SCRUM MASTER

Scrum Manager®

Scrum Master

Core curriculum 1

version 3.07

Scrum Manager ®

Version: February 2022.

Author: Marta Palacio.

Illustrations and cover: María de la Fuente Soro.

Table of Contents

守破離
Shu Ha Ri

The learning process of any skill has three stages:

Shu: the student picks a technique, assuming that it is correct, and mimics it.

Ha: more techniques are collected and practiced.

Ri: the student experiments and invents new techniques, combining and adjusting them through self-discovery.

The techniques of the shu stage are usually safe to apply in most situations. Ri-stage techniques, however, only work under certain circumstances, and they demand a higher level of expertise to know when and how to use them.

"You cannot win in a competitive industry using shu techniques."

Alistair Cockburn

Preface

This guide contains the materials for Scrum Manager®'s official Scrum Master certification. It explores how to implement and improve a scrum framework when managing agile projects, teams, and organizations.

The audience of this book includes all people interested in the agile management model called 'scrum,' either to apply it in their daily work or their team or to learn how to manage different projects the agile way.

Although this model emerged within companies in the technology sector, today it can be found in all kinds of innovative environments. The common factor among them is the production of knowledge, instability, and rapid and constant change. Many companies have discovered that in these industries, agile management is the best adapted.

This manual is suitable if you work in any of the so-called knowledge companies, which often operate in ever-changing environments. Scrum is also a handy tool to understand if you work in the management of cultures and people. Above all, it is appropriate if the value of your product depends on the talent of motivated people, rather than on the processes and tools they use.

***

The content is divided into three parts:

The introduction, which contextualizes the birth of scrum and defines what agility is in the business context. We highly recommend to read it if this is your first manual on agile management. It also explains the difference Scrum Manager® makes between agile 'practices' and agile 'principles and values,' which is key to what follows.

The first part of the manual focuses on the most widespread scrum practices. It explains the 'roles,' 'artifacts,' and 'events' that have become standard over time and that environments with this management model use.

After familiarizing ourselves with these concepts, the second part delves into the principles and values from which they arise. It presents what Scrum Manager® refers to as scrum principles and values for a freer but more conscious use of the framework. Finally, we list a few additional practices that are not part of the standard framework, but they are frequently used and complete the inventory of agile

management tools.

***

When talking about scrum and agility, some terms acquire a specific meaning that is important to keep in mind. Whenever a word of this type is introduced, it will be enclosed in single quotation marks ‘ ’ to make it easier for the reader to find information and to point out that it is a word with a particular connotation.

Introduction

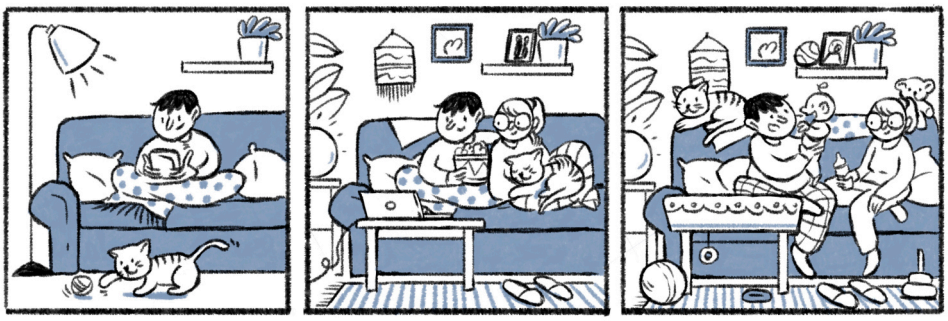
Agility

Agile management emerged as the antithesis of predictive project management, a model that we will refer to frequently in this manual. Both models have their virtues and are more useful in specific industries. Predictive management focuses on planning, calculating a budget, and setting deadlines. If the final product is delivered in time, without exceeding costs, and it includes all the functionalities of the initial plan, it is considered a success.

As reasonable as it sounds, this has many drawbacks when we try to apply it in constantly and rapidly changing industries. That definition of 'successful' serves in a stable environment, where products are the result of scrupulous attention to processes and protocols.

Predictive management is a result of the Industrial Revolution: it comes from the world of construction, automobiles, and factories. If the client is looking for a house, for example, it should be built in such a way that it is durable, safe, and meets the needs of its inhabitants. And, in an ideal scenario, within the planned timeframe and without exceeding the cost.

But we can find plenty of products today that share nothing in common with the Industrial Revolution ones. Firstly, because they can be abstract, like a movie or a mobile app. You can try out new things during development, empirically testing what works and what doesn't. Adjustments can be made at any given moment. And you can start with a first sketch of the basics you need and work your way up. The scenario may change: a functionality that seemed essential at first may be outdated by the delivery date. Or a competitor may launch an exciting new feature that leads to a review of the product's priorities. Being competitive requires the ability to respond quickly in uncertain work scenarios. This means there are no stable requirements when designing new products or services. They need to be available for customers as soon as possible, and then continuously maintained and improved. In these products, innovation is a crucial value.



Agility starts from a viable minimum and develops the project by adapting to the circumstances as they change.

These and more reasons we will see led to question the predictive management model, which didn't seem to fit the reality of what knowledge companies needed. Understanding as such those organizations that develop products or services based on knowledge rather than tools and processes.

The working environment of these companies is very different from the one that originated predictive project management. Now, there are markets with such a rapid evolution that it is pointless to try to start projects with a closed plan. There is a need for strategies that deliver tangible results soon, and that allow responding in time to changes. The product is built at the same time as changes and new requirements are introduced. The client starts from a more or less clear vision, but the level of innovation required, as well as the speed at which the business environment moves, does not allow him to foresee in detail how the final result will be.

Today, there are product managers who do not need to know the 200 functionalities of the final product, or if it will be finished in 12 or 16 months. Some customers need to have the first version with minimum functionalities in a matter of weeks, instead of a complete product within one or two years. Their interest is to quickly put a new concept on the market and increase its value over time.

Origins and why agility is often linked to IT

Knowledge evolves following a dialectical pattern of thesis, antithesis, and synthesis. Each thesis has an antithesis, which brings out its problems and contradictions. The antithesis is also inadequate in some way, and from the confrontation of the two, a third moment called synthesis emerges: a resolution and a new understanding of the problem.

> "It is at this stage that the previous thesis and antithesis are reconciled and transcended. However, over time, even synthesis will turn out to be one-sided in some other respect. It will then serve as the thesis for a new dialectical movement, and so the process continues in a zigzag and spiraling manner." (Nonaka 2004)

Agile practice frameworks did not emerge as a knowledge thesis, but as an antithesis to the one that software engineering had been developing.

Processes and predictive management

In 1968, during the software crisis, NATO held the first conference focused on analyzing programming problems. The need to create a scientific discipline that would allow a systematic and quantifiable approach to the development, operation, and maintenance of computer systems became apparent. It resulted in an attempt to apply process engineering to software, thus "software engineering" (Bau 1969). This first strategy (thesis) was based on two pillars:

- **Process engineering.** A successfully tested principle for quality that comes from industrial production environments says that the quality of the result depends on the quality of the processes. In other words: you don't need brilliant or highly qualified people, as long as they follow quality guidelines.
- **Predictive management.** This umbrella term englobes those management styles that focus on ensuring that agendas and budgets are met, in opposition to reactive management.

As the discipline evolved and was perfected through different process models and bodies of knowledge for project management (MIL-Q9858, ISO9000, ISO9000-3, ISO 12207, SPICE, SW-CMM…) people in the software industry raised their concerns and this strategy was questioned.

From the mid-90s to 2010, radical positions among advocates of process models (thesis) and agile frameworks (antithesis) have been common:

> "Q: What's the difference between a bank robber and a (CMM) methodologist? A: You can negotiate with a bank robber." (Orr 2002)

> "CMM certification is largely about an organization's management processes (estimating, scheduling, control) and not nearly so much about the quality of the software products produced." (Orr 2002)

> "If one were to ask a typical software engineer if CMM and process improvement were applicable to agile methods, the response would most likely range from a blank stare to hysterical laughter." (Turner & Jain 2002)

Critics believed that predictive planning was not appropriate for any project. In practice, complying with pre-established dates, costs, and functionalities is not always a valid measurement of success. On the other hand, it was also questioned if it was reasonable to follow industrial process patterns in software development and other knowledge-based work. In these cases, it became increasingly accepted that the tacit knowledge of the person doing the work contributes more to the value of the result (→ A breakdown of project management > Knowledge).

Manifesto for Agile Software Development

> "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> - Individuals and interactions over processes and tools.
> - Working software over comprehensive documentation.
> - Customer collaboration over contract negotiation.
> - Responding to change over following a plan.
>
> That is, while there is value in the items on the right, we value the items on the left more."

In March 2001, 17 software professionals were convened by Kent Beck, who had published a couple of years earlier the book on the new Extreme Programming methodology (Beck 1999). They all had one thing in common: they were critical of process-based production models.

They met in Salt Lake City to discuss the processes employed by the programming teams.

The meeting coined the term 'agile methods' to define those that were emerging as an alternative to formal methodologies, such as CMM-SW (precursor to CMMI), PMI, SPICE (initial draft of ISO 15504, wich in turn was the precursor to ISO 33000)… They considered these excessively heavy and rigid due to their normative nature and strong dependence on detailed pre-development planning.

The attendees summarized their ideas in four postulates, the Agile Manifesto, which are the values behind these methods. They are the ones that open and are developed in this section. They also established 12 principles, which we'll mention at the end.

Individuals and interactions over processes and tools

The most important postulate. There is no doubt that processes help: they serve as an operation guide, and having the right tools improves efficiency.

In process-based production, the aim is for the quality of the result to be a consequence of the processes. In agile development, processes only serve as help.

The defense of processes at all costs leads to the assertion that extraordinary results can be achieved with mediocre people. But the truth is that this is not the case when creativity and innovation are needed. These tasks require talent and motivated people to provide it.

Working software over comprehensive documentation

The Agile Manifesto does not consider documentation to be useless: only unnecessary documentation. Documents allow recording and communicating relevant information for the project. Furthermore, for legal or regulatory reasons, they can be obligatory. But their relevance must be less than that of the product.

To be able to anticipate how the final product will work by observing prototypes and finished parts provides stimulating end enriching feedback. It serves to reach ideas that were inconceivable at first. More often than not, preparing a very detailed document of requirements before you start is a waste of time. It can be argued that detailed documentation facilitates sharing information about the project among the people involved. But this is rarely the case. It lacks the richness and value that can be achieved through direct face-to-face communication and interaction with product prototypes. In fact, not only does it lack these advantages, but it also creates bureaucratic barriers between departments and individuals.

Therefore, whenever possible, the use of documentation should be reduced to the minimum necessary. The ideal is to eliminate all those that consume work without adding direct value to the product.

Customer collaboration over contract negotiation

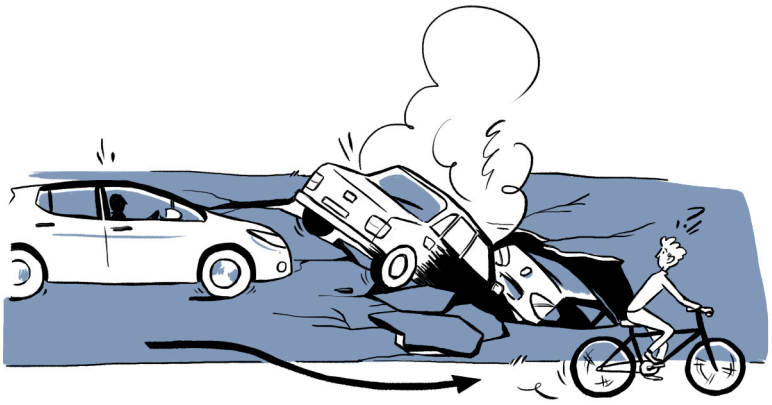
The goal of an agile project is not to control the execution to ensure that the initial plans are met, but to provide the highest possible value to the product continuously.

As mentioned earlier, when developing continuously-evolving products (such as a web app), it is not possible to define in a closed requirements document what the final product should be. It is more efficient to take direct feedback while developing the product, and consequently redefine and improve the requirements of the remaining parts.

Responding to change over following a plan

The central values of agile management are anticipation and adaptation, different from those of orthodox project management (planning and control to ensure compliance with the plan).

Responsiveness is much more valuable than monitoring and assuring plans when developing products with unstable requirements. If the original idea does not work anymore, it has to be possible to change it.



The 12 principles behind the Agile Manifesto

In addition to the four postulates we have just seen, the Agile Manifesto establishes these 12 principles:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity–the art of maximizing the amount of work not done- is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Origins of scrum

Scrum is an agile development model characterized by:

- Autonomous and self-managed teams that share their knowledge openly and learn together.
- An 'incremental' development strategy rather than complete product planning.
- Basing the quality of the result on the tacit knowledge of people and their creativity. Not on the quality of the processes.
- Overlapping the different phases of development, instead of carrying them out one after the other in a sequential or 'waterfall' cycle.

The origin of the term is far removed from that of project management: it comes from rugby. 'Scrum' defines the formation in which both teams, crouching and clinging to each other, push for the ball without touching it with their hands.

But for our purposes, we have to go back to 1980s Japan when researchers Ikujiro Nonaka y Hirotaka Takeuchi gave the term a polysemic dimension.

They identified a novel form of development in the industrial manufacturing companies that were obtaining the best results in innovation and time to market: Fuji Xerox, Canon, Honda, Nec, Epson, Brother, 3M y Hewlett-Packard (Nonaka 1986). They compared their way of working in self-managed teams with the way rugby players advance when in scrum formation, hence the term.

Although this way of working emerged from technology product companies in industrial manufacturing, it started to be also applied to the software industry from 1995 onwards. That year, Ken Schwaber presented in OOPSLA (the Object-Oriented Programming, Systems, Languages & Applications annual conference) a software development methodology based on a scrum environment, using that same term (Schwaber 1995). This first framework presented a series of phases and 'artifacts': pregame, game, postgame, planning, sprints, wrap… Some of them are still in use, and we will see them. But in general, the rules of the game have changed a lot since then.

There is no single authority that determines what ‘scrum’ is and is not. It has changed over time, and it will continue to evolve with the input of the professional community, which defines the most useful practices. The original spirit, however, remains: practices should help teams to self-manage and maintain a continuous flow of progress, producing results iteratively and frequently.

Among the ‘events’ and practices that have been added over the years, we can find, for example, retrospective meetings, refinement product backlog meetings, DoR (Definition of Ready), story maps...

Scrum Manager® uses the term ‘scrum’ with its original meaning, the one given by Nonaka and Takeuchi.

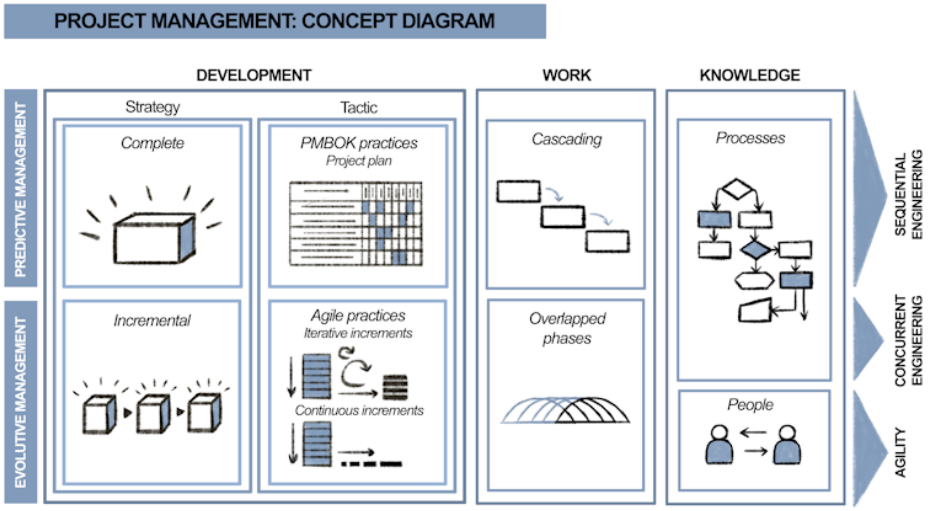
A breakdown of project management

Since 1980, companies have developed so many models and practices to improve the quality and efficiency of their projects that it can be overwhelming to study them. In this section, we will go beyond labels and summarize the principles behind these frameworks, to better understand different management strategies.

We will use three concepts and two management models as coordinates:

- The first three are development, work, and knowledge.
- The models, which have already been mentioned, are predictive management and evolutive management.

We can simplify the apparent labyrinth of frameworks using these five ideas. In the end, every management model can be understood in this diagram:



1. Development

The development of the project can be complete or incremental.

- In a complete development, the description of what the client desires is available at the beginning of the project. The plan is complete and detailed, and it serves as a basis for estimations. It serves to organize tasks, resources, and work schedules. During the execution, the team manages the fulfillment of what has been planned.
- In the case of incremental developments, the complete description of what the client wishes to obtain is not available at the beginning of the project. It increases and evolves during development. This type of development can be managed using two different tactics:
  - Continuous incremental development: using techniques to achieve a continuous flow of development of the product's functionalities or parts. They are delivered at irregular but continuous intervals to the customer.
  - Iterative development: using techniques of prefixed time or timeboxing to maintain the production of product increments at a fixed rate. This is how the standard scrum framework works, which sets the 'sprint' (→ Events) as the measure for each development iteration. At the end of each 'sprint,' a new 'increment' of the product is obtained: that is, a deliverable, ready-to-use part.

2. Work

The way of working can be sequential ('waterfall') or concurrent.

- Sequential work is divided into consecutive phases. A new phase starts when the previous one is finished. The most common example is the waterfall cycle defined in software engineering, which consists of definition of requirements, analysis, design, coding, testing, and implementation.
- Concurrent work overlaps the different phases in time. Following the same example from software engineering, this would mean that all the phases mentioned in the previous paragraph would be reviewed simultaneously and continuously.

3. Knowledge

The different models can place knowledge either in processes or in people.

- In a process-based production: knowledge is explicit. The quality of the result is found, to a greater extent, in the process and technology used.
- In people-based production: knowledge is tacit. The quality of the result depends on the experience of the members of the organization. It isn't about following processes and protocols correctly, but about making sure that people are motivated and talented.

An example of explicit and tacit knowledge could be the difference between a food processor and a cook making dinner. Anyone can prepare a meal using the food processor by following the instructions. The result will always be the same regardless of the skill. In the second case, however, the talent of the person is paramount. An amateur cook will not prepare the same thing as an haute cuisine chef. Both will benefit from having proper tools, such as non-stick pans and sharp knives, but these tools are just a help.

> "Tacit knowledge is personal, context-specific, and therefore difficult to formalize and communicate. Explicit or 'codified' knowledge, on the other hand, is knowledge that can be transmitted with formal and systematic language." (Nonaka 1995)

Predictive management: sequential engineering

Sequential engineering, also known as traditional engineering, is intended to provide predictable results. A successful project, according to these models, will develop the expected product without exceeding the agreed time and resources. The type of development is, therefore, 'complete,' and it employs traditional planning practices.

In the world of software, the main references developing knowledge for this type of management are PMI e IPMA y the process models CMMI, ISO 33000, SPICE… All of them use sequential engineering and process-based production.

Evolutionary management: concurrent engineering and agility

Evolutionary management aims to deliver a viable product as soon as possible and to increase its value continuously. It employs a strategy of overlapping work phases and incremental development. This can be achieved by maintaining a rhythm of short, cyclical iteration, or a continuous flow of development.

Knowledge marks the difference here. This strategy can be carried out with process-based production (concurrent engineering) or people-based production (agility). This distinction is important to avoid confusion, such as considering that 'agility' is equal to the simple application of standard scrum 'rules' (iterative increment cycles with defined 'roles' and 'artifacts'), or visual kanban techniques for a continuous flow of tasks.

Concurrent engineering	Agility
It uses strategies that are typical of agile management: overlapping development phases, multidisciplinary teams, and frequent improvement iterations.	It reduces or eliminates administrative and bureaucratic tasks that do not add value to the product or the development system.
	It's typical of knowledge enterprises.
It focuses on the quality of processes.	It focuses on the tacit knowledge of people, culture, and talent.

Scrum

Summing up, we come back to the characteristics of scrum (→ Origins of scrum) and see how they fit into the characteristics of agile management:

- It uses an incremental development strategy (which can be iterative, through timeboxing, or continuous)
- It overlaps the different phases of development.
- It bases the quality of the result on the tacit knowledge of the people and their creativity.
- In addition, scrum is also characterized by working in autonomous and self-managed teams, which share their knowledge and learn together. Hence the name and the metaphor of moving forward as a scrum.

Differentiating scrum practices from principles and values

When you start working with scrum, as with any other tool, it is advisable to read the manual and follow the instructions; that is, to adopt the standard framework. We are going to explain it in the first part of this manual: its 'roles,' 'events,' and 'artifacts.'

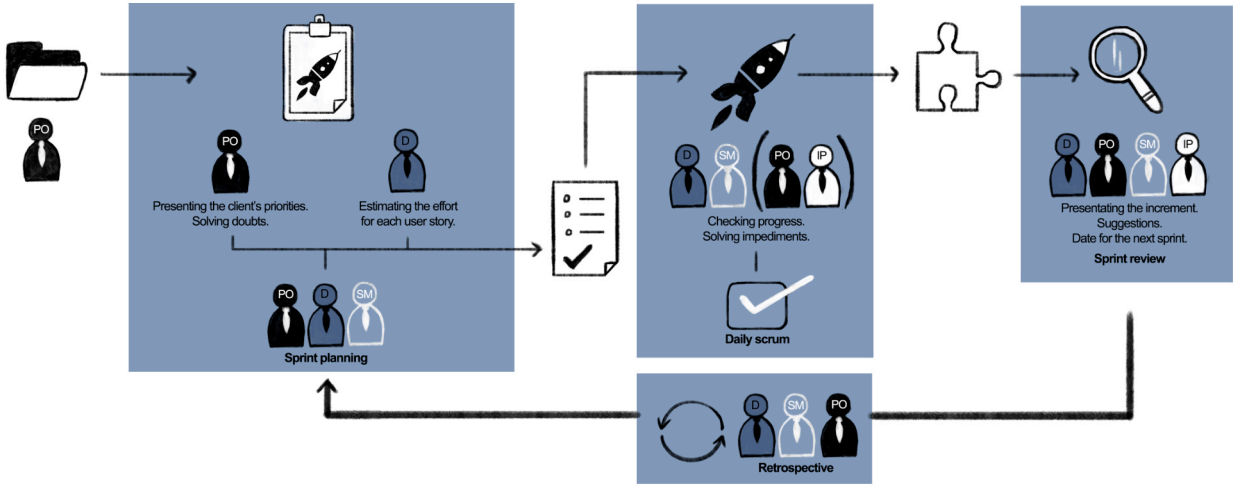
But it's pointless to try to deceive ourselves: if our focus is on the process instead of the tacit knowledge of people, we won't be doing 'agility,' but 'concurrent engineering.' When an iterative flow of progress is achieved, one can try to go beyond it. It is time then to unlearn the practices and rely on the principles and values of scrum, adapting it, and other techniques and frameworks to the specific characteristics of the project or team. In most agile companies, these companies can be adapted and, indeed, they are.

The first part of the book explains the most widespread scrum techniques, which can be found here, on the Internet and in other manuals. In the second part, we will explain how to remove those 'stabilizer wheels' from the bike, which come in handy at first but can hinder us in the long run, so we can keep moving forward.

PART 1: THE SCRUM CYCLE

LEARNING THE STANDARD PRACTICES

SCRUM FRAMEWORK



ROLES

PRODUCT OWNER
Sets priorities.
Individual role.

DEVELOPER
Builds the product.

SCRUM MASTER
Manages and facilitates the scrum framework.

INTERESTED PARTIES
Other people involved.
They observe and give advice.

ARTIFACTS

PRODUCT BACKLOG
Prioritized product requirements, not overly detailed.
The list evolves and all roles have access to it.
The product owner is ultimately responsible for it.

SPRINT BACKLOG
Product requirements for the sprint, with a sufficient level of detail to be carried out by the team.

INCREMENT
Part of the product completed after a sprint, ready to be used.

EVENTS

SPRINT PLANNING
Max. duration: 1 work day. The product owner presents the client's priorities. The team estimates the effort required for those requirements and prepare the sprint backlog. The team defines, in one sentence, the sprint's goal.

SPRINT
Basic development cycle.
Recommended duration: less than a month.
Max. duration: 6 weeks.

DAILY SCRUM
Max. duration: 15 minutes. Each member of the team shares what they did the previous day, what they will do next, and if they are having or foresee any problems. The sprint backlog is updated.

SPRINT REVIEW
Max. duration: 4 hours. Presentation of the increment. Time for suggestions and schedule the beginning of the next sprint.

RETROSPECTIVE
The team members self-analyze their way of work to reinforce positive aspects and plan how to improve negative ones.

The scrum cycle

As to the date of publication of this manual, the components of the standard scrum cycle are:

- Scrum team, composed of the following roles:
  - Developers
  - Product owner.
  - Scrum master.
- Artifacts:
  - Product backlog.
  - Sprint backlog.
  - Increment.
- Events:
  - Sprint.
  - Sprint planning meeting.
  - Daily scrum.
  - Sprint review.
  - Sprint retrospective.

We start with a broad, general vision of the desired result, and from there, we specify and detail the functionalities we want to obtain first.

Each development cycle or iteration (‘sprint’) ends with the delivery of an operational part of the product (‘increment’). A sprint can last from one to six weeks, although ideally no longer than than one month.

In scrum, the team monitors the progress of each sprint in daily short meetings where they review the work done the day before and the work planned for the current one. These meetings have a 5-15 minute duration and take place near a board which features information on the tasks of the sprint. They are often referred to as ‘stand-up meetings,’ ‘daily scrum,’ or ‘morning roll call.’Scrum manages the evolution of the project empirically, with the following tactics:

Review of iterations

At the end of each sprint, all those involved in the project review the functionalities of the result. Therefore, the duration of the sprint is the maximum amount of time to discover approaches that might be wrong, improvable, or misinterpreted.

Incremental development

You don't work over designs or abstractions. Incremental development offers a working product part at the end of each iteration, which can be used for testing, inspections, and evaluations.

Phase overlap

During development, the team refines both design and architecture, instead of setting them fixedly at the beginning of the project. Waterfall development overlaps the different phases, carrying them out one after the other. Scrum overlaps them, so they advance simultaneously.

Self-management

Predictive management assigns the responsibilities over the project's management and results to the project manager. In scrum, teams are self-managed. They have sufficient decision power to adopt the measures they consider necessary and appropriate. It speeds up the decision-making process and allows a quick response to unforeseen events.

Collaboration

All team members collaborate openly with others, according to their abilities and not to their role or position.

Through self-management and collaboration, the work that would otherwise be done by a project manager can be done efficiently by the team members.

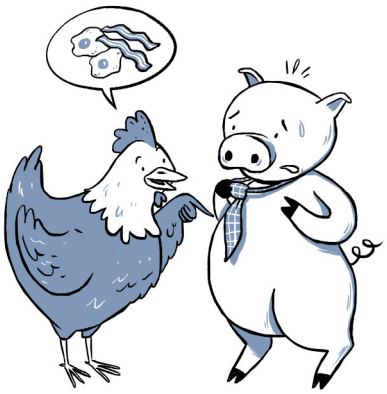
Committed and involved

During the project's development, many people get involved and add value to it. They do so at different levels of commitment and responsibility, which leads to making a distinction between 'committed' and 'involved' people.

Committed: they contribute directly, building the product or developing the service.

Involved: they play functions that don't involve directly building the product. They may be managers, or work in marketing or customer support.

***



A hen and a pig were walking down the street. The hen asked the pig:

"Would you like to open a restaurant with me?"

The pig considered the proposal and responded:

"Sure! What would we call it?"

"Ham and Eggs!"

The pig reconsidered:

"Forget about it… I'd be really committed, while you'd be just involved."

***

In scrum circles, it's common to call those who are committed (without any pejorative connotation) 'pigs', and those involved 'hens'. These nicknames come from the story of the Pig and the Hen, that serves to illustrate the difference between them.

Roles

Product owner

This role is responsible for making the client's decisions, acting in their best interest, and increasing the value of the product.

To simplify communication and decision making, this role should fall to a single person. If the customer is a large company or has several departments, they can adopt the form of internal communication they consider appropriate. Still, only one person should join the project team. This person represents the client and must have the knowledge and powers necessary to carry out decisions.

In internal developments for the company itself, this role is usually assumed by the product manager or the persons responsible for marketing. In developments for external customers, the person responsible for the customer acquisition process. Depending on the circumstances of the project, it is even possible for the product owner to delegate on the team or someone they trust. Even in these cases, the product owner maintains these responsibilities:

- Developing and managing the product backlog. (→ Artifacts)
- Communicating the product vision and user stories, participating in each sprint planning meeting. (→ Events)

The product owner is in charge of the product backlog, which means that they are who ultimately decide what the final product will look like and the order in which the team will build the increments. It's up to them to choose what to add, cancel, and the priority of the 'user stories.' As the customer's representative, the product owner knows the product plan, its possibilities, investment plan, expected delivery dates, and expected return of investment.

The person with this role must have expert knowledge of the customer's business environment, understand their needs, and what the ultimate purpose of the project is. So they can share this vision with the team and prioritize requirements. They must carry a constant analysis of the business environment: market evolution, competition, alternatives… and combine this information with discoveries that might emerge from the team during the development process.

Developers

The group of professionals who develop the increment (→ Artifacts) of each sprint.

It is recommended to keep their number between three and nine. Beyond nine, it is difficult to maintain direct communication, and the usual frictions caused by group dynamics become more evident (they start appearing at six people).

They're multifunctional, in the sense that all developers work in solidarity and share responsibilities. Some of them may be specialists in particular areas, but the responsibility for the final product is shared by all of them as a whole. The main responsibilities, beyond self-management and use of agile techniques (in this case, scrum), make the difference between a 'workgroup' and a 'team':

- In a 'workgroup,' colleagues have specific assignments of tasks, responsibilities, and follow a process or execute guidelines. The operators in a factory chain are part of a workgroup. Although they share a common boss and work in the same organization, each one responds individually.
- A 'team' has a collaborative spirit and a common purpose: to achieve the highest possible value for the customer's vision. A scrum team responds as a whole. It works in a cohesive and self-managed way.
    - There is no manager to set, assign, and coordinate tasks. Team members themselves are in charge of this. Everyone in the team knows and understands the vision of the client. Everyone contributes and collaborates with the product owner in the development of the product backlog, participates in decision-making, and respects the opinions and contributions of others. They share the goal of each sprint and the responsibility for achieving it.

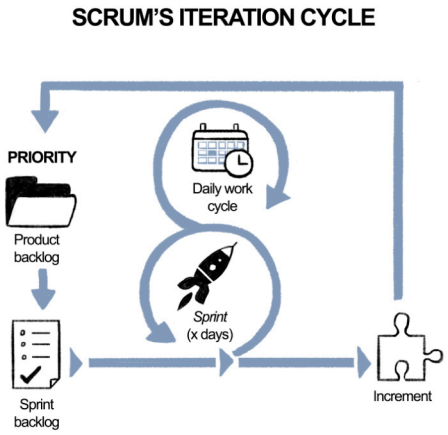
Scrum master

The scrum master is responsible for compliance with the rules of the scrum framework in the project. They ensure that these rules are understood within the organization and that work is done according to them. This role advises and provides the necessary training to both product owner and developers, and they also configure, design, and continuously improve the agile practices of the organization. The aim is for the development team and the client to be able to organize themselves and work with autonomy.

It is also the scrum master's responsibility to moderate the daily scrum meetings, to manage the difficulties of group dynamics that may arise in the team, and to solve any impediments detected during each daily scrum so that the sprint can progress.

Artifacts

Scrum 'artifacts' are the framework's tools: it's basic building blocks. They assist the 'roles' during the 'events.'



Most widespread artifacts

A standard scrum cycle has three main artifacts:

- Product backlog: it records and prioritizes requirements from the customer's point of view. It starts with an initial vision of the product and grows and evolves during development. The requirements are usually called 'user stories,' which are broken down into smaller 'tasks,' usually lasting a maximum of one workday.
- Sprint backlog: it reflects the requirements from the developers' point of view. It is a list of the necessary tasks for a sprint to generate the expected increment.
- Increment: the result of each sprint.

Other artifacts

It's also common to use these:

- Burn down chart: it indicates the pending work and the speed of task completion to deduce if they will all be finished within the estimated time. It is updated daily by the developers.
- Burn up chart: if the burn down chart measures what's left, the product chart measures how much has already been built.
- Definition of Ready (DoR): an agreement that defines when a user story can be considered 'ready' to be broken down into tasks, estimated, and included in a sprint.
- Definition of Done (DoD): agreement on the criteria to consider that a part of the work ('user story') is finished.

Product backlog: the client's requirements

The product backlog is the inventory of features, improvements, technology and bug fixes that must be added to the product through successive sprints. It represents everything that customers, users, and other interested parties expect. Everything that implies work for the team must be in it. Most commonly, entries in this stack are referred to as 'user stories.' Some examples:

- "Allow users to view files published by a member of the platform."
- "Check orders placed by a seller in a range of dates."
- "Offer to query a file via API."

The essential characteristic of this artifact is that it contains live information. It is continuously evolving. Rather than a traditional document of requirements, it is a tool that facilitates the communication of information to the team. At the beginning of the project, this list contains few requirements: only those which are known and best understood at the time. It will be expanded and modified as development progresses. The dynamic character of this backlog allows the product to adapt to changing circumstances.

Team and client usually draw up this list together during an initial meeting, in which they share an overview of the business objective that the project pursues. Once the product backlog has enough stories for a first sprint, it is enough to get started.

After that, the product owner will keep the stories in the backlog in order of priority. The level of urgency will be dictated by how necessary and valuable each feature is.

On the other hand, the degree of concreteness of the user stories should be proportional to their priority. The highest priority stories should be detailed enough to break them down into tasks and move them on to the next sprint.

The tasks of prioritization, detail, and pre-estimation of the stories, before the sprint, are usually called 'refinement.' The product owner and the team can perform them at any time, in a collaborative manner, but grooming should never consume more than 10% of the team's working capacity. Later, the team will make a second, more detailed estimate, during the sprint planning meeting (→ Events), when they'll break down each 'story' into 'tasks.' The responsibility for estimating the foreseeable effort for each element of the subsequent task list (→ Sprint backlog)

belongs to the developers who will do the work. (→ Agile metrics and estimation).

Product backlog's user stories that can be added to a sprint are considered 'ready.' This term (or similar) indicates that the product owner and the development team agree that the story is defined, pre-assessed, and has a size and level of concretion to make I assumable in a single sprint. They have also agreed on the criteria to consider the story 'done' (finished) and on the person or people who will be responsible for verifying that these criteria are met.

To make and maintain the product backlog, it is better to use simple means, known and shared by the whole team. It has to be a radiator of useful information and a tool to facilitate direct communication. User stories can be written down, for example, on sticky notes on a board, ordering them according to their priority, or using a management tool (e.g., Trello) known by the whole team.

The purpose of the product backlog is to describe the state the product will have in the future and that draws the vision shared by the team to plan.

Sprint backlog: the team's tasks

The sprint backlog is the list of all the tasks needed to build the user stories in a sprint. In it, user stories are broken down into smaller units to monitor progress daily, as well as to identify risks and problems without complex management processes.

The whole team collaborates in the creation of this list, during the sprint planning meeting (→ Events), indicating for each task the effort that they estimate it will require. The 'effort' is calculated using a relative measure unit, 'point,' or 'ideal time' (→ Agile metrics and estimation). It is common to use techniques such as poker estimation (→ Practices to make scrum more flexible). Larger tasks are divided into smaller ones so that one task never takes longer than a day's work.

While the product backlog is product owner territory, the sprint backlog belongs to the developers. Only they can change it during the sprint.

It provides direct visual communication and allows the development team to check their progress day by day. Ideally, it is located on a board or wall in the same physical space where they work, so that it is visible to everyone. Some common supports are physical boards, shared spreadsheets, and collaborative project management tools such as Todoist, Flow, or Trello.

It is appropriate to use whatever format is most convenient for everyone, taking

into account the following criteria:

- It should include only the necessary information:
- List of tasks
- The person responsible for each task.
- State in which each task is and ‘effort’ pending to complete it.
- It should serve as a means to record the pending ‘effort’ for each task during the daily scrum meeting.
- It should facilitate communication and information sharing among team members.

The sprint backlog must define and be aligned with the sprint target, which marks a milestone in the progress towards product vision.

Increment

An ‘increment’ is a part of the product that results after one sprint. It should be deliverable, that is: finished, tested, and operational. Prototypes, modules, or parts pending testing are not to be considered increments.

Ideally, in scrum:

- Each element of the product backlog refers to a deliverable functionality, not to internal tasks such as “database design.”
- The team produces an increment for every iteration/sprint.

However, the first sprint is usually an exception. It is often called ‘sprint zero’ when it has objectives such as “contrasting the platform and the design,” which are necessary at the beginning of some projects. They involve design work, development of prototypes, or to contrast tools and working methods.

If the developed part requires documentation, or documented validation and verification processes, these must also be done to consider the increment ‘done.’ It is only finished when it is completely ready to be used and delivered to the client.

The increment is to meet the quality measures required by the product. The "definition of fact", must be known and shared by the whole team, and the increment is not considered finished until it is reached.

Events

This section details the practices and activities that make up the scrum work routine.

- Sprint: it is the core of scrum, all other events work around it. Sometimes it is also called ‘iteration.’ It is the name given to each work phase with a specific objective within the project. The division of the work of sprints, which have fixed and constant duration (timeboxing), allows to maintain a stable progress pace.
- Sprint planning meeting: marks the beginning of each sprint. In it, the team sets the sprint’s goal and the tasks necessary to achieve.
- Daily scrum: a brief daily meeting in which the team reviews their progress. If they identify impediments hindering them, this is the opportunity for prompt detection and searching for solutions. The sprint backlog is updated with the pending ‘effort’ for each task.
- Sprint review: analysis and inspection of the increment produced during the sprint. If necessary, the product backlog is modified.
- Sprint retrospective: meeting at the end of the sprint in which the team analyzes operational aspects of its work methods and creates a plan for improvements, to be applied in the next iteration.

Sprint

The sprint is the core event of scrum, the key to set and maintain the pace of development. A sprint is a limited period (ideally no longer than a month) during which the team builds an increment. The increment, as we already saw in the Artifacts section, must be finished: operation and useful for the client, in conditions to be deployed or distributed.

When starting to use scrum, it is advisable to consider the sprint as the even that contains all the others:

- It marks the daily progress rhythm and allows to visualize and share it during the daily scrum meetings.
- It sets a fixed pace for checking the product's evolution, at sprint planning and review meetings.
- At the same rhythm, retrospective meetings are introduced, to reflect and improve the working methods.

In more mature scrum implementations, however, it is possible to consider that the scope of the sprint is only the construction of the increment, leaving the meetings aside.

It may be in the team's best interest, for example, when they need to calculate the sprint speed without including planning, review, and retrospective meetings: only working time. Or to have more flexibility when carrying out sprints of different durations, or to separate the frequency of retrospective meetings from the frequency of sprints.

Sprint planning meeting

This meeting marks the beginning of every sprint. In it, the team and the product owner consider the client's business' priorities and needs, to determine what functionalities should be added to the product by the end of the sprint, and how.

The scrum master moderates the meeting, or a team member in the absence of a scrum master. The product owner and the developers must attend, and it's open for others involved in the project. It may last up to a full day, depending on the volume or complexity of the user stories.

The meeting must answer tree questions:

1. Why is this sprint valuable?

The owner of the product exposes how the product can increase its value with the result of the sprint to be performed. This question determines what the objective of the sprint is.

2. What can be done in a sprint?

Once the value increase expected by the product owner has been shared, the developers determine the elements of the product backlog they are going to make. In this process they can refine the elements of the backlog that may need further specification or explanation.

3. How is the selected work going to be done?

The developers break down each element of the product backlog into tasks that should be able to be done in one day's work or less.It is recommended to articulate the meeting in two parts of similar duration, separated by a break:

- First half: what will be delivered at the end of the sprint.
- Second half: how the increase will be achieved, estimating the working time and the necessary requirements.

Preconditions
The organization has determined and allocated the available necessary resources for the sprint. The highest priority user stories are 'ready'; they have a sufficient level of concreteness and a rough estimate of the effort they'll require. The team has sufficient knowledge of the technologies and the product's business to make estimations, and to understand the business concepts presented by the product owner.

Input	Output
Product backlog. The current state of the product (except for 'sprint 0'). Speed of the previous sprint, to estimate the achievable workload. Circumstances of the customer's business and the technological scenario and value that the owner of the product expects to obtain.	Sprint backlog. Sprint duration. Date for the review meeting. Sprint goal.

First half: why is dis sprint valuable and what will the team deliver at the end of the sprint?

The product owner presents the top priority user stories during this first half of the meeting. They explain what the client needs and what they expect to achieve with the coming sprint. If the product backlog has changed significantly since the last meeting, they should also explain the causes of these changes.

The goal is for the whole team to understand, with a sufficient level of detail, what the increment produced with the sprint should be. The presentation should be open to questions, and team members may request clarifications. They can propose suggestions, modifications, and alternative solutions, and modify the backlog accordingly.

This meeting encourages participation and the cross-fertilization of ideas, to add value to the product vision.

After reordering and rethinking the stories in the backlog, the team defines the 'sprint goal': a phrase that synthesizes what value will be delivered to the customer.

Except for sprints dedicated to the collection of tasks unrelated to each other, defining a slogan together during the meeting guarantees that the whole team understands and shares the same purpose. During the sprint, it will serve as a reference to guide decisions.

Second half: how will the increment be achieved?

This second part works as a team meeting. All developers should be present, and they are in charge of breaking down, estimating, and assigning the work. The role of the product owner is to answer questions and check that the team understands and shares the client's goal.

The team breaks down each feature into tasks and estimates the effort for each one, thus making up the sprint backlog. They establish the priorities for the first days and assign them, taking into consideration the knowledge and interests of each member and trying to distribute the workload homogeneously.

Functions of the scrum master during the sprint planning meeting:

- Ensure that this meeting takes place before every sprint.
- Ensure that the product owner has prepared the product backlog for the meeting.
- Help maintain an open dialogue between the product owner and the team.
- Ensure that the product owner and team reach an agreement about what the increment will include.
- Ensure that the team understands the client's vision and the needs of their business.
- Ensure that, by the end of the meeting, these have been objectively determined:
  - The items of the product backlog that will pass to the sprint.
  - The sprint goal.
  - The sprint backlog. All tasks should be estimated.
  - The sprint duration and the date of the review meeting.
  - The definition of done that will determine that the increment is finished.

Daily scrum

The daily scrum or stand-up meeting is supposed to be short, no longer than 15 minutes. In it, the team synchronizes and establishes the plan for the next 24 hours.

Input	Output
Sprint backlog and burn down graph, updated with the information from the previous meeting. Information on each developer's progress.	Updated sprint backlog and burn down graph.

Meeting format

It is recommended to have the meeting standing up by the board featuring the sprint backlog, or by a kanban board, to share information and update it on the go.

All developers must attend this meeting, and it's open to others involved in the project, although they do not intervene.

Each developer shares what they have achieved since the previous daily scrum, what they plan to do until the next one, and if they are having any problems or foresee any impediments. They estimate the pending effort for their tasks and update the sprint backlog accordingly. At the end of the meeting, the development team updates the burn down graph (→ Practices to make scrum more flexible > Burn down graph).

The developers are responsible for this meeting, not the scrum master. Also, it should not be conducted as a control meeting, but as a moment for the team to catch up and communicate. They share the status of their work, check their progress, and collaborate to solve difficulties. The scrum master, however, will be responsible for taking the appropriate measures to solve any identified impediments after the meeting.

Sprint review

This meeting is held at the end of the sprint to check the increment. It usually lasts from one to two hours, longer in case of more relevant or complex increments that require it. It can take a maximum of four hours. The list of attendees includes developers, product owner, scrum master, and all involved parties who wish to participate.

This meeting marks, at regular intervals, the pace of development, and it serves to assess the evolution of the product's vision. It's a space where all interested parties can share their suggestions. The developers share the user stories they planned, which ones they developed, which ones they didn't, and the impediments they encountered. It's important to set the right expectations from the beginning of the meeting.

Input	Output
Finished increment.	Feedback for the product owner: • The current progress of the project. • Valuable information observed during the sprint. The date for the next sprint planning meeting.

Meeting format

It is an informal meeting to show the result of the iteration in action. Depending on the project's characteristics, technical or user documentation may also be provided.

Later, with all the information, the product owner will deal with possible modifications.

Recommended protocol:

- The development team presents the sprint goal, the list of functionalities, and which ones have been developed.
- The developers show the built parts in action. After this, there is a session for questions and suggestions. During this part of the meeting, the product owner and the developers can share their opinions to improve the product's vision.

Sprint retrospective

This meeting takes place after the review of each sprint, before the planning meeting of the next one. It lasts from one to three hours. It's one example of a scrum practice that was not part of the original framework but has consolidated over time.

In it, the team reflects on their way of working. They identify strengths and weaknesses to strengthen the former and plan improvement actions on the latter.

The fact that it normally takes place at the end of each sprint sometimes leads to the error of considering it a 'sprint review' meeting, when it is advisable to treat them separately. They serve different purposes. The objective of the sprint 'review' is to analyze 'what' is being built (the product). The objective of the sprint 'retrospective' focuses on 'how' it is built (the working framework).

The sprint retrospective meeting involves: the developers, the scrum master and the product owner. The product owner should act like a member of the team rather than a "customer". They should be involved and understand the principles and values of scrum. If this is not the case, the scrum master should act as a facilitator to achieve their commitment and participation. The product owner's perspective can be very valuable during retrospective meetings, as their job is to keep the focus on what the client needs.

Agile measurement and estimation

In this section, we will explain some basic concepts about agile estimation that may come in handy.

However, as a general rule, the fewer metrics, the better. The objective of scrum is to produce the highest possible value for the client consistently, so it is always worth asking how the use of an indicator contributes to said value. Measuring is costly and should serve a greater purpose, not become an end in itself.

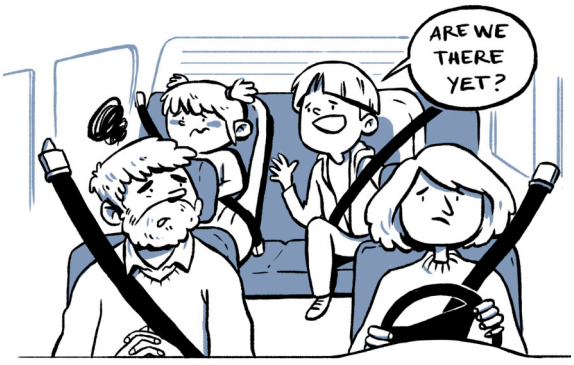
The goal of estimation tools should be, first and foremost, to be able to plan the duration of each sprint realistically and to set deadlines. In teams that are starting to work using agile methods, they can also help in establishing a development pace. And, in some cases, estimation can serve to synchronize teams.

Two key concepts:

- You don't measure the work done, but the work that remains.
- It is measured using relative units.

Are we there yet?

Measuring our work may be necessary for two reasons: to record what has been done, or to estimate in advance what needs to be done. In both cases, we need an objective unit and criterion of quantification.



Measuring the finished work is not difficult. We can do it using units related to the product, such as completed stories, or resources, such as costs or working time.

But agile project management does not evaluate progress by keeping track of completed work. If a task was supposed to take a week and three days have already passed, that does not necessarily mean that we will finish it in four days. Unforeseen events occur, or we

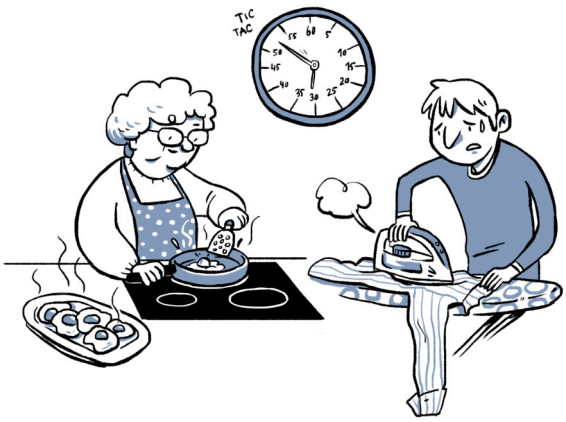
may find a shortcuts, making the time estimated at the beginning inexact. Progress is not determined by the we've done, but by the work we have left.

Children perfectly understand this. When on a trip, what is it that they ask, again and again? How long they've been in the car? Of course not, they don't care! The question is: how much longer until we arrive?

Other processes in the organization may need to record the invested effort and resources, but calculating the progress of the project is different. Thus, scrum measures the work that remains. Firstly, to estimate the effort and time expected to perform specific tasks, user stories, and 'epics' (large user stories). And second: to determine the advance of the project in general, and each sprint in particular.

Relative units: story points

We can't accurately predict the amount of work needed for a requirement or a user story, because they are rarely problems with a single solution. And even if we could make accurate predictions, the complexity of such a metric would be too heavy for agile management.



It is not possible to accurately estimate the amount of work involved in a user story. Consequently, it is also not possible to know in advance how much time it will require, because the uncertainty of time estimation compounds the uncertainty of work/effort estimation. We can't estimate the quantity or quality of the work performed by the "average person" per unit of time, because the differences from one person to another are too significant. What is more: the same task performed by the same person will require different times depending on the circumstances.

For all these reasons, when estimating in an agile way, it is preferred to use relative units. In agile management, these are called 'story points' or just 'points.'

Each organization institutionalizes its work metric, its 'point,' according to its specific circumstances and criteria. It is an abstraction based on a task that is familiar, well-known, and easy to estimate by everyone. In a programming team, the 'point' might be equal to preparing a login screen. For a graphic design team, it could be the layout of a leaflet. Once established, this metric should be understood and shared by all.

Story points help to dimension the size of tasks by comparing them with an already known one, and to understand the difficulty a task might present for each team member, according to their specialties. An example to illustrate this can be the effort required to fry an egg. The estimation of how many 'fried eggs' it would cost to iron a shirt will depend on the person. Someone may be very skilled at frying eggs, but have never ironed a shirt, and estimate that it would cost "8 points" (8 fried eggs). Someone very used to housework, on the other hand, might estimate the task as "1 point" (1 fried egg). Both are right. The person estimating should be the one who will perform the task.

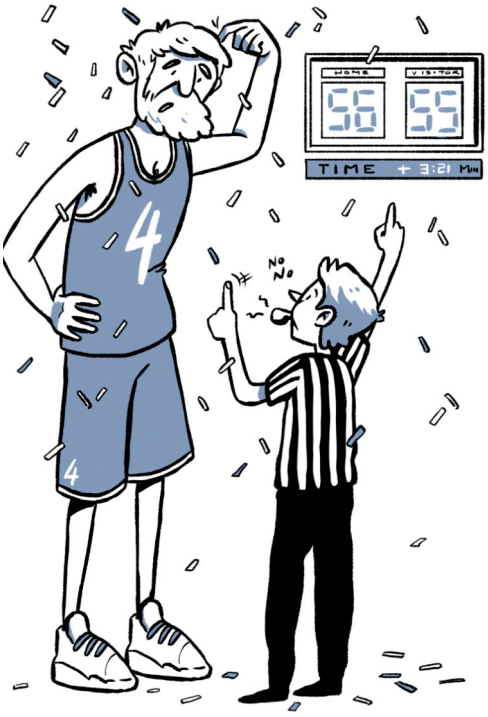
Finally, we may use this to estimate speed: in scrum, this is equal to the amount of work done by the team in a sprint. For example, a speed of 150 points indicates that the team makes 150 story points per sprint.

However, when leaving the standard scrum framework, we may find sprints of different lengths. When this happens, the speed can be expressed in time units instead. The average speed of the team would be x points per week/month.

Measuring speed with story points has received criticisms that are worth considering. For example, it has been observed that a misuse of this practice can lead to unwanted work dynamics (→ Practices to make scrum more flexible > Estimation without story points, #NoEstimates).

Real and ideal time

When calculating the duration of a sprint, we tend to estimate the effort in 'ideal time': time spent working in ideal conditions. It is what it would cost us to perform tasks in a state of flow, focused, without any distractions or impediment. It is important to be aware of the difference between 'ideal time' and 'real time' when estimating.



How long does a basketball game last? In exact playing time, the answer is 40 minutes. In the same way, we could say that writing a report takes us one hour. But the actual duration of a basketball game usually takes more than an hour, because it cannot end in a draw. It gets longer because of time outs, fouls, half-time, and extra time. And sometimes a report, which would usually take us an hour of 'ideal time', ends up taking up half a day's work.

Using these two concepts of time can help us organize our work more objectively and avoid the stress of unattainable goals.

PART II: VALUES AND PRINCIPLES

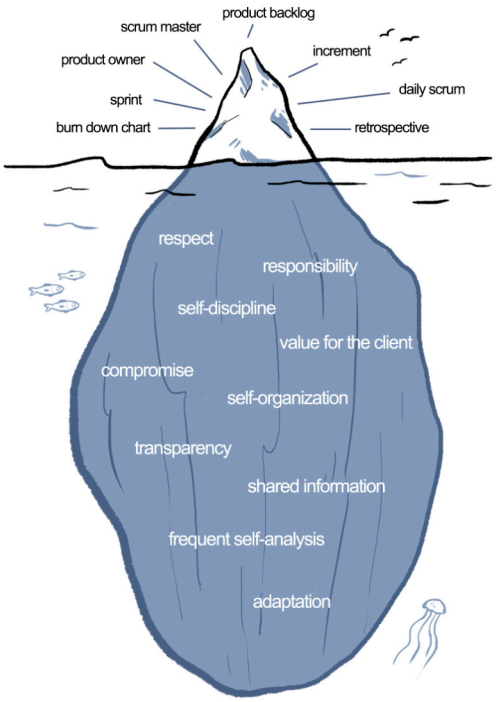
INTERNALIZING AND ADAPTING

Scrum values and principles

There is something that is not usually discussed in scrum courses: the tools that we have just studied, the practices and tools in the first part of this manual, do not work without an aligned set of values and principles.

This happens because scrum 'practices' are not 'processes.' They are not guidelines that guarantee results no matter who carries them out.

Scrum is based on the tacit knowledge of people, as well as on organizational values. The practices are just the branches of a tree that won't bear fruit without good roots. Something that should be remembered from time to time, as not to fall into the error of focusing only on the tools.



Management models should serve to facilitate the development of certain working values and principles within the organization.

We may find companies where these principles are so internalized that they develop their own agility model, with practices and techniques which are completely different from those we have seen. Maybe from other agility models, maybe completely original. And they still obtain innovative and quality products. On the other hand, there are companies that apply all the practices of the scrum standard framework, but only get alienated, unmotivated, stressed employees, and mediocre results.

There needs to be 'flexibility' in order to adapt what is learned to the reality of

each company and project. The aim is for the organization to be agile as a whole, capable of moving forward in innovative and unstable scrum scenarios.

Good management should be up to date with the latest tools and then look for ways to adjust them to their team, taking advantage of what works and modifying or discarding what does not. There is no point in trying to make people adapt to methods that do not serve them well.

***

We may divide values and principles depending on whether they are behind agile practices or organizational culture. The ‘principles’ support the practices; the ‘values’ support the culture.

Principles

Artifacts, events, and other agile techniques are intended to base work on the following principles:

- **Delivering value.**

The client and the team collaborate in order to share a common understanding of the product's vision. During the development process, this will guide all efforts to deliver real value to the client.

- **Continuous improvement.**

The team reflects on their work methods, questioning their effectiveness, and adapting them. The same self-critical effort is also applied to the improvement of products and services.

- **Incremental and iterative development.**

The final product is not built according to a detailed and complete initial plan. Rather, a 'minimum viable product' is built, and then increments are added to it.

- **Sustainable working pace.**

The pace should be able to prevent Parkinson's Law and the stress of finding out problems that could cause delays too late.

- **Constant attention to excellence.**

Use of techniques that ensure the quality of products and services and allow errors to be detected in advance or right when they occur.

- **Operations visibility.**

Information is clearly shared to facilitate collaboration, identify impediments early, and allow the whole team to know the status of the product and contribute ideas.

- **Global timing and synchronization.**

This principle is most relevant when trying to synchronize multiple teams working on related products or services. The aim is to predict the frequency of meetings and delivery dates.

- **People over processes.**

The collective intelligence of the team, their tacit knowledge, is directly responsible for the quality of the product.

Values

Company culture is a sum of organizational and governance traits. They can accelerate or slow down the development of agility. In organizations with a culture based on industrial work values, based on processes, agile principles deploy modest results. Collective intelligence and innovative value flourish in organizations with values that enhance:

- Assertiveness.
- Talent appreciation.
- Clarity.
- Confidence.
- Non-hierarchical structure.
- Common purpose.

Finally, managerial support is necessary. Those in charge should share a similar culture, stay involved, and support people with sufficient training and resources.

***

For the purpose of this guide, which is to provide you with all the tools to perform the functions of a scrum master, we are going to introduce the 'principles' associated with scrum roles, artifacts, and events. Finally, we'll add a few more practices that are commonly used, although they are not strictly part of the standard scrum framework.

If you are interested in studying these agile values and principles in more depth, Scrum Manager® has developed a second manual available at Scrum Level: https://scrumlevel.com.

People and their roles

Although artifacts and events help to develop the agile principles, they have to be taken up by people first. Without talented, committed, and responsible people, practices will be useless. And without the influence of certain company culture values (→ Scrum Level), talent and commitment cannot grow.

> "If you have a team of outstanding engineers that are using excellent engineering tools, understand the business and technology domains inside out, and aren't interrupted and have all the resources they need… then you can use scrum. Whilst true that people like that can build an increment of software every iteration. That's good.
>
> However, scrum works with idiots.
>
> You can take a group of idiots that maybe even didn't go to school, don't understand computer science, don't understand software, hate each other, don't understand the business domain, have lousy engineering tools… and uniformly, they will produce crap every increment. This is good! You want to know at the end of every iteration where you are." (Schwaber 2006)

Product Owner

The product owner is responsible for the product backlog. They manage it and prioritize the requirements it contains, proactively, and making changes when deemed necessary. They make the early and continuous delivery of value to the client possible. They represent the customer and communicate the product vision to the team so that everyone is aligned towards the same goal. Their attitude is key to facilitate honest and fluid communication.

Developers

The developers team as a whole is responsible for producing an increment for the customer with each iteration, for maintaining a sustainable work pace, and for applying self-awareness to improve their own results and work methods. Lastly, they must be aware of the importance of their collective talent and intelligence, not only on an individual level.

Scrum master

They ensure that the scrum framework is applied and works effectively, by moderating the daily scrum meetings and handling the resolution of impediments identified during these. They assist the team so they can smoothly move forward.

Having a dedicated scrum master is recommended when the product owner or the team has little experience using scrum, or in large organizations with a constant flow of staff turnover or training. In small, stable teams with experience in agile methods, these responsibilities to use and improve the scrum framework may be internalized and assumed by the team as a whole instead.

Artifacts

Artifacts enable certain agile principles to develop:

Product backlog

Delivering value: it allows embracing the variability of the customer's business environment and focus all efforts on the stories that provide the greatest value according to the circumstances.

Iterative and incremental development: the product backlog, unlike a traditional requirements document, makes this agile principle possible because it's a living document. It allows changes in the user stories it contains and their priority.

Visible operations: it is an information radiator through which the product owner and the team can share the vision of the product at all times.

Sprint backlog

Iterative and incremental development: this artifact delimits the work contained in each increment (sprint) and establishes a development pace.

Visible operations: it's a tool for the team's internal communications. Everyone has access to it and serves to know the state of the sprint at a glance.

Increment

Delivering value: presenting a part of the finished product ready to be used at the end of each sprint allows checking if the team's efforts are producing value.

Iterative and incremental development: the delivery of the increments help to set a pace of progress.

Events

Sprint

Continuous improvement: short iterations make it easier to set milestones to stop and reflect on how to improve the quality of the product and the team's work techniques.

Iterative and incremental development: as the basic time unit during which each increment is built, the sprint is the core gear around which the whole iterative development revolves.

Sustainable working rhythm: it marks the pace of progress.

Operations visibility: it allows identifying impediments early.

Global synchronization: it makes delivery and meeting dates predictable through timeboxing, and it facilitates the synchronization of different teams.

Sprint planning

Delivering value: during this meeting, the team and the product owner collaborate directly, deepening their shared knowledge of the product.

Daily scrum

Operations visibility: in this meeting, the team catch up, share the current state of their tasks, and collaborate to help each other and solve impediments.

Sprint review

Delivering value: once again, the team is working directly with the product owner in this meeting.

Continuous improvement: the purpose of this meeting is to analyze the increment, to draw conclusions that will help shape the next sprint.

Sprint retrospective

Continuous improvement: the team analyzes its own work strategies, to decide what to keep and what to modify or eliminate.

Practices to make scrum more flexible

The professional community is constantly developing new ‘practices’ to shape product backlogs, user stories, communicate the product’s goals visually, conduct events and meetings, and estimate tasks. The two most commonly used together with the standard scrum framework are the burn down graph and poker estimation. We’ll explain how they work alongside other practices:

- Burn down graph.
- Poker estimation.
- Estimation without story points.
- #Noestimates.
- Kanban.
- Pair work.
- Error prevention techniques.
- Wall estimation.
- Burn up graph.
- Diagrams for retrospective meetings.

These practices are just a few of the agile management techniques that can help customize your management model for your project or team. Rather than exploring them in depth, they are intended as examples to encourage the reader to investigate and experiment.

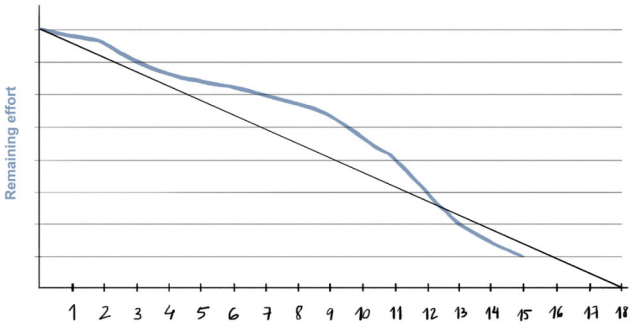
Burn down graph

The developers updates it during the sprint, daily if possible. It serves to monitor progress and detect deviations that may compromise the delivery date.

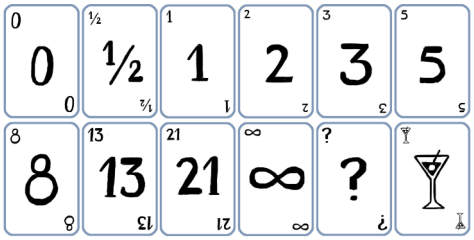
- On the Y-axis: the 'points' of work that remain to complete all tasks.
- On the X-axis: duration of the sprint in days.

Each developer estimates the remaining effort for each of their tasks every day. With this information, the graph can be updated, reflecting the total effort that remains to obtain the increment. The ideal progress of a sprint would be represented by a straight diagonal that reduces the pending effort gradually until the delivery date. However, this is rarely the case, and a sustainable pace is possible without a perfect diagonal line.

If the line stays for several days well above the diagonal, however, it is a sign that the sprint has been underestimated and will require more time. When the opposite happens, and the line descends faster than the diagonal, it will be finished earlier than expected



Poker estimation



Jane Grenning devised this planning game to conduct meetings during which the team estimates the effort and duration of tasks (sprint planning meetings). The original model consists of 8 cards, with the values ½, 1, 2, 3, 5, 6, 7, and infinite.

Each participant has a set of cards, of which they can reveal one or several to make their estimation. The sum they reveal is equal to the number of 'points' they estimate for the task.

The most widespread model of poker estimation, however, uses a deck of cards with the Fibonacci sequence, as in the illustration. In this case, participants can't add up numbers. They show one card for each task: the one with the closest figure. This variant is based on the fact that, as the size of the tasks increases, so does the margin of error. Thus, if a person believes that the appropriate size of a task is 6, he or she is forced to reconsider. They'll either accept that some of the perceived uncertainty is not such and opt for a 5 or go for a more conservative estimation and raise an 8.

It is common to add a card with a question mark to indicate that an estimate cannot be made, for whatever reason. It is also possible to include another card with some allusive image to suggest a break. The infinity symbol means that the task exceeds the maximum effort value and that it should be broken down into smaller units.

When the estimates are very different, the person in charge of the meeting may choose what to do. People with the most extreme estimates may explain the reason behind their choices, and then repeat the process to see if other team members have changed their minds. Another option is to set aside the task for the time being and estimate it again later. Or ask the product owner to break down the task and assess each of the resulting sub-tasks. One can also decide to opt for the most optimistic or the most pessimistic estimate or to take the average. It will depend on the task and management style of the team.

The use of poker estimation can be fun and make meetings more dynamic. It also

avoids circular discussions between different implementation options. It allows all assistants to participate, it helps to reach consensus without long discussions, and it reduces the time to estimate each functionality.

Estimation without story points

Story points have received criticism due to the tendency of misinterpreting their function. When they're used to measure the speed of several teams in a company, for example, they can cause tension. Story points are not a good measurement for productivity: they are relative, and they depend on each team's context. A higher number of story points does not equal more effort or better quality, but this isn't necessarily intuitive. Teams may start to bloat estimations, and select stories not by priority but in order to get more points. Either because they want to keep their average speed, or to compete against other teams. All of this works against the project and impairs collaboration.

There are some alternatives to estimate while avoiding these issues:

- Shirt sizes: this is an intuitive system that still allows for some detail, in which every story is given a size out of XS, S, M, L, and XL.
- Goldilocks: it uses only three measurements: too big, too small, or adequate.

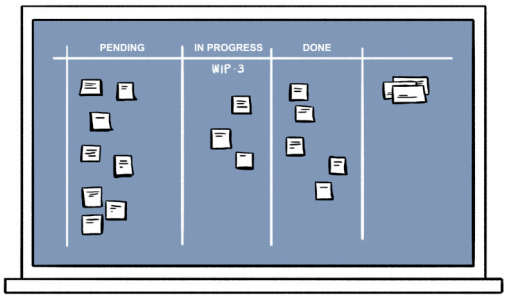
Both systems require looking for other ways to synchronize teams and identify bottlenecks and other issues in advance. But they allow to prioritize tasks intuitively and quickly spot large stories in need of deeper analysis.

#NoEstimates

This is another reaction to the misuse of estimation techniques in agility: to stop using them altogether. Sometimes, in high-performance teams that are used to working in an agile way and can sustain a continuous workflow, this approach can make sense. It helps avoid Parkinson's Law, according to which "the time required to perform a task tends to extend to all the time available to perform it."

Opting for this way of work will also require to find alternative ways to synchronize deliveries with other teams.

Kanban



Kanban is a popular visual technique to manage continuous delivery flows. That is, flows without timeboxing, with-out dividing the work into sprints of a set duration.

The team writes down the tasks or user stories and positions them on a board. Their location indicates their current state. The most common ones in kanban boards describe progress stages: “pending,” “in progress,” and “finished.” They are ordered progressively from left to right. The format of each board responds to the circumstances of the product and the equipment. It can include additional states such as “tested” or “validated.”

This board not only helps to manage the working pace visually and clearly, but it is also a great tool for sharing information among the team. It shows immediately, with each update, the status of tasks and whether bottlenecks are forming.

The absence of temporary milestones like sprints prevents Parkinson’s law. Conversely, this absence could lead to delays due to perfectionism or procrastination. But Kanban’s WIP (work in process), structure, and visibility mitigate this negative effect. The WIP is the maximum number of tasks that can be at the same time in the same stage. A “WIP=3” indicator in the “in progress” stage means that the team cannot be working on more than 3 tasks simultaneously.

The visibility provided by the kanban board allows the whole team to identify bottlenecks and downtime early own, to adjust or reassess priorities. With the information they obtain, they can make adjustments to improve their flow and assign team members efficiently.

Pair work

This concept is best known among IT teams, where it is known as pair programming.

It consists of assigning two people to perform the same task simultaneously, normally alternating execution and supervision between them. While one of them carries out the task, the other watches what they are doing and makes any observations that may be appropriate. This practice can be suitable when the quality of the result depends, above all, on the knowledge of the person carrying it out.

It is the reason why, for example, trains run with an assistant and a driver when the technology cannot prevent human errors 100%, or we feel safer knowing that in the plane's cabin there are two pilots.

Error prevention techniques

An example of this type of technique which also comes from IT is test-driven development (TDD). It consists of first developing the tests that the code must pass, and then the code.

In more general terms, agility sometimes uses poka-yoke techniques and andon control devices. Both concepts come from lean manufacturing production frameworks.

Poka-yoke fail-safe techniques can serve for:

- Making human error impossible: an example is plugs designed so that they cannot be wrongly coupled.
- Highlighting the error in an obvious way when it occurs: this is what spell checkers in text editors do, or syntactic programming checkers.

Andon control systems are specific to the production process. They usually consist of indicators with different colored lights or graphic representations that reflect the normal operation of the system or failures. They should be in the workplace and very visible, so they can immediately alert the team.

Wall estimation

It is a technique to estimate and prioritize lists of user stories, usually the product backlog. The developers places sticky notes with the stories on a wall. The smaller ones on the left, the bigger ones on the right. Then, the product owner sets their vertical position depending on their priority: the higher the sticky note, the more urgent.

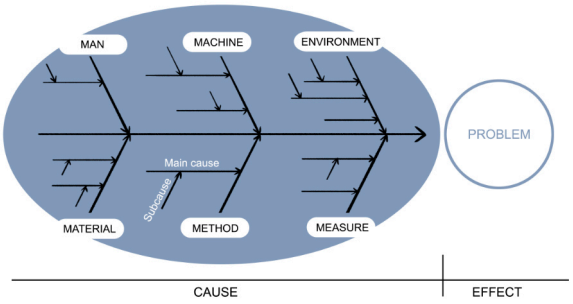
Burn up graph

It is a planning tool commonly used by the product owner. It often shows three estimates: pessimistic, realistic, and optimistic. The three appear on a Cartesian diagram that represents, on the ordinate axis, the estimated effort to build the different stories of the product backlog, and on the abscissa axis, the time measured in sprints. Each new increment goes on the vertical axis in the position corresponding to the estimated effort to build all the stories it includes.

It is a tool for agile development, a living document that should not represent stable plans, but rather, serve to estimate the product's future progress.

Ishikawa and tree diagrams

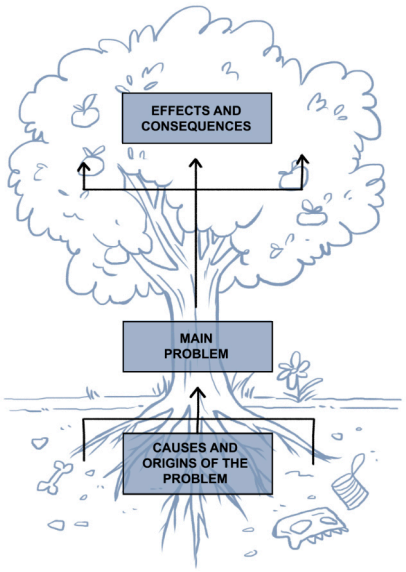
The following diagrams can be useful in retrospective meetings, when the team needs to reflect on the last sprint to improve their working methods. It is also common in quality management.



The Ishikawa diagram, also known as "spine diagram," "fishtail diagram" and "cause-effect diagram," shows the causal relations acting on a problem that needs to be analyzed.

After identifying the problem, which is the central axis of the diagram (a central horizontal line), the team lists possible causes to explain it, which may have sub-causes.

***



Tree diagrams are often used to reach constructive solutions from the analysis of a problematic situation. They use the elements of a tree (roots, truck, branches, fruits…) to represent the means or resources available to solve the problem and the possible outcomes. There isn't a single way to draw this type of diagram nor a standard system, but as an example we recommend to have a look at the way Khurram Bhatti uses them.

ANNEXES

Resources

Last version of this book:

- https://scrummanager.com/info/resources.html

Self-training platform:

- https://www.scrummanager.net/oks/

FAQs:

- https://scrummanager.com/website/c/info/faqs.php?about#scrum_manager

Scrum Manager® quality reviews

If you have participated in an official Scrum Manager® training activity, we would like to ask you to please rate its quality. Your opinion helps us to maintain the level of our materials, academies, courses, and teachers.

All information you provide will remain anonymous. You can send your comments through our website's members area:

- https://scrummanager.com

Bibliography

Bau 1969: Bauer F., Bolliet L., & Helms H., *Software Engineering. Report on a conference sponsored by the NATO*, 1969.

Beck 2000: *Extreme Programming Explained,* 2000.

Khurram Bhatti, 2019: https://khurrambhatti.com/2019/03/08/tree-retrospective-model/

Nonaka 2004: Nonaka I., & Takeuchi H., *Hitotsubashi on Knowledge Management,* 2004.

Nonaka 1986: Nonaka I., & Takeuchi H., *The New New Product Development Game,* 1986.

Nonaka 1995: Nonaka I., & Takeuchi H., *The Knowledge-Creating Company, 1*995.

Orr 2002: Orr K, *CMM versus Agile Development: Religious wars and software development,* 2002.

Schwaber 1995: Schwaber K., *SCRUM Development Process - OOPSLA 95,* 1995.

Schwaber 2006: *Scrum Et Al,* 2006.

Turner & Jain 2002*: Agile Meets CMMI: Culture Clash or Common Cause?*, 2002.