1

Agility: What Is That?

Ideas that began to form a decade ago among a handful of enthusiasts are today fundamental, one of them being agile product development using Scrum. Companies such as Immobilienscout24.de at least doubled their productivity due to the introduction of Scrum (Zeitler, 2011). These impressive results led large automobile and telecommunication enterprises to adopt the same product development method. Despite the success of the agile approach, there is one drawback: the realization that company processes that may have seemed to have in fact nothing to do with a project will now also need to undergo change. This may include strategic purchasing, key account management, demand management, and development departments. These processes need to be adapted to form a framework that will benefit significantly from the agile development process. Only when this is achieved can enterprises recognize the full extent of the enormous potential which results from their information technology (IT) teams adopting the agile approach.

Commercial-legal agreements with suppliers and partners are a substantial part of these basic conditions. They set the requirements that drive service providers to produce products faster and more effectively.

Agile Contracts: Creating and Managing Successful Projects with Scrum, First Edition.
Andreas Opelt, Boris Gloger, Wolfgang Pfarl, and Ralf Mittermayr.

Cannot Be Stopped

The U.S. software vendor VersionOne conducted a survey on the extent of agile method implementations over the past six years. Participants from the worldwide software development industry took part between July and November 2011 (VersionOne, 2011). Of the 6042 responses, the following can be concluded:

1. More than half of the participants questioned have already worked with agile methods for over two years.
2. Approximately one in five of those surveyed (17%) expressed the fact that they were aware that their company was planning a shift toward agile in the near future.
3. Almost two-thirds of those interviewed indicated that their companies complete almost half of all projects by using agile methods, and that three or more teams already work in this way.
4. For almost one-fourth of the participants (22%), time to market is the main motivation for applying agile methodology.
5. Those questioned rate higher productivity as one of the most significant advantages (75%). An even greater advantage is the ability to handle changing customer requirements (84%). Project progress also becomes more visible (77%).

It is not only the teams' employees who are losing their apprehension of agility. Management—despite a certain fear of scaling, legal regulations, and a lack of documentation—is more open to the agile approach. In two-thirds of the cases (64%), the initiative came from management. Interesting to note is that despite rising support by management, the largest hurdle in the conversion to agile methods is not knowledge of the methodology but of the internal organizational culture.

Let's us emphasize this again: **An understanding of the methodology is not the largest hurdle in a conversion to the agile model. Rather, the challenge is the internal organizational culture and the potential changes to the internal processes.**

Clearly, the organizational structure will not change from one day to the next. The following situation is characteristic: In one of our projects, the top management of a worldwide company decided to implement agile methodologies. So far, so good. Often enough it is management that is toughest to convince that the majority of self-organized and responsible work functions that have their own clear goals experience enormous leaps in productivity. However,

the employees, who should fulfill the order of priority set by top management, were forced to infringe on the rules given by the purchase department. By no means did the purchase department want to permit contracts with providers to be closed according to an agile model. Such a reaction immediately poses a question among employees as to whether an agile approach toward tenders has a future in the company and whether it can be sustained. How successfully the agile model is received by a supplier does not depend only on the supplier but also on necessary changes within your own company. Meanwhile, the company has found a new way of aligning successfully the set up and handling of agile projects together with procurement.

Traditional processes block the way for new requirements. Currently, most purchasing and sales processes are based on the traditional *waterfall* methodology (synonymous with predictive processes) for project management and product development. However, the waterfall process model has plenty of shortcomings, and IT services often attempt to exploit these in their business models. It is only natural that the purchasing departments of large organizations are against new processes, as such processes create defensive strategies and demand tougher contracts. Nevertheless, a project may fail to succeed.

A number of disadvantageous facts have been known for decades:

1. New products are not developed using the waterfall method, and effective projects are not organized based on that method. This was shown by Nonaka and Takeuchi (1986).
2. Even Winston Royce, creator of the waterfall model, said that this process does not work entirely and that it has to be carried out at least twice (Royce, 1970).
3. Studies carried out by the National Aeronautics and Space Administration in 1996 confirm the testimony of Barry Boehm, a software engineer from the 1980s, who calculated that estimates given at the beginning of a project life cycle (before the requirements phase) carry on average an uncertainty factor of 4 (Boehm, 1981). Thus, the actual time needed for a project could be either four times as long or only a fourth of what had originally been predicted, which seems very close to being unpredictable.

Especially in large organizations, we can find yet another prerequisite for setting up contracts with IT service providers that would have a negative effect on the productivity of software development projects: Every project has a budget. As a rule, these budgets have to be allocated very early—often even a year before the project is scheduled to begin. This means that departments define how much money should be spent before knowing the actual project goals of the company. Since no one knows exactly what the requirements will look like in 12 months, they are broadly defined. These requirements are requested from the service providers and judged as well as priced by the service providers.

Buyers, sellers, departments—all of them are doing their best to ensure that project costs do not get out of control and that schedules are met. Despite this, more than 60% of all IT projects fail. Many large IT projects extend their budgets by up to 400% and deliver only 25% of the desired functionality. Such "black swans" can destroy entire companies, as Bent Flyvbjerg and Alexander Budzier wrote in the *Harvard Business Review* of September 2011 (Flyvbjerg and Budzier, 2011). This also results in a tremendous loss for the economy.

[***Note:*** *The term* black swan *was coined by the financial mathematician Nassim Nicholas Taleb. He describes those events* (*both positive and negative*) *that are not only rare and unpredictable but also have major consequences that in retrospect were not actually so unlikely. Flyvbjerg and Budzier use the term only in its negative form*.]

There is as much discussion about the details of the analysis and facts of IT projects as about the actual success of IT projects themselves. And rightly so, because if we study the details of statistics or the background of an analysis, we will find a wealth of information that may shed a different light on various numbers. The bottom line, however, is that all studies agree.

Let's take a little tour d'horizon of some of the unpleasant facts:

- The *Standish Group* collects information on IT projects and their associated problems, and regularly publishes the Chaos Report. A project is defined as successful if time and budget constraints are met and if it complies with the required features and functions. However, some critical parameters are missing in this analysis, such as quality, risk and customer satisfaction. More important, and without focusing on the background details of the statistics as such, is how the measured success of projects has developed in recent years.

Findings of the Chaos Reports, 1994–2009

	2009	2006	2004	2002	2000	1998	1996	1994
Success	32%	35%	29%	34%	28%	26%	27%	16%
Partial success	44%	19%	53%	15%	23%	28%	40%	31%
Failure	24%	46%	18%	51%	49%	46%	33%	53%

Although the situation has improved significantly over the last 17 years, the percentage of successful projects is still well below 50%. According to the study, let's have a look at the most common causes for the failure of IT projects (Standish Group, 2009):

1. A lack of reliable input from users (2009: 12.8%)
2. Incomplete requirements and specifications (2009: 12.3%)
3. Changes in the requirements and specifications (2009: 11.8%)

To phrase it differently: The hit list for these stumbling blocks would be as follows:

1. Lack of cooperation
2. Missing knowledge of the requirements (the customer does not know what he or she really wants)
3. The fact that what is partially unknown is usually also described incorrectly or incompletely

- Unpleasant findings on the project's success can also be found in a study by the *Technical University of Munich* (Wildemann, 2006): Only about half of all IT projects of the period examined were successful. Either the projects took longer than planned, cost more than expected, or emerged with different results. Other projects actually had to be canceled and the costs had to be written off. The renowned Viennese IT expert Walter Jaburek (a court-certified expert on information technology and telecommunications) said in an interview that we conducted with him:

  Here is my experience congruent with the study by the Standish Group: Most projects take three times as long as planned, thereby costing approximately 2.8 times more and bringing 70 to 80% of the planned functionality. The contract and the negotiating skills decide who carries the additional 180% of the costs.

- In 2007 *Assure Consulting* reported that most IT projects fail due to unclear goals, unrealistic time constraints, and lack of coordination of the project participants (Assure Consulting, 2007).
- The consultants at *Roland Berger* reported that 20% of all IT projects are canceled. Every second project exceeds the agreed-upon time frame or is more expensive than planned. Vital is the indication that the probability of failure increases with the duration and complexity of projects (Roland Berger Strategy Consultants, 2008).
- A fall 2004 Forrester survey indicated that nearly one-third of customers are dissatisfied with the time it takes to deliver the custom applications requested (Forrester, 2005).
- Various studies and the experience of experts give reason to believe that requirements of IT projects change by up to 3% per month. As is referenced in the literature over and over again: The requirements for a software solution cannot be described deterministically. This can be summarized with the observation that, typically, 35% of all requirements change during a software project (Schwaber and Sutherland, 2012).

In your own daily practice you may also experience the fact that IT projects often take one step forward and two steps back. This could be for one or more of the following reasons:

- There is not enough user input.
- There is no simple, clear vision that describes a project's purpose.
- There is very little teamwork.
- Projects are becoming increasingly complex.
- The technologies used in projects have become more versatile.
- Systems have increasingly become more widely distributed.
- Functional and transparent monitoring of progress is often not possible.
- Experts on all sides (supplier, consultants, and customers) find it increasingly difficult to predict potential problems.
- The planning of projects is often very complex, sometimes almost impossible.
- Knowledge is poorly distributed.

We need to put a stop to this immense waste of resources, time, money, and creativity. This was first noted by some software developers in the 1990s—without trials or lengthy discussions. It was simply due to their own suffering that they discussed ideas for new project management and the development of new methods to allow teams, together with their project managers, to deliver continuously.

1.1 THE AGILE MANIFESTO OF 2001

To indicate at what points there is a rethinking of the purchase and sales processes and how this affects the setup of a contract, we follow the popular agile management framework: Scrum (in the State of Agile Survey by VersionOne (2011), Scrum was named as the method of use by 52% of respondents). Scrum is a perfect example of agility as we understand it. It is not merely just a method. It is based on very specific values and principles of cooperation that are aimed primarily at self-understanding of development teams. Of course, due to its strengths, it also affects the relationship between customers and service providers. Let's start our journey through the agile methodology at the origin, the *agile manifesto*. In the winter of 2001, a few representatives of the agile movement got together to discuss how they should promote the emerging trend in software development to reach as many people as possible. They also wanted to clarify what agile software development methods could actually achieve. It turned out that there are deep beliefs or even values that define agile's capabilities.

Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others to 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, although there is value in the items on the right, we value the items on the left more (http://www.agilemanifesto.org).

What do these values mean in more detail?

Individuals and interactions over processes and tools

Once again, look at your own project practice: How often do you realize that just talking to others can create shortcuts and solve a problem? That working together more effectively helps reach goals in a shorter time? How often do you experience that the available rigid processes are actually more of a hindrance than a help?

All agile development processes assume that to deliver a product it is essential that the team members and all other stakeholders talk to each other and exchange ideas constantly. For self-organization, it is essential to respect and recognize that individuals differ from each other. Obviously, teams with clearly defined processes and good development tools are productive. However, processes and tools are by no means more important than the individuals and their interactions.

This statement is often misunderstood and is interpreted as if team members are suddenly allowed to do everything. It is understood, mistakenly, as if all the walls are broken and, for example, no external influence or requirements should be provided to Scrum teams. Management feels especially strongly about hierarchical organizational cultures and is threatened by this aspect of the agile manifesto. But this is obviously not the case. Many developers have this attitude when they first come into contact with Scrum, but Scrum does not tell them how they should work. In Scrum, it is assumed that developers use their common sense and do everything necessary professionally to deliver the product. The principle of this thought is the essence of self-organization. **The nature of self-organization is that within a clearly defined framework,**

creative freedom is allowed and that this is actually the only way in which creative freedom can happen.

There are, of course, requirements and guidelines that must be followed. You can certainly not build a car and say: "Let the team get on with it and we'll see what comes out at the end." In the present day, no one would do that. Of course, cars must be built in a way that takes all legal guidelines and physical circumstances into account.

The next misconception: The customer is no longer able to define what he wants in a Scrum project. This, too, is nonsense. All of these misinterpretations have, contributed to the history of Scrum failures, as people who wanted to use Scrum were faced with these misinterpretations. Tragically, for this reason, Scrum projects often did not show the success that was possible. Rather than looking closer, the simple conclusion often was that the method was bad.

Tip: *Accept the statement of the manifesto as it stands. It states the belief that people are successful only if they communicate with one another, and that this is especially effective when using tools that allow them to achieve their results faster.*

Working software over comprehensive documentation

No statement from the agile world has been and will probably continue to be misunderstood more frequently than the one regarding the value of software. That it is gladly and consciously depicted incorrectly makes development teams open to attack. We hear again and again from customers and partners that teams do not document anything, due to the fact that they are using Scrum. Let us consider again the underlying problem: Do you enjoy documenting? Do you enjoy writing reports, and are you passionate about writing notes on the course of events? How many documents are not relevant because they were written solely for the filing cabinet? And let's face it, particularly in the software development environment, an excessive number of pages of documentation are produced.

However, this is not the reason that good or improved software is written. Most people see documentation as a useless by-product that is inevitably not going to help to make their work more meaningful and of higher quality. We're not talking about cases where people are sloppy at work. That obviously does exist. No, it's about making it clear. Documentation is only useful when it enables a person to handle a task or understand a topic faster and more efficiently.

Obviously, there are documents that we consider meaningful and that are necessary. For example, a doctor's note is necessary in order for hospital personnel to know how to help a patient. The plans of an architect, which support work at a construction site, are useful, meaningful and above all, necessary. In software development, this could be documentation that allows the customer to pass the work on the current software increment to another software provider (e.g., an incrementally growing document on the high-level architecture

of the software). This may be necessary if the relations have weakened for the initial service provider or the service provider decides to discontinue product development. This documentation is also useful and helps to make sure that we can continue from where others have left off.

In conclusion: *Necessary documentation must be produced, either by the Scrum team itself or by the Scrum support teams from within a large development department.*

What does the agile manifesto statement say? At the end, a project's success should not be based solely on whether the plan was provided or whether a doctor's diagnosis is present on paper. The document is not the product. Thus, the measure of product success is not about whether the documents were written correctly according to a process. Very often we have seen projects which were in trouble even though the first steps in the waterfall process, which delivered hundreds of pages of requirements and detailed design were present. However, we know of hardly any customers who stopped projects that delivered high-quality software even if the number of "documentation pages" was minimal!

Customer collaboration over contract negotiation

The next argument: No, the third principle does not mean that no contracts should be concluded or negotiated. Why would we write a book like that if this were the case? We even present in Chapter 7 many details on how to negotiate a treaty framework for the agile approach. This is, instead, how we interpret this basic principle: Of course, you need contracts. Clearly defining *together* how you want to collaborate is useful: to regulate how payment is to proceed and how much is to be paid; to think about what happens if one of the parties no longer cooperates as originally agreed—all that is necessary and must be done.

It appears, however, that the best contracts do not always lead to the fact that they also shared the success of the project. Especially in the software industry, IT and software development departments like to be regarded as service providers. The software suppliers are typically pushed into a corner when it comes to service, and here they remain with too little information to perform their task purposefully and successfully. However, it appears that software development projects are successful only when those who write the software and those who require the product work closely together. It is clear that customers get the products they really need only if they involve themselves actively. They must make themselves available as partners during a project. The functionalities, which facilitate productivity, are obtained when the software development teams are assisted. From our own experience we can say that if the parties work together and want to succeed, the probability is extremely high that the product delivered will be satisfactory.

Thus, it is important to express the duty to cooperate fully and to emphasize the cooperative approach without placing the responsibility for quality on the

contractor. If you read the agile manifesto carefully, you'll see that it does not say that the item on the right is not useful. It says that item is valid and useful; however, the item on the left is even more important from an agile perspective!

Keyword respect: *The former indicates what this principle wishes to express. We are respectful of each other. The customer should not pressurize the service provider, and the service provider should not attempt to mislead the customer.*

On the Agile Tour 2011 in Vienna, Mitch Lacey, an agile practitioner and consultant, told the following story about a conversation between customers and suppliers. A client came to him and explained his project in half an hour. He then proceeded to ask what such a project would cost. Mitch replied:

> This is a question I cannot answer because you should expect a professional response from me. I do not have enough information after 30 minutes to be able to make a meaningful statement. That would be totally unprofessional. I'll make another suggestion: You work for two weeks with us and if you like what you get, then you pay for these two weeks. If not, then you don't pay. And so we continue. You pay when you are satisfied with the work we deliver. You could of course abuse this principle, since we obviously cannot exclude the functionality, which you are not satisfied with and for which you have not paid, from the product. New functionality is developed on top of the supplied functionality of the last iteration. In this case you would pay for the development in the first two weeks, then you would not pay for the next two weeks, and then you would pay again and so on. This would cut your costs and at the end you would have the finished product with all the functionality at half the cost. However, we would note in this case that you had not dealt with us fairly and we would have to stop working.

This way of dealing with a customer who we may not yet know minimizes the risk. It is also a successful practice to be able to start from a basis of trust and to respond if the trust is broken (this is called the *tit for tat strategy*). The beauty of agile projects is that at the end all that counts is what is actually delivered.

Incidentally, this is a first indication of how contracts should be designed. It is expected that the result desired would be that software is delivered and not a question of whether intermediate steps were generated successfully.

Responding to change over following a plan

Reading this head, our attention jumps immediately to the last word: plan. This wording is interpreted by many to mean that in agile projects and agile product development, there are no plans. As if there would only be chaos: No one knows what he or she will get and no one can say how expensive a project or product will be.

This interpretation is obviously wrong. With agile projects, you plan more frequently and more concretely than with traditional methods—on a total of five levels:

- On the level of the vision
- On the level of the road map
- On the level of release
- On the level of the sprints and iterations
- On the level of daily work

Agile methodologists have developed countless planning procedures and tools, beginning with clear conceptions about how you produce a vision and how you subsequently produce release plans. There are concrete procedural instructions on how to arrange sprint planning and much more. At all levels, the participants are aware that each of these planning activities will be repeated iteratively and that the plan must be adjusted continuously. The development team plans every day to collaborate to achieve the sprint goal. During the sprint, the development team and product owner discuss (or to phrase it differently, they plan in cooperation) how the next sprint will be executed. At the beginning of a release, the Scrum team and customers discuss what they would like to see produced in the forthcoming release.

During the current release, the product owner and the customers talk about how the product is to be developed further in the long run. The product road map and the vision of the product are tested in the market, and if necessary, a more sustainable vision for the product is generated. Ideally, the entire planning process is therefore completely transparent. Recently, Pries and Quigley (2011) have compared Scrum to standard project management techniques, with the result that there are comparable planning mechanisms in this agile methodology.

Each of these planning processes has its own visualization techniques and presentation methods. For the plans to proceed as effectively and as rapidly as possible, the communicative process among the various parties must be coordinated. In agile development, not having a plan is not an option.

1.2 AGILE DEVELOPMENT BASED ON SCRUM

Scrum is now the de facto standard in agile software development. In recent years it has evolved from a project management methodology to a new understanding about how to manage dysfunctional working teams, departments, entire organizational units, and companies [in software development and even in industries such as education (Pries and Quigley, 2011)]. Typically, companies use Scrum initially at the team or project level as a project management method. Some companies continue to use it that way. Others evolve even further and shape the life cycle of their entire organization with Scrum. Basically, it is not a method of software development but a management framework within which software development, in whatever way and by whatever means, is taking place (Table 1.1).

TABLE 1.1 Agile Development Methods Within an Agile Framework

Agile Development Method	Agile (Management/Process) Frameworks
Adaptive software development	S
Agile data warehousing	
Crystal	c
Dynamic system development method	
Extreme programming	r
Feature-driven development	
Software expedition	
Universal application	u
Usability-driven development	
Kanban	m

With Scrum, there are a few principles and just a few rules. However, these rules and principles are adhered to strictly. Completely in accordance with the agile manifesto, Scrum is based on another conception regarding people: Scrum respects each person involved and the subject is perceived of as a literate member of the team. Scrum is designed to give teams freedom so that the talents of its members are exposed and so that an enjoyable and productive working environment can exist. In our interpretation, Scrum enables each team member to regain the skills and competence they need to take over responsibility.

The opinion that Scrum is a development method is a common misunderstanding. That Scrum leaves unrestricted freedom to team members is another misunderstanding. Agile approaches such as Scrum are based on a meaningful interaction of rules, discipline, personal responsibility, thinking together, assisting each other, and not using your knowledge simply to shine personally.

- **Product development vs. project management.** Scrum is not about producing a predefined end result but, rather, a steady stream of product parts, resulting in an overall end product. Using Scrum, the developers create a product that approximates the future by the constant inclusion of current changes. By default with Scrum, the applicable software is developed at the end of each sprint. In addition, this results in *continuous intermediate products*. These products make it possible to start risk-free and to measure the progress of a project based on the product parts already supplied. This is the real reason that agile software development projects do not conform to traditional contractual concepts. Something is always delivered, which is not the case with traditional project management methods.
- **Management framework vs. development methods.** Scrum has no regulations on how to work, but sets out roles and responsibilities very clearly. It also defines very clear limits for the development part. The principle of the time box produces not only creative pressure but also the security that is necessary for the development of self-organization.

- **Product owner vs. project manager.** The term *project manager* does not exist in Scrum. A *product owner* is a product visionary who can deliver the product idea to the team, thereby encouraging the team members to productivity. The product owner is responsible for the financial success of the product but, in contrast to a standard project manager, also, has a deep understanding of the client's domain and requirements.

Before we take a closer look at Scrum, we need to resolve the three greatest misconceptions.

1. **Scrum is based on temporary thoughts and hence involves no planning.** Planning in Scrum is carried out consistently and strictly on three levels: at the daily level (daily Scrum), at the sprint level (sprint planning), and at the release level (release planning). Scrum follows the Deming cycle, the basic idea of continuous improvement and permanent planning following the motto plan–do–check–act (Deming, 1982).
2. **Scrum promotes unprofessional work.** This view has its foundation in a world where freedom is perceived as a threat to problem solving and where inches-thick documents are considered quality criteria for good software. Scrum allows freedom of creativity for the team and prescribes in no way *how* a problem should to be solved. In sprint planning, you dog however, specify *what* needs to be present at the end of a sprint. If documentation is deemed to be necessary, it will be included at the end of the sprint. Scrum reveals unprofessional work relentlessly. Due to the daily Scrum, unprofessional work by individual developers is visible to all team members.
3. **Agile methods and Scrum are not disciplined.** Agile processes are extremely consequential in their implementation, as each person's actions become permanently visible and obvious. Discipline in Scrum is, in fact, so extreme that every meeting starts on time and to the minute and whoever is not present must remain outside.

The values of Scrum

It is quite clear that people's perception of the agile manifesto and consequently of Scrum is different from that of the command receiver and the executing aides, who work according to a strict plan. **In Scrum, we assume that intelligent people have a fundamental interest in contributing their ideas to improving things or even to developing new things** (see, in addition, the X and Y theory of Douglas McGregor). Those who promote Scrum are of the opinion that we are all adults and are therefore basically responsible for our own actions. In Scrum we believe (and know) that people give their all if they are fascinated by a vision. *Commitment, focus, openness, courage*, and *respect* are therefore the values of Scrum on which the thinking of those who work with Scrum should be based.

1.2.1 The Principles of Organization

The concepts on which Scrum is based, require a different form of organization. In principle, the procedural principles of the Toyota production system are transferred to software development.

- **Small, self-organized, and cross-functional teams.** Ideally, a Scrum team is made up of seven people: the scrum master, the product owner, and the five members of the development team. The members of the development team do not rely solely on their own specializations but are able to perform various tasks in the work process (according to the concept of the *T-shaped person*; see, e.g., Reinertsen, 2009). This means that they can share their knowledge with each other, applied in various combinations, and they have no fear of tasks that do not correspond directly to their core competencies. They organize their tasks entirely by themselves.
- **Working according to the pull principle.** The team has sole authority to decide how much labor and how many product parts can be delivered during a sprint. The team has control over how much work it receives. (For a description of the pull principle in production, go to http://de.wikipedia.org/wiki/Pull-System.)
- **Intervals with a clear time limit: the time box.** The team gets challenging targets, which are specified at intervals with specific time frames. All actions are limited in time and require a result. This creates a clear framework.
- **Useful business functions: potential shippable code.** At the end of each time interval, the team must deliver a product that meets the standards, guidelines, and requirements of the project.

1.2.2 The Process Model

The Scrum process model defines the framework for running all the activities of product development (Figure 1.1). Besides the six roles outlined above, the Scrum process consists of six meetings and 12 artifacts, shown in Table 1.2.

A weakness of traditional development methods is that they separate customers and developers. This causes a separation between the strategic and the tactical-operational levels. As a result, the team knows that it should do something, but not why. Knowing why is background information essential to developing innovative approaches to problem solving. This fact has long been known. For example, Richard Feynman observed during the Manhattan atomic bomb project that his team's productivity increased extremely as soon as they were given more information about the "why" (Feynman, 1985).

Software developers in traditional processes are usually focused strongly on their work and ignore existing medium- to long-term business issues. With Scrum, however, the software developers are included in the strategic

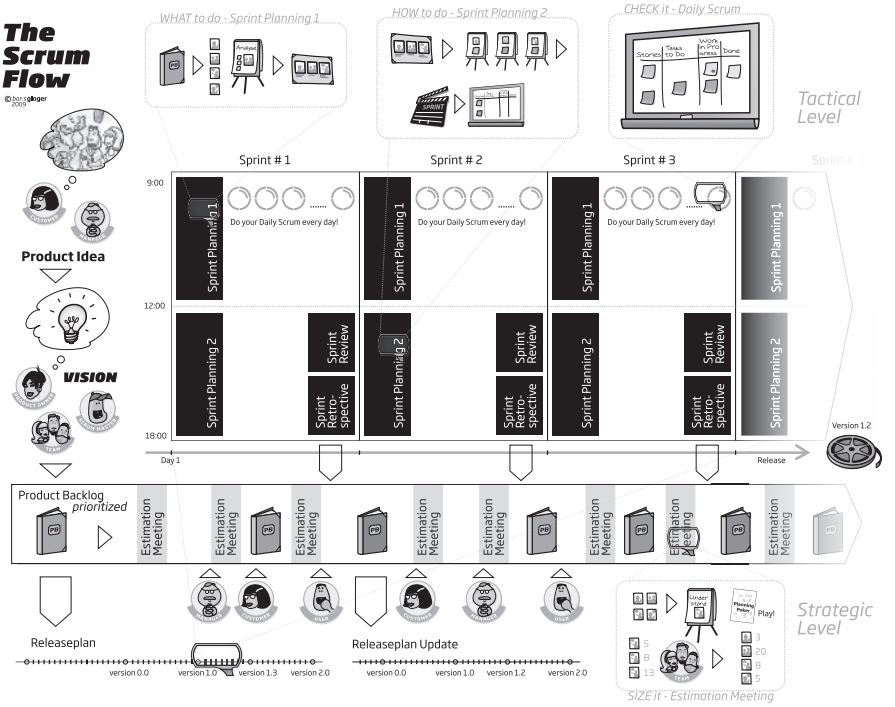

Figure 1.1 *The process model:* "Scrum Flow."

TABLE 1.2

Roles	Meetings	Artifacts
Team	Estimation meeting	Product vision
Product owner	Sprint planning 1	Product backlog item (story)
Scrum master	Sprint planning 2	Product backlog (list of stories)
Manager	Daily scrum	Sprint goal
Customer	Estimation meeting	Selected product backlog
End user	Sprint review	Tasks
	Sprint retrospective	Sprint backlog
		Release plan
		Impediment backlog
		Product increment: usable software
		Definition of done
		Burndown chart

considerations in the following two ways, and developers begin to understand the context in which the success or failure of their work affects their firm and its customers.

- On the one hand, the product owner develops a product vision for the product, either alone or together with the team.
- On the other hand, the team always becomes involved in later strategic planning.

In these two substrategies, product line strategy and organizational strategy higher-level strategies are obviously also factored in.

Strategic planning provides us with a perspective from which, to assess whether a project can succeed, and with the ability to decide which approach will lead to achieving the goals. In summary, we are planning the following:

- On the strategic level, the goals that we want to achieve
- On the tactical level, the actions that are necessary to achieve these objectives

The roles

The strength of Scrum lies in the clear allocation of responsibilities and the separation of responsibilities of Scrum master, product owner, and team. In practice, to strengthen the situation intellectually within the teams or in an organization, we add the roles of customers, end users, and managers.

- **The development team: the suppliers.** The development team delivers the product. It manages its own affairs and is authorized to do anything goal-oriented that is necessary to achieve the desired result. This is done while complying with the standards and procedures of the organization. The team itself controls the amount of work that it can handle and therefore accepts responsibility for the quality of the delivery.
- **The product owner: the visionary.** The product owner steers the product development and is responsible for ensuring that the team develops the desired functionalities in the correct order. He or she ensures that the project results justify the financial investment for the project. The product owner works on a daily basis with the team and takes all necessary decisions in a timely manner. He or she is working continuously on the product backlog and the release plan.
- **The Scrum master: the change agent.** The Scrum master helps the team achieve its goals. He or she works to ensure that all the difficulties, obstacles, and problems that are present are solved. Although not authorized to give instructions, this person ensures that the Scrum process is followed. One of the main tasks of the Scrum master is to educate all persons involved in the project so that they can understand and carry out their roles.

- **The manager: the provider.** Management provides resources and guidelines within the organization. It creates the framework within which the team, the product owner, and the Scrum master may move. Management often solves the problems identified by the Scrum master.
- **The customer: the financer.** The customer is the requester of the project; he or she buys it or has been ordering the project. Typically, executive managers in organizations buy products from external companies. In an internal project development team, the person responsible for the budget often has the role of customer.
- **The end user: the user.** The user of the product is an essential source of information for the Scrum team. He or she is the one who will eventually use the "usable software." Therefore, the Scrum team includes the user in the product development process. During the sprint planning, the user collaborates with the product owner to define the requirements. Later, he or she will work together with the team to ensure that the application is deliverable.

Scrum on a strategic level

- **Develop a product vision.** Initially, a team member is faced with a product idea that is often introduced by the customer: the product owner. He or she handles this idea until there is a product vision. The product vision includes the basic idea for the project, including the necessary contraints, which are envisioned from the start.
- **Create a product backlog.** The product owner develops, either alone or with the help of team members, the product functionality (product backlog items). These items are listed in a very simple form: the user stories (or contextual groups of user stories, called epics). A *story* is a short sentence that represents a part of functionality in a special way. It is described by Mike Cohn (2004), who establishes the following structure for user stories:
- As a user with a role, I want a function, so that I can get benefits.
- **Example.** As a bank customer, I want the ability to identify myself, so that I can retrieve my customer data.
- All user stories (or epics) are included in a list, the product backlog.
- **Order the list by priorities.** The product owner places the items in the product backlog list in order of expected financial gain from the respective functions (http://www.scrum.org/scrumguides, last accessed on 6/3/12).
- **Hold an estimation meeting.** Next, each product backlog item must be valued as to its size. The estimation is performed by team members. A Scrum team includes all those people who are necessary to ensure that the backlog items are transformed into software that can be delivered. The team members estimate the extent of each product backlog item to be delivered and communicate the result to the product owner (Gloger, 2011).

- **Estimate and prioritize the product backlog.** The product backlog estimate is now completed. All team members have an idea of what the product should look like, and the product owner has a first impression of how much effort it will take to create the product.
- **Determine the velocity.** To determine when something can be delivered, we need to know the order and the size of the stories as well as the capacity (i.e., velocity) of the team.
- **Create the release plan.** With the capacity of the team we can calculate the duration of the project. Assuming that the team will remain as it is now, we can determine the number of sprints and, consequently, specify release dates for the various stories.

Scrum on a tactical level

The actual implementation phase in Scrum is carried out in clearly defined time intervals, the sprints. Until the end of each sprint, the team must do its best to deliver functional and quality software (*potential shippable code* or, as termed more recently, *usable software*). At the beginning of a sprint, the tactical implementation is discussed, based on the plan that was developed in the strategic planning phase. On the basis of rough ideas about what features (user stories) are to be delivered in the respective sprints, it is now decided how much can actually be delivered in this sprint. A sprint covers a maximum period of 30 days and is divided into a series of (planning) workshops: sprint planning 1, sprint planning 2, daily Scrum, estimation meeting, sprint review, and sprint retrospective.

- **Sprint planning 1: explaining the requirements for the sprint.** In this first sprint workshop, the product owner, the team, the management, the users, and the Scrum master are present. The product owner explains the stories and, together with the team members and management, defines the goals for the upcoming sprint. The stories are then selected suitably, according to the goals and abilities of the teams to deliver them. This is how sprint backlog develops (in accordance with the Scrum Guide, http://www.scrum.org/scrumguides, last accessed on 6/3/12).
- **Sprint planning 2: design and planning.** Here the team members plan, together with the Scrum master, how they will meet the target agreed to in sprint planning 1. They advise each other on how the application should be structured, what architecture should be selected, which interfaces should be written, and whether test cases should already have been created and written. In short, they discuss in detail what needs to be done.
- **Daily Scrum: coordination and feedback.** Every day the team members meet (the product owner may also participate) at the same time in the same place for 15 minutes for a daily planning meeting moderated by the Scrum master. Here, each team member selects the task that he or she will work on that day. The team members inform the Scrum master of

obstacles and problems so that he or she can solve them as quickly as possible.

- **Estimation meeting: advance planning and estimation.** The product owner and team members update the product backlog at least once during the sprint. Thus, stories containing new estimates are provided and new stories are included in the product backlog. At the same time, the order of the backlog items is adjusted, to take the new information into consideration. This meeting allows the product owner to update and complete the release plan of the project.
- **Sprint review: presenting the results.** At the end of the sprint, the Scrum team presents the stories that have been developed. The team shows only the stories that are really complete, that is, that could be put into production.
- **Sprint retrospective: constantly improving.** The sprint retrospective enables the team to learn systematically. In this stage the team analyzes which processes must be improved in order to work more effectively. The results of the retrospective are captured in the impediment backlog and contribute as suggestions for improvements in the sprint planning.

The key principle is: *At the end of a sprint, the development team must provide potentially useful functionality. This means that no further work is needed to pass this functionality to the end user. This principle must be adapted to the respective conditions of development. Therefore, the level of completion is agreed between the development team and the product owner* (*the "definition of done"*). *The Scrum master is working continuously with the Scrum team to improve the efficiency of the teamwork. Ideally, the end user receives a tested delivery at the end of the sprint*.

Scrum can help companies to compete in the global marketplace. It does so by designing the software product development to be more responsive and problem-oriented rather than just a work-through project. The principles of Scrum, the roles and the process framework, create structures and rules by which to guide the staff but at the same time to give them the freedom to develop their potential. In this way, they find new, innovative approaches and begin to think beyond their horizons. The fact that Scrum is much more than simply a single method is usually not recognized by companies until they are already working with it.

1.2.3 Estimation in Scrum

Estimating the complexity of the functionality to be delivered is an essential part of the cost negotiation and implementation of projects. This is not special for Scrum, but the way this is done within an agile framework agreement is different. To prepare you for later work as to how estimates in Scrum are analyzed and prepared, we provide here an explanation and basic overview.

In the real business world there are many reasons why the product owner must create a release plan. This plan should show at which point certain sets of specific functionality are available. To create this plan, the product owner needs three pieces of information:

1. The size (in terms of complexity) of the backlog items.
2. The prioritization: that is, the order of the backlog items in the list of functions.
3. The capacity of the Scrum team: that is, the number of backlog items (counted in complexity points or story points, which are discussed later) that the Scrum team can develop in one sprint.

If these factors are available, the product owner can very easily calculate at which point the functionality will be available. The problem is that this information is not known at the beginning of a project. Therefore, a way must be found to estimate the size of the backlog items and the capacity and/or velocity of the Scrum team.

Predictability and estimates

Why is estimating functionality so problematic? The answer is because estimating as we generally think of it estimates the wrong thing: namely, the effort. When estimating a project, we must distinguish between:

- Estimating the effort
- Estimating the size (i.e., the complexity)

Very often, the estimate of the size of the functionalities is confused with the estimate of the effort. It is understandable that a project manager would appreciate an estimate of the effort, because this gives the project sponsor information about the cost of the project. But if estimates are based on effort, this also means that the project plans must be based on an estimation of the activities.

In software development, where a more productive programmer is up to 25 times as effective as a weaker one, it is impossible to predict the time taken for a particular programming task. What is even more extreme is that there is no correlation between the time required for a particular task and the end result. Even if the project manager asked the developer for an estimate each time a task was performed, he or she would still have the problem that the same developer does not always complete the task. Estimating in Scrum means to estimate the size, not the effort.

In Scrum, the team performance is measured by its *velocity*, the amount of functionality that a team can provide in a time unit or sprint. In other words, the velocity is an expression of the throughput (measured in terms of the size of the functionality) of the team. The more a team gets done during a sprint,

the higher its throughput (i.e., velocity). If the product owner knows the measured velocity of a team, he or she can accurately calculate when a specific product part is finished.

Estimates with story points

Before we can determine the size of the backlog items, we need to establish what size means. Size refers to the degree of understanding that a team has of the functionality of the backlog item. The more accurate the understanding, the smaller the associated size.

To estimate the size of a backlog item, we require only three things:

1. First, we need a *reference*. We obtain this by selecting an item from the list of backlog items, which at first glance appears to be small and manageable. While browsing, the team together determines which properties the backlog item has, thus establishing the reference. The reference represents the dimensions or aspects that the team needs to use to determine the size. If, for example, we want to determine the size of countries and we had a list of all the European countries, we might first look for the smallest country. For this we would first have to agree on the properties according to which we determine size. We could take into consideration the area as well as the size of the population. We could, of course, also use a combination of many properties. The factors incorporated in the determination of the size are set when estimating backlog items. Here you select the dimensions that best help you understand the backlog items. The responsibility for setting the dimensions of the unit lies completely with the team. The reference should express all relevant aspects.
2. If you have agreed to a reference, that is, a backlog item that appears suitable as a reference, the next step is to establish the *unit of measure*. The unit is simple in our case. We need something that expresses the size of a backlog item. The agile community has agreed to call this unit of measure *story points*. This is completely arbitrary; you might as well count gummy bears, as long as you are aware that we are dealing here with the designation of a unit.
3. Finally, we need a *scale*. Scaling is difficult because it leads easily to misinterpretations. We are dealing with estimates of relative size (i.e., the relative understanding of the functionality that will be generated). We therefore require a scale that takes into account the fact that estimates have larger fluctuations when big things, which are often associated with a greater lack of understanding, are estimated. In other words, an estimate will be more accurate if we are dealing with a small, manageable package than if it is a very large package.

The agile community has, not least thanks to the work of Mike Cohn (2005), selected Cohn's impure fibonacci series as the agreed-upon scale (Table 1.3).

TABLE 1.3 The Impure Fibonacci Series According to Cohn

Step	1	2	3	4	5	6	7	8	9
Value	1	2	3	5	8	13	20	40	100
Standard deviation at 50% accuracy	0.5	1	1.5	2.5	4	6.5	10	20	50

This scale already indicates to us, simply through its values, how "accurate" the estimate is. A high value means automatically that the standard deviation is higher. The estimate is therefore not inaccurate, but the range in which our backlog item is located is much larger. We will see that we use this property to schedule the release plan.

Now we have everything that necessary to estimate the backlog items: a reference, a unit, and a scale. At this point we invite the team to an estimation meeting. As this can sometimes be a rather large meeting, we have to perform as efficiently as possible.

Planning poker

Planning poker generates estimates in a relatively short time, based on expert opinion, and it also makes the proceedings entertaining. The use of Planning Poker in the estimation process is so effective because it uses the intuition of experts and helps to avoid the communication problems that every group of experts experiences. Planning poker is "played" by all the Scrum team members. It is, in fact, important that all team members (i.e., software developers, database engineers, testers, business analysts, and designers) estimate the backlog together. In an agile software development project, that usually involves no more than 10 team members. If the team is any larger, you should split it up and carry out the estimation in two teams. It is crucial that the product owner be present, but he or she has no right to estimate.

Planning poker is played with planning poker cards. To prepare in advance, each team member is given a set of "playing cards" with the values of the impure Fibonacci series according to Cohn (i.e., 0, 1, 2, 3, 5, 8, 13, 20, 40, 100). When all team members have their card set, in the next step they agree on the reference backlog item or call to mind once again what the reference backlog item was in the last round of estimations and which features were relevant in the assessment.

Once the reference backlog item has been found, the actual estimation process begins. The moderator of the meeting then reads out loud the description of the backlog items that are to be estimated. Any questions about the understanding of issues for this backlog item are answered at this meeting (where relevant, these answers are used to extend the description of the backlog item). When all questions have been answered, each team member selects a card which represents the value that this team member believes to be correct. While poker is being played, nobody announces his or her

selection. Only when all team members have decided are the cards revealed (simultaneously).

Almost always, the estimates of individual team members differ. This is good, as it gives each person the opportunity to learn something. The two team members with the highest and lowest estimates now explain how they got to their respective numbers. This explanation is only for the exchange of information, not about who is right. At this stage, the moderator of the meeting should ensure that no conflicts arise. Perhaps things can be clarified once again through the exchange of information or via additional information supplied by the product owner. The moderator may, if deeming it necessary, retain this information in the form of notes.

Once both team members have explained how they arrived at their values, the estimation is repeated. Once again all the parties involved select a number and show it simultaneously. Generally, the figures have now aligned themselves: for example, the values 8, 8, 5, and 8. The moderator then asks whether we can agree on a value of 8. If the team members are in disagreement, a third round is played. This time, the values should be almost identical. If not, the "most sensible" value is considered. This estimation procedure is not about accuracy but, rather, about selecting a value that makes sense.

In this way we are able very quickly to obtain a valued product backlog in which all team members were involved. This factor is crucial because only when all team members have together completed the estimate of the backlog items can they get involved in these estimates. More important is the fact that during the poker, all team members manage to gain an idea of what is to be developed. For larger teams and projects, Boris Gloger has developed another method of estimation: magic estimation, which is described in Chapter 3.

1.3 AGILITY FROM THE PERSPECTIVE OF PROCUREMENT

It is already quite common in modern procurement of custom software to insist that after no later than four weeks, the first fully functional increment of the product be shipped. (Custom software is designed specifically for the client. As part of the custom software or product concept we also understand software development that happens during customer-specific software integration projects.) The "elevator pitch" of Scrum inventor Ken Schwaber has always been: "I help companies deliver software in 30 days" (Schwaber and Sutherland, 2012). That's what the agile software development is: fast, iterative, one increment at a time. The buyer is able to learn from mistakes on both sides and does not have to make a big decision without the possibility of handling the risks properly in the course of the project.

The contractor should deliver product features consecutively. Feedback from the customer should be incorporated as soon as possible, and despite possible changes to the scope of the project, the overall result desired should be delivered. It is, however, very unlikely that for large orders a contractor

will be able to deliver an entire product within 30 days, but there is a high probability that after 30 days a first increment will be delivered with which the customer will be able to start working.

The traditional development processes of Winston Royce are not able to meet the foregoing expectations placed on software development services. The traditional model, developed in the 1970s, is still in frequent use. It is, somewhat, a contradiction to Scrum, as the capturing of requirements, the creation of the entire design, and all the contractual duties and a subsequent tender often take much longer than four weeks. Thus, first units are typically delivered much later than with the agile model.

The problem inherent in all software development projects and all services is the variability as to what should be delivered: *the inability to know what you actually need*. We know that this is difficult to digest for those in procurement (and perhaps for many others, too). However, the understanding that this inability is a systematic, even a necessary principle is essential. The simple fact that you agree to purchase and develop a project for which you cannot describe all the details in advance is actually the first important step toward project success (Reinertsen, 2009).

An example: A mechanical engineer wanted to implement a new method of materials testing. Part of the work of the project team consisted of operating the essential innovation and inventing the vital approaches and components. In this example it is obvious that you cannot know whether the team will manage to deliver the desired outcome on a given time line and at a specified cost. The value of the product lies in the new product ideas, or the invention. If you know what you want to invent but still do not know how to get there in detail, the value of the new product is to eliminate the lack of knowledge as to how to get there.

And here lies the paradox. To plan would mean to know what to do and to know how the result can be achieved. This is not so because you want to explore new territory. The waterfall project cannot help here and it is understandable that it would be a waste of money to invest in detailed specifications as to what should be done. A project based on traditional practices simply cannot consider all the uncertainties associated with the problems that exist.

Despite the weaknesses of traditional tendering processes and classical implementation approaches, it is interesting to hear the following from some procurement representatives and even key account managers. These statements are often linked to the rejection of an agile contracting model and show the tendency toward fixed-price contracts:

- "I want to know what I get for my money!"
- "We need to know exactly what the client wants; otherwise, we cannot estimate our costs."
- "The customer always wants more than he or she asked for initially!"

- "How can I be sure that the contractor will not inflate the price? They can simply work more slowly."
- "I need a work contract, not time and materials development, because I can capitalize the investment accordingly."

We hear such statements in almost any discussion of agile software development and agile project management approaches. They suggest a deep-seated conflict among the parties—development, seller, and buyer—as to trust and, of course, the fact that on the one hand you want to buy something as inexpensively as possible and, on the other hand, sell something at the highest possible price with manageable risk in the development process.

Wait a moment. Can we not do anything about this? Is it not true that the nature of the business follows this exact principle? We buy things as cheaply as possible so that we can make as much profit as possible. We cover this point in subsequent chapters. It is, however, important at this point to understand that it is clear that this principle concerns business, and that we need new approaches for complex IT projects and custom software. Traditional contracts with fixed-price agreements or, at the other extreme, contracts based on time and materials often lead to lose–lose situations with just the illusion of solving the concerns listed above. In Chapter 5 in particular we show that modern procurement is capable of conducting tenders for agile fixed-price contracts jointly with the business departments, thereby awarding a contract to the supplier who reveals the best quality and price with the lowest associated risk.

1.4 AGILITY FROM THE PERSPECTIVE OF THE SOFTWARE PROVIDER

A seller of software services should be able to make customers pay adequately for the value and quality of the product and service provided. The sellers' (also called key account managers throughout the book) job is to offer a product that the customer wants at a higher price than a product that the customer does not want. The actual underlying problem is described brilliantly in an article about the agency David and Goliath:

> "We cannot complain about too little work, quite the opposite," says Matthias Czech, owner and creative director [of the agency David and Goliath]. There is however a catch. More and more frequently, says Czech, the agency is invited to pitches where the customer evaluates which agency best suits the company, based on the offer. This as opposed to evaluating based on creativity, efficiency and performance (http://bit.ly/rNiNAC, last accessed on 5/1/12).

Customers want the cheapest provider. They want to buy software integration services and software development services as if such services were a standard consumer product. They want to know at the beginning what it will

cost, but as described above, they do not have a detailed idea of what they really want. Sellers or service providers cannot offer a product when they do not yet have one. They have no idea at what price they can offer a product, because they do not know the market for a new product. This can only be established in the course of a project. At the same time, a seller knows that there will always be someone who claims to be able to offer the same thing more cheaply.

The variability and illusions surrounding expenses play a trick on sellers. If there is a really creative, highly effective team in the background which provides excellent quality quickly, a seller can potentially make a favorable offer. Even if he or she knows that the team requires little time (low cost), a seller cannot be sure that competitors will not offer lower price and provide lower quality. Indeed, the customer and even the seller cannot define exactly what is to be delivered, but both believe in the illusion that this can be described perfectly.

And then there is the fact that the seller should in no case offer a project at a discounted cost. He or she puts a team to work that is very good, works fast, and offers high quality. As a result, the seller should offer the project at a fairly high price because, after all, the customer is being provided with a product of high value. In both cases, whether a fixed-price project or a time and materials project, the situation is not advantageous for the seller. This is different, of course, if the seller follows a philosophy of offering at lowest cost and then generating additional revenue based on tons of change requests.

1.5 THE 12 PRINCIPLES OF AGILE SOFTWARE DEVELOPMENT

In addition to the four pairs of values, the authors of the agile manifesto have named 12 additional principles that apply to the agile management framework. Next we show how the parties involved—customer, supplier, and development or Scrum team—can shape these principles and achieve project success through shared support.

The practices listed below are just a few of the possibilities for collaboration. They symbolize a different way of dealing with customers, suppliers, and teams. Implementing Scrum or other agile management framework always influences the entire organization. Often, not all parts of an organization are able to implement all aspects at once. Therefore, consider the statements below only as references of where relationships should evolve.

1. Emphasis is on delivery

"Our top priority is to satisfy our customers through early and continuous delivery of valuable software content."

How do we behave as a customer?

- We participate in sprint reviews.
- Our department takes on finished software at the end of a sprint if possible, otherwise as early in the process as feasible.

- We integrate the software into our existing systems as soon as possible after delivery, as this often deters the risks inherent in the fact that the software operations department has a very specific view of the software delivered.
- We give critical but respectful feedback.

How do we behave as a service provider?

- Our software development processes allow us to show the customer fully functional software after each sprint.
- We make appropriate environments accessible to the customer.

How do we behave on the Scrum team?

- We optimize our development practices in the team so that we are able to present completed software at the end of a sprint.
- We talk extensively with users and customize applications according to the user's requirements.
- We deliver the most effective solution that fulfills the user's requirements.

2. Free exchange

"Accept changes in requirements even late in development. Agile processes use changes to the competitive advantage of customers."

How do we behave as a customer?

- We distinguish between functional requirements and the conditions of the project.
- We are personally involved in the project vision and understand the technological implications.
- We are aware that we are permitted to make changes.
- We understand that there is a profound change when we alter the environment.

How do we behave as a service provider?

- We welcome changes.
- We develop in such a way that we are able to respond rapidly to changes.
- We make this possible with good documentation, refactoring, and continuous and open communication about what has actually occurred.
- We invite customers to daily Scrums, sprint plannings, and reviews.

How do we behave on the Scrum team?

- We communicate with customers to fulfill their needs. We try to think like customers and users.

- We are open to criticism when our applications are under review.
- We acknowledge our errors and correct them immediately.

3. Deliver in iterations

"Deliver functional software regularly within a few weeks or months and favor these shorter time periods."

How do we behave as a customer?

- We are present at sprint reviews and give feedback.
- As quickly as possible, we integrate into our existing infrastructure the partial functionality delivered.

How do we behave as a service provider?

- We invite customers to sprint reviews and openly discuss the current status.

How do we behave on the Scrum team?

- We deliver complete functionality at the end of each sprint.

4. End user and developer sit together

"Experts and developers must work together daily throughout the project."

How do we behave as a customer?

- We make experts from various departments available to the development team.
- We are available when the development team has questions.
- We take time for the project and respect the fact that nobody can guess our wishes without us giving feedback and helping to understand the details.

How do we behave as a service provider?

- We welcome the experts when they are present in the Scrum team.
- We call them when we have questions.
- We make a point of inviting them to sprint planning.

How do we behave on the Scrum team?

- We work with the experts daily.
- We make an effort to understand the experts.
- We observe how the expecrts work. However, we do not ask them what they want; instead, we work with them until we know what they need.

5. Trust the individual

"Build projects around motivated individuals. Give them the environment and support they needs and trust that they will get the job done."

How do we behave as a customer?

- We search for an agile development partner.
- We provide motivated experts.
- We trust the delivery team for at least the first three sprints.
- We also check whether they deliver what we expect.

How do we behave as a service provider?

- We select project team members who really want to work on the project.
- We give them the tools they need for their work and remove unnecessary bureaucratic hurdles.

How do we behave on the Scrum team?

- We state openly when we need something or when we are obstructed by something.
- We have a Scrum master who clears obstacles out of our way.
- We behave respectfully toward customers and management.

6. Face-to-face communication is more effective

"The most efficient and effective method of delivering information to a development team is face to face."

How do we behave as a customer?

- We understand that good documents are always only a result of successful face-to-face communication.

How do we behave as a service provider?

- We communicate openly with the client. All information, including problems and areas where we are lagging behind, is visible to the customer.
- We do not hide anything.

How do we behave on the Scrum team?

- We talk to users.
- We understand their needs.
- We observe users while they are working.

7. All that matters is completed functionality

"Functioning software is the primary measure of success."

How do we behave as a customer?

- We call on our service provider to deliver the first part of completed software within 30 days.
- We are not satisfied with documents as representing of progress.

How do we behave as a service provider?

- We deliver software in short intervals.
- All obstacles on the customer's side are discussed openly, and all obstacles on our side are also addressed and resolved openly.

How do we behave on the Scrum team?

- We constantly deliver software that is potentially usable.

8. Sustainable pace

"Agile processes promote sustainable development. Clients, developers, and users should be able to maintain a steady pace indefinitely."

How do we behave as a customer?

- We will not push the team for functionality and deadlines, and we will not demand overtime or changes at the last minute.

How do we behave as a service provider?

- We constantly work professionally and to a high standard.
- We deliver only tested, documented, and representative software.
- We do not commit to functionality over long periods.

How do we behave on the Scrum team?

- We honor our commitments in the sprint.
- We work as a team, striving continually to deliver consistently on a high level.

9. Quality is an attitude

"Continuous attention to technical excellence and good design promotes agility."

How do we behave as a customer?

- We expect high technical quality from our contractors, and we know that this is not something that we can buy for a bargain price.
- We do not, therefore, select our service providers according to price, alone.

How do we behave as a service provider?

- We deliver to the customer an excellent design with extensible architecture, and we invest in the training of our staff.

How do we behave on the Scrum team?

- We look constantly to the future and search for solutions that are extensible.
- We perform test-driven development, we automate, and we document.
- We educate ourselves continuously to correct weaknesses.

10. Keep it simple, stupid (KISS)

"Simplicity. The art of maximizing the amount of work not done is essential."

How do we behave as a customer?

- We check constantly whether we still require what we requested.
- We cancel projects if we already have what we need.
- We accept contracts that allow us these things.

How do we behave as a service provider?

- We deliver from the start the features that are most valuable to the customer.
- We favor short project lead times.
- We deliver fast, we deliver functionality, and therefore we do not bill in hours.

How do we behave on the Scrum team?

- We are always looking for the simplest solution that can be produced professionally.

11. Complexity can only be answered with self-organization

"The best architectures, requirements, and designs emerge from self-organized teams."

How do we behave as a customer?

- We make how our systems work completely transparent.
- We do not yet define solutions.
- We allow the teams to do their work.

How do we behave as a service provider?

- We educate the staff in such a way that all the necessary skills are present in the team.

How do we behave on the Scrum team?

- We reveal when there is something that we cannot achieve due to a lack of skills.
- We actively ask questions.
- We work proactively with customers.

12. Learn from postmortems

"At regular intervals, the team reflects how it can be more effective and adjusts its behavior accordingly."

How do we behave as a customer?

- We expect to hear of errors made by the teams that have led to improvements.
- We participate by invitation in the project retrospective.
- We respond to requests for change and respect them as having the potential to increase productivity.

How do we behave as a service provider?

- We work with customers continually to improve our relationships and inform them as to where we see potential for improvement.

How do we behave on the Scrum team?

- We perform our retrospectives rigorously.

1.6 SUMMARY

Companies that want to meet the requirements of dynamic markets are relying more and more on agile methods of software development. The management framework Scrum is the method most commonly used. While development teams using agile methods are already presenting impressive results, the benefits of agile development are still not obvious to many buyers. Therefore, agile-developed products and projects are often grouped into inappropriate "traditional" contract constructs.

The main drawback: Valuable principles of cooperation between customers and suppliers, as they are to be implemented in the agile way of thinking, based on the agile manifesto, continue to be disregarded in these rigid contract constructs. Customers as well as suppliers do not have a successful project in sight as a goal but, rather, each entity to its own advantage. Both sides struggle with the same problem: the inability to know what is actually needed and how the details of the scope will change during the project for various reasons. The basic problem for all software development projects and all service providers is therefore the variability of what is to be delivered.