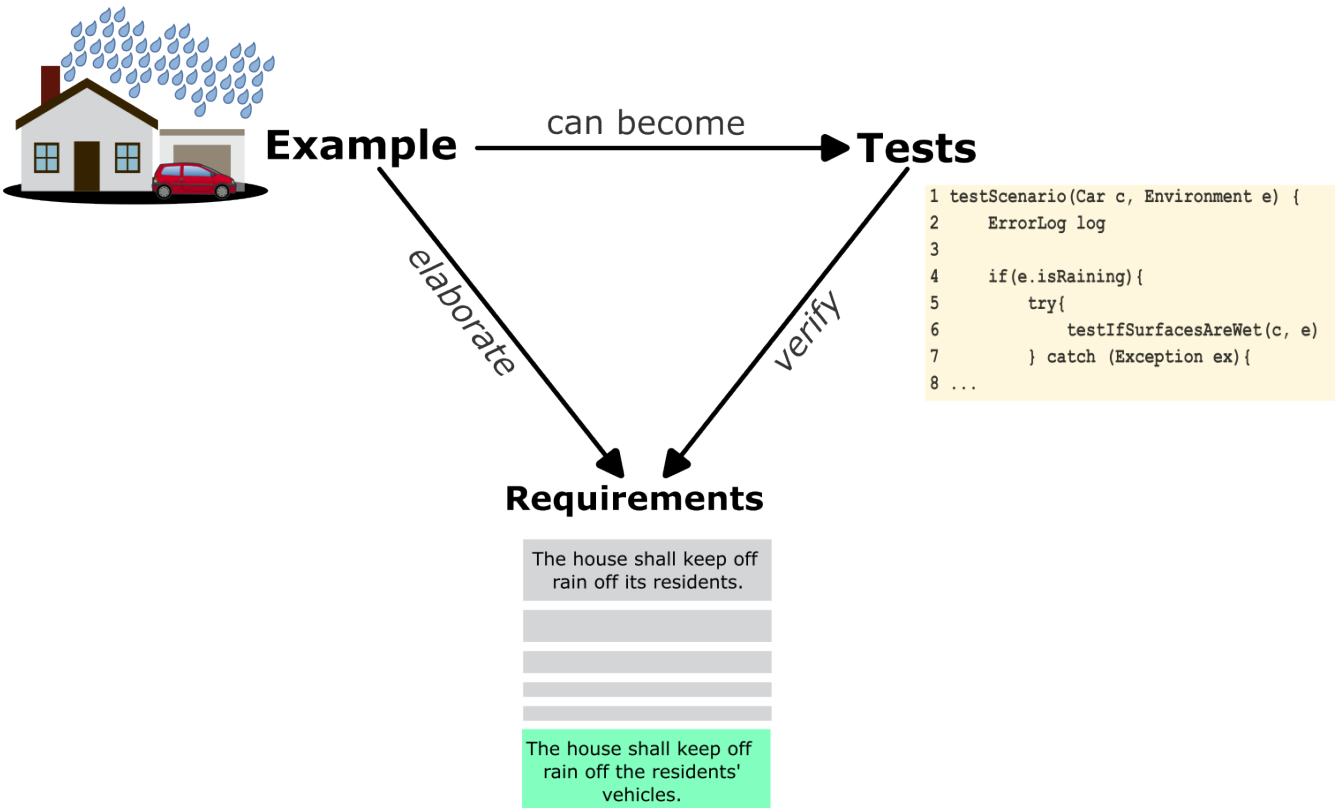

Figure 2. Relationship between examples, requirements and tests for a system.

The examples can not only help developers to better understand the requirements, they can also help to discover them. Gojko Adzic [Adzi00] is dedicated to this connection between examples, tests and requirements. In his book, he deals with the challenges of identifying good requirements. Adzic proposes an approach in which we use examples and feedback to arrive at a common language - and use it to describe what we want to achieve together in the end: the product. This approach is directly reflected in the refinement scaffold.

The image above also provides a hint on how to ensure that the implementation meets the requirements: Through the tests. The necessary information for these tests is identified in the refinement. For software, this can even be recorded directly, e.g., by using a corresponding syntax or formal language. These specifications, often referred to as "scenarios", can then be automatically converted into tests. The advantage for a developer is that the information only has to be written down once and no manual transfer between the specification and the tests has to take place. This also describes an essential difference between hard- and software, as the necessary information for hardware can of course also be identified in the refinement. However, the transfer to a test often has to be done manually. This does not mean that corresponding tests for hardware cannot also be

automated, it is just significantly more complex and therefore often associated with higher costs than for software.

These tests generated in the refinement enable us to verify the implementation of the requirements (even automatically for software). This is also referred to as "executable spec". A large number of publications (in addition to Adzic, see also Farley [Farl21]) describe what such tests can look like and what problems can arise. However, we do not want to ignore the fact that the implementation for the software is also more demanding than Figure 2 may suggest. Practical obstacles range from different levels of knowledge of the development teams about the overall system to incorrectly used concepts during implementation (mocks vs. stubs).

All of the described activities take place during refinement and before the development teams implement their items. This means that the refinement allows us to identify parts of the requirements during development. It also means that at least a portion of the associated tests is created along with the examples during the refinement, which implies they are created **before the implementation**. In summary, we recommend conducting the process of finding examples together with the according tests as part of the refinement sessions and using the introduced refinement scaffold for this purpose.

The tests created during refinement typically describe a different level of our system than component tests do. Methodologically, we are talking here about the difference between Test-Driven Development (TDD for short, where software development is controlled by tests) and Behaviour-Driven Development (BDD for short, a form of acceptance test-driven development). These different test levels should be known to all teams and be documented accordingly. Identifying these tests together with the requirements before they are implemented by the teams is an essential basis for clean requirements engineering - and for a good agile way of working (see Larman [LaVo10], and Adzic [Adzi12]).

Identify requirements in Workshop

One possible starting point for creating the foundation for identifying requirements is an experiment in the form of a series of workshops. These workshops should:

- happen live, in a large room with plenty of wall and whiteboard space so that ideas can be sketched out easily
- with participants from all teams, product owners and (where available) project and product management to ensure that all perspectives are represented
- take place in at least three consecutive sprints or weeks, not on three days in a row, as otherwise it becomes very exhausting for the participants and this has consequences for the quality of the discussion
- have an experienced and neutral moderator so that the participants can concentrate fully on the content

These workshops have two main objectives: On the one hand, we enable the teams and product management to jointly map goals from the planning for the product (roadmap) to specific activities of the teams. On the other hand, we want to ensure that the teams start to identify common denominators in the development activities and thus create a suitable structure for the product

backlog.

The format of the respective workshop days is flexible, we recommend working as much as possible with sticky notes on whiteboards and documenting the results of the workshop days, e.g. in the form of photos.

Over the years, we have repeatedly observed certain challenges for the participants when organising these workshops. We want to offer solutions to these challenges here:

Creating the Roadmap

Challenge: Creating a roadmap through product management is difficult and often involves significant effort. Sometimes, the vision for the product or its strategic direction is unclear.

Solution: Foster transparency by collaboratively documenting the product vision and strategy. Continuously work on this together with product management and teams. Consider using a business model perspective (e.g., Business Model Canvas) to help clarify the vision.

Customer Distance

Challenge: There's often a significant gap between product management, teams, and actual customers. To create a product roadmap, it's crucial to understand customer problems and product usage.

Solution: Establish direct contact with customers. For example, engage in joint refinements with the customer and provide product management with an easy way to try out the current development status (e.g., through one-click access to a test environment with current "release candidates").

Backlog Structure

Challenge: Teams struggle to find a common structure for the backlog. This issue often arises due to high specialization within teams combined with a strong component focus. This lack of overview is sometimes exacerbated by so-called "split heads", i.e. people who belong to several teams at the same time.

Solution: Orient teams strongly around the product. Ideally, teams should be cross-functional and interdisciplinary.

The complexity of the technology stack (includes all software, technologies and programming languages etc. used to develop the product) can have a similar effect if, for example, the same solution is implemented using several technologies in the stack. If different teams are each working on one of the technologies, it becomes very difficult to identify the common denominator. The only thing that helps here is to reduce the complexity of the technology stack, e.g. to use only a few or only one technology for certain problems.

Inclusion of All Disciplines

Challenge: Essential development work is not adequately represented in the workshop series due to organizational structures. Often, missing development components are located in other

departments or even separate companies.

Solution: In the short term, invite the missing teams to the workshops. In the long term, consider consolidating critical development components within a single organizational unit.

Conclusion

In order to successfully describe the requirements for a product, we recommend not only separating the product backlog and requirements, but also developing a common language using examples and investing more time in joint refinement. An agile way of working provides us with a procedure, methods and other tools for this journey.

A multitude of concepts and empirical knowledge are often hidden behind simple terms. This inevitably leads to different interpretations or even misunderstandings. In our experience, it is very helpful to be prepared to experiment in order to achieve a better understanding and therefore also better implementation. Or as Margaret Heffernan once aptly put it: "Failed experiments look inefficient, but they are often the only way you can figure out how the real world works".

Bibliography

- AAMI, AT: Guidance on the use of agile practices in the development of medical device software. In: Association for the Advancement of Medical Instrumentation Bd. 21 (2012), S. 22–78
- Adzic, Gojko: Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing - ISBN 978-0-9556836-1-9
- Adzic, Gojko: Specification by example: how successful teams deliver the right software. 2., corr. printing. Shelter Island, NY : Manning, 2012 - ISBN 978-1-61729-008-4
- Farley, D.: Modern Software Engineering: Doing what Works to Build Better Software Faster : Addison Wesley, 2021 - ISBN 978-0-13-731491-1
- General Principles of Software Validation - Guidance for Industry and FDA Staff.
- Guidance on the use of agile practices in the development of medical device software. In: AAMI TIR45:2023, 2023
- Larman, Craig ; Vodde, Bas: Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum. 1. Aufl. : Addison-Wesley Professional, 2008 - ISBN 0-321-48096-1
- Larman, Craig ; Vodde, Bas: Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum. 1st. Aufl. : Addison-Wesley Professional, 2010 - ISBN 0-321-63640-6
- Larman, Craig ; Vodde, Bas: Large-scale Scrum: more with LeSS, The Addison-Wesley signature series. Boston : Addison-Wesley, 2017 - ISBN 978-0-321-98571-2
- Newell, Allen ; Shaw, J Clifford ; Simon, Herbert A: The processes of creative thinking. In: Contemporary Approaches to Creative Thinking, 1958, University of Colorado, CO, US; This paper was presented at the aforementioned symposium. : Atherton Press, 1962
- Poppendieck, Mary ; Poppendieck, Tom: Leading lean software development: results are not the point, The Addison-Wesley signature series. Upper Saddle River, NJ Munich : Addison-Wesley, 2010 - ISBN 978-0-321-62070-5
- Reinsch, Daniel: TIR45: Agile Software-Entwicklung für Medizinprodukte. URL https://www.johner-institut.de/blog/iec-62304-medizinische-software/tir-45-agile-software-entwicklung/. - Johner Institut Blog