

v. 5.0 (2026)



SCRUM MASTER



Scrum Manager®

Scrum Master

Agile management of projects, teams and organizations

Version 5.0 (2026)

Scrum Manager ®

Version date: October 2026.

Author of the version: the Scrum Manager® team.

Illustrations: María de la Fuente Soro and Ana Andrés Soria.

Uncovering Better Ways SLU is the publisher and owner of the distribution rights, which it releases under the terms of the Creative Commons Attribution-NonCommercial 4.0 International license (CC BY-NC 4.0). Scrum Manager's official trainers and centers are licensed under the CC BY 4.0 terms for their training activity.

Rights registered with Safe Creative. Registration number: 2610167341144

Table of contents

Table of contents	3
Preface.....	5
Introduction.....	7
Agility.....	7
Processes and predictive management	9
The <i>Agile Manifesto</i>	10
The twelve principles of the <i>Manifesto</i>	13
Where Scrum came from	14
Development, work, and knowledge	15
Predictive management and evolutionary management.....	17
Practices, principles, and values.....	18
AI in knowledge work.....	19
What changes and what stays	21
Part I — The Scrum cycle.....	23
The Scrum cycle.....	24
Empiricism.....	26
The committed and the involved	27
The Scrum Team and AI.....	28
The Product Owner	30
The Developers	31
The Scrum Master.....	32
Scrum artifacts.....	33
Work items	34
The Product Backlog.....	36
The Sprint Backlog.....	37
The increment.....	38
The Definition of Done.....	39
The Definition of Ready	40
Scrum events	41
The sprint	42
Sprint Planning.....	43
Planning with AI on the team.....	44

The Daily Scrum.....	45
The Sprint Review.....	47
The Sprint Retrospective	48
Measuring progress.....	49
Estimating	51
Output and outcome.....	54
Part II — Principles and values.....	56
Scrum principles and values	57
The principles of Scrum.....	58
The values of Scrum.....	59
People above roles	60
Artifacts and events in service of the principles.....	61
Ways of working for quality.....	62
Working in pairs	64
Part III — The Scrum Master's craft in AI-enabled teams.....	66
A role that evolves	66
Facilitating the human-AI work system	67
AI working agreements for the team.....	68
AI literacy	69
Quality and workslop.....	70
Impediments in AI-enabled teams.....	71
Learning and developing people	72
Psychological safety and change	73
Antipatterns in AI-enabled teams	74
Scrum with AI beyond software.....	75
Appendix.....	77
Glossary	77
Bibliographic references.....	80
Information, resources, and professional community.....	83
Continuous improvement and quality control	83

Preface

This is the syllabus for the [official Scrum Manager® Scrum Master certification](#). It explains how to implement and improve Scrum as applied to the agile management of projects, teams, and organizations, in the setting where most of them work today: with artificial intelligence built into their day-to-day work.



It is written for anyone interested in the agile management model known as Scrum, whether to apply it in their own day-to-day work or in their team's, or to learn how to manage projects of any kind in an agile way.

Although the model first took shape inside technology companies, today you'll find it in innovative environments of every kind, and the common thread is always the same: knowledge work, instability, and fast, constant change. This syllabus will suit you if you work in any of the so-called "knowledge companies" and, above all, if **the value of what you produce depends on the talent of motivated people** rather than on the processes and tools they use.

This version brings the use of artificial intelligence in teams into the whole syllabus. AI doesn't get a chapter of its own; it appears wherever something changes: in the roles, in the artifacts, in the events, and in the work of the Scrum Master. The position of this guide is that Scrum's principles and values hold, and matter even more, while many practices evolve, and that each team must decide how to make that change in its own context. The examples come both from software development and from other knowledge teams: marketing, human resources, legal, education, or operations.

The content is divided into four blocks:

- The **introduction** puts the birth of Scrum in context, defines what agility means in a business setting, and explains what has changed with the arrival of AI. It also presents the distinction Scrum Manager® draws between agile "practices" and agile "principles and values," which is key to everything that follows.
- **Part one** focuses on the most widespread **Scrum practices**: the roles, artifacts, and events that have settled into a standard, and what AI changes in each of them.
- **Part two** digs into the **principles and values** those practices grew out of, and into a few techniques that, while not part of the standard framework, round out the agile management toolkit.

- **Part three** deals with the **Scrum Master's craft in teams that work with AI**: a role that evolves and takes on new areas of responsibility.

Not all the practices this guide describes are equally established. Alongside those that belong to the consolidated knowledge of the profession, the integration of AI has brought others that are gaining adoption quickly and others that have only just appeared. The sections that deal with these newer practices are marked with a box at the start, carrying one of two labels:

- **Consolidating trend**: a practice that is gaining adoption quickly but is not yet widespread.
- **Emerging trend**: a recent practice, highly relevant and highly volatile.

The content of those sections is professional information on where the field was heading at the time this version was published (October 2026). It is worth knowing, but **it is not part of the certification exam**.

Discussions of Scrum are full of terms that take on a specific meaning within the world of agile management. The syllabus follows three conventions:

- The key terms of the craft are used consistently throughout, so they are easy to find and to look up.
- Concepts with an established name in the profession keep that name, with a brief explanation the first time they appear.
- The names of the framework's roles and artifacts, such as Scrum, Product Owner, or Sprint Backlog, keep their capitals, as does the Definition of Done (DoD).

Introduction

Agility

Agile management emerged as the antithesis of a particular management model we'll be referring to often: predictive project management. Both have their merits, and each proves more useful in certain industries. The predictive approach is built around planning, working out a budget, and setting delivery dates. If the project is finished on the agreed date, within budget, and with every feature from the original plan, it counts as a success.

Reasonable as that strategy sounds, in industries that change rapidly and constantly it has many drawbacks. That definition of success holds up in a stable environment, with products that result from scrupulous attention to processes and protocols.

Predictive management is a product of the Industrial Revolution: it comes from construction, from the car industry, from the factory floor. If what the client wants is a house, it will have to be built so that it's solid and safe and meets the needs of the people who will live in it and, ideally, delivered on schedule and within budget.

Today we build and sell products that bear no resemblance to a house. They can be intangible, like a film, a mobile app, or a communication campaign. You can try new things out during development and see empirically what works and what doesn't, adjust on the fly, and start from a rough first version with the bare essentials and grow from there.

The landscape can shift, and a feature that looked essential at the outset may be obsolete by the delivery date; or a competitor may launch something new that forces you to rethink priorities. Staying competitive means responding quickly in uncertain settings, without stable requirements, with clients who need to use the product as soon as possible and improve it continuously, and with products where innovation is a core part of the value.



These reasons, and others we'll look at, are what led people to question predictive management models, which were out of step with what the so-called "knowledge

companies" needed: those that develop products or services on the basis of knowledge rather than tools and processes.

In those markets there's no point starting a project with a closed plan. What's needed are strategies that deliver tangible results early and make it possible to respond to change in time, because the product is built at the same time as requirements are revised and added. There are clients who need a first version with minimum functionality within weeks, not a complete product in one or two years, so they can get a new concept to market and build its value continuously.

Where we come from, and why agility tends to get associated with IT

Knowledge evolves along a dialectical pattern of thesis, antithesis, and synthesis. Every thesis gives rise to an antithesis that exposes its problems and contradictions. The antithesis turns out to be inadequate in its own way, and out of the clash between the two comes a third moment, the synthesis, which offers a new understanding of the problem.

It is at this point that the earlier thesis and antithesis are reconciled and transcended. In time, however, even the synthesis will prove one-sided in some respect. It will then serve as the thesis for a new dialectical movement, and so the process continues, zigzagging in a spiral.

— **Nonaka and Toyama, 2004**

Agile frameworks didn't emerge as a thesis of knowledge, but as the antithesis of the model software engineering had been developing. That's why, although they are applied to all kinds of work today, their history is tied to IT.

Processes and predictive management

In 1968, during what became known as the "software crisis," NATO held the first conference devoted to analyzing the problems of programming. It made clear the need for a discipline that would bring a systematic, measurable approach to the development, operation, and maintenance of computer systems, and so "software engineering" was born (Naur and Randell, 1969). This first strategy, the thesis, rested on two pillars:

- **Process engineering.** Industrial production had a quality principle with a proven track record: "the quality of the result depends on the quality of the processes used." Put another way, you don't need brilliant people; as long as the processes are good, the result will be good too.
- **Predictive management.** A style of management focused on making sure schedules and budgets are met.

While the discipline refined itself through process models and project management bodies of knowledge (MIL-Q-9858, ISO 9000, ISO 9000-3, ISO 12207, SPICE, SW-CMM...), doubts about this strategy were surfacing in the software industry. From the mid-nineties through to around 2005–2010, entrenched positions were the norm between defenders of process models (thesis) and of agile frameworks (antithesis).

The difference between a bank robber and a CMM theorist is that you can negotiate with the bank robber.

— Orr, 2002

CMM assessment depends more on a good presentation on paper than on the real quality of the software product. It has more to do with blindly following a methodology than with developing and deploying a system in the current technology landscape.

— Orr, 2002

Ask a typical software engineer whether CMM can be applied to agile methods, and you'll get either a look of surprise or hysterical laughter.

— Turner and Jain, 2002

It was far from clear that predictive planning was appropriate for every project. In practice, the criteria for success aren't always about meeting predetermined dates, costs, and features. There was also the question of whether software, like other knowledge-based work, could be produced along industrial process lines.

People began to accept that the tacit knowledge of the person doing the work can contribute more to the value of the result than the technology and processes used. We'll look at this in detail in "Development, work, and knowledge."

The Agile Manifesto

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.

— *Manifesto for Agile Software Development, 2001*

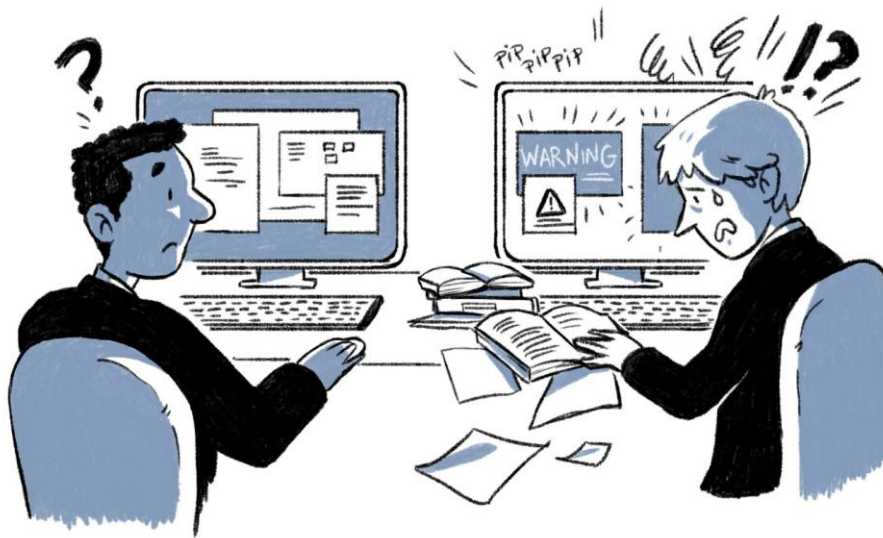
In February 2001, seventeen software professionals met at the Snowbird ski resort in Utah, at the invitation of Robert C. Martin. The meeting had a precedent in the gathering that Kent Beck, author of the book that introduced the *Extreme Programming* methodology (Beck, 1999), had organized in Oregon the previous spring (Highsmith, 2001). They all had something in common: they were critics of process-based production models.

It was at that meeting that the term "agile methods" was coined, to describe those emerging as an alternative to formal methodologies (CMM-SW, the forerunner of CMMI; PMI; SPICE, the forerunner of ISO 33000...), which they saw as heavy and rigid, given their prescriptive character and their reliance on detailed planning ahead of development. The participants boiled down into four statements what we know as the *Agile Manifesto*, the values these methods rest on, and they also established twelve principles.

"Individuals and interactions over processes and tools." The most important of the four. Processes help as a guide for how to operate, and the right tools improve efficiency, but some work requires talent, and it needs motivated people to bring it. In agile development, processes are an aid to guide the work, and the quality of the result depends above all on people's knowledge.



"Working software over comprehensive documentation." The *Manifesto* doesn't treat documentation as useless: only unnecessary documentation. Documents record and communicate relevant information, and legal requirements sometimes make them compulsory, but they should matter less than the product. Looking at prototypes and completed parts brings feedback that generates ideas nobody could have conceived at the outset, whereas a highly detailed requirements document written before starting is usually a waste of time and puts up barriers between departments.



"Customer collaboration over contract negotiation." The goal of an agile project is to continuously deliver as much value as possible to the product, rather than to control that the initial plans are met. With products that evolve continuously, you can't set out in a closed document what the end result will look like; it's more effective to take direct feedback while the product is being built and redefine the requirements for whatever is still to come. For that, the client stays alongside the team throughout the process.



"Responding to change over following a plan." The core values of agile management are anticipation and adaptation, as opposed to the planning and control of orthodox management. With products whose requirements are unstable, where change is inherent, the ability to respond is more valuable than the ability to guarantee a plan is met.



The twelve principles of the *Manifesto*

Alongside the four statements we've just seen, the *Agile Manifesto* sets out twelve principles. They were written with software in mind, but their meaning holds for any knowledge product:

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.

Twenty-five years on, several of these principles have become more relevant than ever. When part of the work is produced by AI, the measure of progress is still the result that works and not the volume produced (principle 7), technical excellence and simplicity (principles 9 and 10) guard against piling up work nobody has reviewed, and the team's regular reflection (principle 12) is the mechanism for adjusting how AI is integrated. We'll come back to them throughout the guide.

Where Scrum came from

Scrum is an agile development model characterized by:

- Autonomous, self-managing teams that share their knowledge openly and learn together.
- An incremental development strategy, rather than planning the product in full up front.
- Basing the quality of the result on people's tacit knowledge and creativity, not on the quality of the processes used.
- Overlapping the different phases of development instead of running them one after another in a sequential, "waterfall" cycle.

The word itself comes from a world far removed from project management: sport. In rugby, "scrum" is the formation in which both teams crouch and bind together, pushing to win the ball without handling it.

For our purposes, we need to travel to 1980s Japan, when researchers Ikujiro Nonaka and Hirotaka Takeuchi gave the term a new dimension. They identified a way of working in the manufacturing companies that were getting the best results in innovation and time to market (Fuji Xerox, Canon, Honda, NEC, Epson, Brother, 3M, and Hewlett-Packard) and compared the way these self-managing teams worked to the way rugby players advance in formation (Takeuchi and Nonaka, 1986).

Although this way of working originated in the manufacture of technology products, from 1995 onwards it began to be applied to software too. That year, Ken Schwaber presented a software development methodology based on a Scrum environment at the OOPSLA conference (*Object-Oriented Programming, Systems, Languages & Applications*), using that very term (Schwaber, 1995). That first framework had phases and artifacts (pregame, game, postgame, planning, sprints, wrap...); some have stuck around, but the rules of the game have changed a great deal since then.

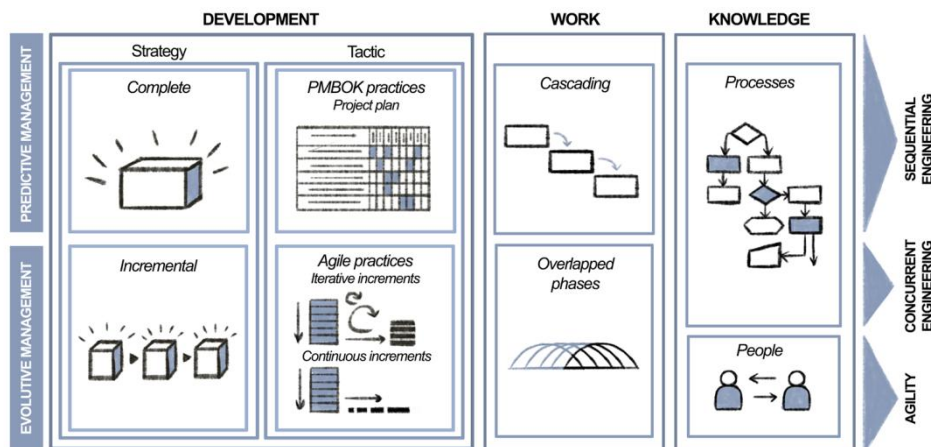
There is no authority that decides what is and isn't Scrum. It has changed and will keep evolving through the contributions of the community of practitioners, who determine which practices prove most useful. Among those added over time are retrospectives, Product Backlog refinement, the Definition of Ready (DoR), and story maps, and the integration of AI is the most recent change in that evolution.

The original spirit holds: practices should help teams manage themselves and maintain a continuous flow of progress, producing results iteratively and often. Scrum Manager[®] uses the term "Scrum" in the original sense that Nonaka and Takeuchi gave it.

Development, work, and knowledge

Since the 1980s, so many models and practices have been developed to improve the quality and efficiency of projects that it can all feel overwhelming. In this chapter we'll look past the labels and summarize the basis of these frameworks: the principles underneath them and the strategies they use to put those principles into practice.

We'll use three concepts (development, work, and knowledge) and two management models we've already mentioned, predictive management and evolutionary management, as our coordinates. These five ideas clear up what looks like a maze of frameworks, because every one of them fits somewhere on this diagram.



Development

Project development can be complete or incremental.

- **With complete development**, a description of what you want to end up with is available at the start; it is full and detailed, and it serves as the basis for estimating, scheduling the work and the resources, and then managing the delivery of what was planned.
- **With incremental development**, the full description isn't available at the start; it's filled in and evolves as development goes along. It can be managed with two tactics:
 - **Continuous incremental development**, using techniques that achieve a continuous flow of features or parts of the product, which are delivered to the client as they are finished.
 - **Iterative development**, using timeboxing techniques that keep product increments coming at a steady rhythm. This is the one used by the standard Scrum framework, which calls each iteration a sprint (→"The sprint"); at the end of each one comes at least one product increment, a deliverable part that is ready to use.

Work

Work can be organized sequentially (in a "waterfall") or concurrently.

- **Sequential work splits the work into phases**, and each phase starts once the previous one is finished. The most familiar example is the waterfall cycle of software engineering: requirements, analysis, design, coding, testing, and implementation.
- **Concurrent work overlaps the different phases in time**, so that all of them are revisited simultaneously and continuously.

Knowledge

Different models locate knowledge either in the processes or in the people.

- **In process-based production**, knowledge is explicit, and the quality of the result lies, for the most part, in the processes and technology used.
- **In people-based production**, knowledge is tacit, and the quality of the result depends on the experience and talent of the people doing the work.

You can see the difference by comparing dinner made with a kitchen robot to dinner cooked freehand. Anyone can produce the first by following the instructions, and the result will be the same regardless of skill. In the second, talent is what counts, and an amateur won't turn out the same dish as a chef. Both benefit from good tools, like a non-stick pan or a well-sharpened knife, but those are only an aid.



Tacit knowledge is personal, context-specific, and therefore hard to formalize and communicate. Explicit or "codified" knowledge, on the other hand, is knowledge that is transmittable in formal, systematic language.

— Nonaka and Takeuchi, 1995

This distinction will be useful later on to understand what AI brings to a team and what it can't bring (→"AI in knowledge work").

Predictive management and evolutionary management

With the three coordinates above, we can describe the two main management models and place Scrum between them.

Sequential engineering

Sequential engineering, also known as predictive management, aims to deliver predictable results. A project is a success if it develops the planned product without exceeding the agreed timeline or resources. Its development is complete, and it uses traditional planning practices. In software, its main reference points are PMI and IPMA and the process models CMMI, ISO 33000, or SPICE, all of them with sequential engineering and process-based production.

Evolutionary management: concurrent engineering and agility

Evolutionary management aims to deliver a minimum viable product as early as possible and then increase its value continuously. It overlaps phases of work and develops incrementally, with short, cyclical iterations or with a constant flow.

It can be carried out with process-based production (concurrent engineering) or with people-based production (agility). The distinction matters, because without it the mere application of the standard rules of Scrum, or the use of kanban visual techniques to keep a continuous flow, gets mistaken for agility.

Concurrent engineering	Agility
Uses resources drawn from agile management: overlapping development phases, cross-functional teams, and frequent improvement iterations.	Reduces or eliminates administrative and bureaucratic tasks that add no value to the product or to the development system.
Focuses on the quality of the processes.	Characteristic of knowledge companies. Focuses on people's tacit knowledge, on culture, and on talent.

Video: [Managing plans and ideas](#) (→"Bibliographic references").

Video: [Purpose of Agile vs. Predictive Management](#) (→"Bibliographic references").

Scrum

To recap, we come back to the characteristics of Scrum that we saw when looking at where it came from, and see how they fit within agile management:

- It uses an incremental development strategy, either iterative with timeboxing or continuous.
- It overlaps the different phases of development.
- It bases the quality of the result on people's tacit knowledge and creativity.
- It works with autonomous, self-managing teams that share their knowledge and learn together. Hence the name, and the metaphor of "advancing as a scrum."

Practices, principles, and values

When you start working with Scrum, as with any other tool, it's a good idea to read the Scrum Master manual and follow the instructions; in other words, to adopt the standard framework. In part one we explain what it consists of, and which roles, artifacts, and events make it up. Think of the framework as a template rather than a mold. **These aren't rules** to be followed to the letter to guarantee speed or quality, but a good **starting point** for beginning to work iteratively.

There's no point kidding ourselves: **if our focus is on the process, on following the rules, and not on people's tacit knowledge and talent, then what we're doing isn't agility but concurrent engineering.** Agility requires trust in the team's capabilities. Tools and processes should help manage the work, not the people, because the end goal is for teams to be able to **manage themselves**.

This warning applies to AI too, the most powerful tool ever to enter a team. Using it to automate compliance with the rules, or to produce more without thinking better, moves the team away from agility instead of closer to it.

Once you've reached an iterative flow of progress, you can go beyond the standard framework. That's the moment to unlearn the practices and lean on Scrum's principles and values to adapt the framework and other techniques to the specifics of the project or the team. In most agile companies these practices are adapted, and the integration of AI obliges all of them to do so.

Part one covers the most widespread Scrum techniques, the standard framework you'll find here and in other manuals. In part two we'll explain how to take the training wheels off the bike, which are very useful at first but can get in the way in the long run, and in part three we'll see what the Scrum Master's craft demands today.



AI in knowledge work

Since 2022, generative artificial intelligence has entered most knowledge work and, since 2024, agents capable of carrying out complete tasks with little human intervention have become widespread. Today AI drafts campaigns and reports, summarizes legal documentation, analyzes data, proposes designs, and writes code. The result is a change in the way value is produced, which affects every team, though not all of them equally.

Let's go back to the kitchen robot from the chapter "Development, work, and knowledge." AI is, to a large extent, explicit knowledge on a scale never seen before: it reproduces and combines, at enormous speed, whatever can be expressed in formal language. What it doesn't bring on its own is tacit knowledge, that is, judgment about what is worth doing, an understanding of the context, and responsibility for the result. That is the part that remains in people's hands.

The bottleneck shifts

Throughout the history of agile management, the narrow point of the work has almost always been execution. Writing the code, drafting the document, or producing the piece took time and effort, and management was organized around that effort to make the most of it, estimate it, and measure it.

When execution becomes cheap, the bottleneck shifts to other activities: **deciding what is worth doing, reviewing what has been produced, and validating that it solves a real problem.** A marketing team can generate twenty variants of a campaign in an afternoon, but someone has to choose which ones fit the brand and check that they don't promise what the product doesn't deliver. A software team can generate the code for a

feature in hours, but someone has to review it, integrate it, and make sure it's the feature that was needed.

AI amplifies the work system

AI is not neutral. It magnifies the conditions that already exist in the team: where there are good practices, frequent delivery, and clear quality criteria, it speeds up the delivery of value; where there are fragile processes and little clarity about what to build, it produces more unvalidated work and makes the problems worse. The 2025 DORA report, with some 5,000 software professionals, put AI use at 90% and concluded that its main effect is that of an amplifier of the strengths and weaknesses the organization already has (DORA, 2025).

That's why the mechanisms that let you see early what's happening and correct course in time gain value with AI instead of losing it, and they are precisely those of Scrum: working in small increments, frequently inspecting both the result and the way of working, and adapting to what is discovered.

Perceived productivity and measured productivity

The feeling of going faster doesn't always match reality. In a 2025 experimental study, a group of experienced developers took 19% longer to complete their tasks with AI than without it, while believing they had been 20% faster (Becker et al., 2025). The authors themselves point out that the result applies to a specific context and to the tools of that moment, but the lesson is general: productivity has to be measured, because perception is misleading.

Nor is it a matter of denying the gain. In an experiment with product innovation teams, one person working with AI reached a quality comparable to that of a pair working without it (Dell'Acqua et al., 2025). AI brings real capacity, and **the question for a team is how to turn that capacity into value and not just into volume.**

The detailed analysis of this change, its effects on agile management, and the criteria for integrating AI into any kind of team has its own syllabus at Scrum Manager: [Agile Bedrock](#).

What changes and what stays

Faced with the impact of AI, the agile community has split. One position holds that Scrum practices have reached the end of the road: cycles of several weeks would be too slow, teams are shrinking, and roles are changing or disappearing. The other holds that AI multiplies agility and that its principles are more necessary than ever, because speed demands more reflection.

Scrum Manager's position takes from each what it gets right. **Agile principles remain fully valid; many practices need a profound transformation**, and each team makes that transformation according to its context, without waiting for someone to prescribe it. In this chapter we preview what stays and what changes, and where we'll deal with it.

What stays

Empiricism stays: transparency about the real state of the work, frequent inspection, and adaptation to what is discovered. **Early and continuous delivery of value** stays, and with it the idea that progress is measured by useful results and not by the volume of what is produced.

Human accountability stays too. AI can propose, draft, or execute, but decisions about the product and responsibility for the result always belong to people. And the **values of the Manifesto**** stay, values that AI makes more critical. Conversation between people is still irreplaceable, and when building the wrong thing is fast and cheap, customer collaboration is the best defense. Responding to change, for its part, still demands moments of reflection that no tool provides on its own.

What changes

The makeup of the team's work changes, as it takes on AI as one more capability under human accountability (→"The Scrum Team and AI"). Quality criteria change, because what AI produces needs explicit review before it can be considered done (→"The Definition of Done"). Some aspects of the events, of how progress is measured, and of the Scrum Master role itself change too, and we'll deal with them in parts one and three.

None of these changes is the same in every team. The effect of AI is greatest in software development, where it can generate and test code, and more gradual in other knowledge work, where it usually assists with drafting, analysis, or synthesis while the person still does the core work. A human resources or legal team, moreover, often works with sensitive information and regulatory requirements that limit what can be delegated.

Where to go deeper

This guide is about Scrum and the Scrum Master's craft. Two Scrum Manager syllabi develop aspects that we only summarize here:

- [Agile Bedrock](#): the foundations of agile management in the age of AI, the integration of intelligent assistance into any kind of team, and the criteria for deciding how to work according to the context.
- [Product Owner](#): the craft of deciding what to build, product discovery, Product Backlog management, and measuring value.

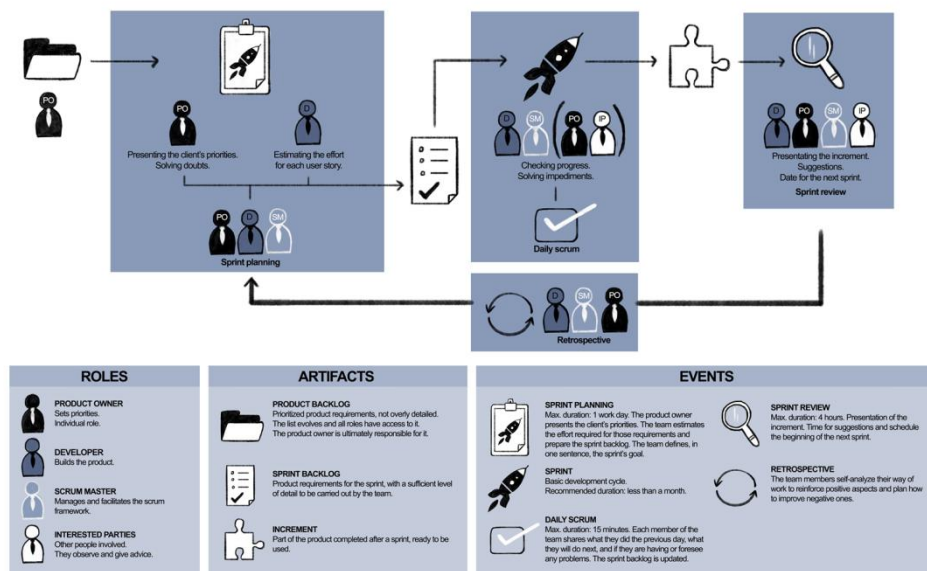
Part I — The Scrum cycle

Learning the standard practices.



The Scrum cycle

In this first part we will look at the standard Scrum framework: its roles, its artifacts and its events, and what AI changes in each of them.



The components of the standard Scrum cycle are:

- The Scrum Team, made up of three roles: Product Owner, developers, and Scrum Master.
- The artifacts: Product Backlog, Sprint Backlog, and increment.
- The events: the sprint, Sprint Planning, the Daily Scrum, the Sprint Review, and the Sprint Retrospective.

You start from an overall vision of the result you want, and from there you specify and flesh out the features you want to have first.

Each development cycle or iteration (sprint) ends with the delivery of one or more working parts of the product (increments). **A sprint lasts no more than four weeks.** Most commonly all sprints are the same length, setting a cadence, but that cadence can **evolve or be adjusted**.

Video: [Scrum](#) (→"Bibliographic references").

Scrum handles the evolution of the project empirically, through the following tactics.

Reviewing the iterations

At the end of every sprint the result is reviewed with everyone involved in the project. The length of the sprint is therefore the longest you can go without discovering approaches that are wrong or could be improved, or misunderstandings about what is being built.

Incremental development

You don't work with designs or abstractions. Incremental development delivers, at the end of each iteration, a working part of the product that can be used, inspected, and evaluated.

Overlapping phases

Design and architecture are refined as you build, rather than being locked down in an early project phase. The phases that waterfall development runs through one after another overlap in Scrum and move forward at the same time.

Self-management

Predictive management makes a project manager responsible for running the project. In Scrum, teams are self-managing, with enough decision-making scope to reach whatever resolutions they see fit. That speeds up decisions and makes it possible to respond quickly when the unexpected happens. With AI, that decision-making scope also covers how AI is integrated into the team's work.

Collaboration

Every team member collaborates openly with the rest, according to what they can do rather than their role or job title. Between self-management and collaboration, the team handles perfectly well the work a project manager would do in other models.

Empiricism

Scrum is based on **empiricism**, the idea that knowledge comes from experience and that decisions are made on the basis of what is observed. Instead of relying on a detailed plan drawn up at the start, the team moves forward in short cycles, looks at the result, and adjusts course. This way of working rests on three pillars.

Transparency

The real state of the work, the product, and the process must be visible to those who make decisions about them. Without transparency there is no reliable inspection, because what gets inspected is a distorted picture of reality. Scrum's artifacts exist, to a large extent, to make the work visible: what is left to do, what is being done, and what is already finished.

With AI, transparency takes on a new aspect: the **provenance** of the work. The team and the stakeholders must be able to know which parts AI has produced, who has reviewed them, and by what criteria. A report that looks finished but that no one has checked is an example of a lack of transparency, even if it is in plain sight of everyone.

Inspection

The team frequently examines the product and the way it works in order to detect deviations and problems. Inspection happens in the Scrum events, and its value depends on its being frequent and on its being done on reliable information.

AI multiplies the information available and can help analyze it: summarizing progress, detecting patterns, or flagging risks. But generating data is not the same as inspecting. Inspection requires someone to stop, look with judgment, and draw conclusions, and the speed of production tends to push the team to skip that moment.

Adaptation

When inspection reveals that something is drifting beyond what is acceptable, the team adjusts the product or its way of working as soon as possible. Adaptation is what turns learning into results, and it requires the team to have the autonomy to decide on the changes.

The three pillars reinforce one another and are, as we saw in the introduction, what AI makes more necessary. The faster things are produced, the more important it is to see early what is happening and to correct course in time.

The committed and the involved

Plenty of people take part in a project and add value to it, with different levels of commitment and responsibility. To tell them apart, a distinction is usually drawn between "the committed" and "the involved."

- **The committed:** they take part directly in building the product or delivering the service and answer for its result. They are the members of the Scrum Team.
- **The involved:** other stakeholders who contribute information, resources, or decisions within their own area, but who do not answer for the result of the sprint. For example, management, sales, marketing, the people who operate the system being built, or user support.

The distinction is not a hierarchy of importance. The involved are indispensable, because without their information the team would not know what to build or whether what it has built is any use. What the distinction makes clear is who takes on the commitment to the Sprint Goal and who decides how the work is done. The involved can attend some events and contribute their view, but they do not direct the team's work.

In practice, the boundary is not always clear-cut, and sometimes one of the involved joins the team for a while, or a team member moves to another project. What matters is that everyone knows which position they are in at any given moment, because their responsibilities depend on it.

The distinction also helps place AI, which takes part in building the product but cannot commit to or answer for a result. We will look at this next.

The Scrum Team and AI

AI already takes part in the work of many teams: it drafts, summarizes, analyzes, proposes and, in some cases, carries out entire tasks. This raises a practical question for the framework: where AI sits within the Scrum Team.

This guide's answer is that **AI is a capability of the team, under the accountability of the people who make it up**. It extends what the team can do, as any powerful tool does, but it does not hold a role or take on responsibilities. The Product Owner, the Scrum Master, and the developers are always people, and they answer for the result of the work, whoever or whatever produced it.

Accountability cannot be delegated

AI can propose, draft, or execute, but it cannot be held accountable. When an agent makes a mistake, accountability does not dissolve into "the AI did it": it must be clear which person answers for what AI has produced. In Scrum that accountability is shared by the team, which accepts work as finished only when it meets the criteria the team itself has agreed on (→"The Definition of Done").

This idea, that AI proposes and the person decides and answers for the decision, runs through the whole guide. It is also what makes it possible to use AI with confidence, because confidence rests on knowing who has reviewed what.

Four explicit decisions

Integrating AI is a self-management decision of the team, not a tool setting. Whatever use the team makes of it, every team must explicitly agree on four questions:

1. **What AI does:** which tasks are delegated to it and which are reserved for people.
2. **Who reviews:** which person reviews what it produces, and by what criteria.
3. **How it is integrated:** at what point in the workflow it comes in and how its output is incorporated.
4. **What limits it has:** what it cannot do, what information it cannot access, and when it must be stopped.

A legal advisory team, for example, may decide that AI prepares the first draft of a case-law summary, that the person responsible for the matter always reviews it before it is sent, that it is used only in the research phase, and that it never receives clients' personal data. When these decisions are left unmade, tasks appear that no one supervises and that become risk zones.

The agreements are not made once and for all. The team reviews them in the retrospective as it learns what works (→"The Sprint Retrospective"), and the Scrum Master helps keep them alive (→"AI working agreements for the team").

AI as a team member

Emerging trend (2026). This section describes an open debate in the profession. It is not covered by the certification exam.

Part of the community proposes treating AI agents as team members with their own assigned work, provided that at least one person answers for their result. Some teams already assign them Sprint Backlog items and apply to them the same Definition of Done as to the rest of the work.

Other voices consider the analogy imperfect, because AI does not commit, does not learn from the team the way a person does, and does not genuinely take part in reflecting on the way of working. From this perspective, it is a very powerful tool in the service of the committed.

However the debate ends, the practical consequences are the same: the four explicit decisions and human accountability for the result.

The Product Owner

The Scrum Team has three roles: the Product Owner, the developers, and the Scrum Master. We start with the first.

The Product Owner is the person who makes the client's decisions about the product, and their responsibility is to **maximize the value** of what the team builds. To keep communication and decision-making simple, this role sits with a single person. If the client is a large organization, or one with several departments, it can handle its internal communication however it sees fit, but only one person joins the team: someone who represents the client and has the knowledge and the authority needed to make decisions.

On in-house developments, the role is usually taken on by the product manager or the head of the area that needs the product. On developments for external clients, it falls to whoever leads the client's procurement process. Depending on the circumstances, the Product Owner may delegate tasks to the team or to someone they trust, but the responsibilities remain theirs:

- Developing and administering the Product Backlog.
- Presenting the product vision and the value expected from each sprint, and taking part in Sprint Planning for every sprint (→"Sprint Planning").

The Product Owner owns the Product Backlog. They have the final say on what the product will look like and the order in which increments are built: what goes in, what comes out, and what priority each item has. They know the product plan, what it's capable of, the investment, and the expected return, and they share that vision with the team.

The Product Owner has expert knowledge of the client's business. They analyze the environment continuously (how the market is moving, the competition, the alternatives) and combine that information with what comes out of the team during development: suggestions, technical alternatives, and the evaluation of each increment. They need to know Scrum and, ideally, to have worked with the same developers before. The organization respects their decisions and does not change the priorities or the items of the Product Backlog.

Related link: [The 5 keys to a good Product Owner, Scrum Manager Podcast \(in Spanish\)](#) (→"Bibliographic references").

The Product Owner and AI

When building gets cheaper, the bottleneck of the work shifts to deciding what deserves to be built, and that is the Product Owner's territory. The role gains importance with AI, provided it evolves from the tactical to the strategic: a Product Owner who limits themselves to writing and ordering backlog items is doing exactly the part of the work that AI can assist with most.

Their contribution is concentrated in keeping the vision, in designing how what is built will be validated, and in deciding where AI adds value and where it only generates volume. If the team can produce ten campaign proposals in a sprint, the Product Owner needs a way of knowing which ones work, or the team will produce a lot without learning anything. The Product Owner's craft has its own syllabus at Scrum Manager: [Product Owner](#).

The Developers

The developers are the professionals who build the increment in every sprint. Although Scrum calls them that, they are not always programmers. The term covers everyone whose work contributes directly to building the increment: writers, designers, analysts, technicians, or specialists in any field, depending on the product.

There is no fixed rule for the size of the Scrum Team. Counting the Product Owner, the Scrum Master, and the developers, and depending on the circumstances of the organization, the project, and the team's use of AI, a team of three or four people may be appropriate, or a larger one, **up to a maximum of ten**. Beyond that, direct communication becomes hard to sustain, and the friction of group dynamics becomes more pronounced.

Hackman and Vidmar (1970) put groups of between two and seven members through production, discussion, and problem-solving tasks, and found that dissatisfaction with the size of the group rose in roughly linear fashion as it grew, with an optimum at around 4.6 members.

They are cross-functional professionals who work supportively, with shared responsibility. Some may specialize in a particular area, but responsibility for the result rests equally with all of them. What sets a "team" apart from a "working group" is precisely that shared responsibility:

- In a **working group**, each person has their own assigned responsibilities, follows a process or set of execution guidelines, and answers individually.
- In a **team**, people share a common purpose, getting the most possible value out of the client's vision, and manage themselves so that they work and answer for the results together. There is no manager to assign and coordinate the work; the developers coordinate it themselves, work with the Product Owner on the Product Backlog, and share the goal of each sprint and the responsibility for reaching it.

The developers and AI

With AI, the value of the developers' work shifts in part from executing to **specifying, orchestrating, and reviewing**. Knowing what to ask AI for, with what context, how to evaluate what it produces, when to reject it, and how to integrate it becomes a core competence of the team.

This does not make knowledge of the craft dispensable; on the contrary, evaluating work produced by AI takes more judgment than producing it yourself. A financial report generated in minutes may contain a calculation error that only someone who knows the subject well will spot, and the same goes for a contract, a design, or a piece of code. That is why people's experience remains the team's main guarantee of quality, and why it is worth making sure that the less experienced profiles acquire it (→ "Learning and developing people").

Teams that work with AI tend to be smaller, because each person can cover more work. A small team coordinates with less effort, but it has less diversity of perspectives and depends more on each person. The right size is a decision for the context, within the maximum of ten people for the whole Scrum Team.

The Scrum Master

The Scrum Master is accountable for the **effectiveness of the Scrum Team**. They achieve it by helping the team and the organization understand and apply Scrum, and by creating the conditions for the team to continuously improve its way of working. The aim is for the team and the client to become capable of organizing themselves and working autonomously.

It is a service role, exercised in three directions:

- **To the team:** helping it self-manage, focus on creating increments of value, and resolve the impediments that hold back its progress; handling group-dynamics difficulties and making sure the events take place and are useful.
- **To the Product Owner:** helping them manage the Product Backlog, express goals clearly, and work with the stakeholders.
- **To the organization:** training and advising on Scrum, helping plan its adoption, and removing the obstacles between the team and the rest of the organization.

The Scrum Master does not direct the team's work or assign tasks. Nor do they run the Daily Scrum, which is a meeting of the developers (→ "The Daily Scrum"): their role is to facilitate when needed and then deal with any impediments that have come up.

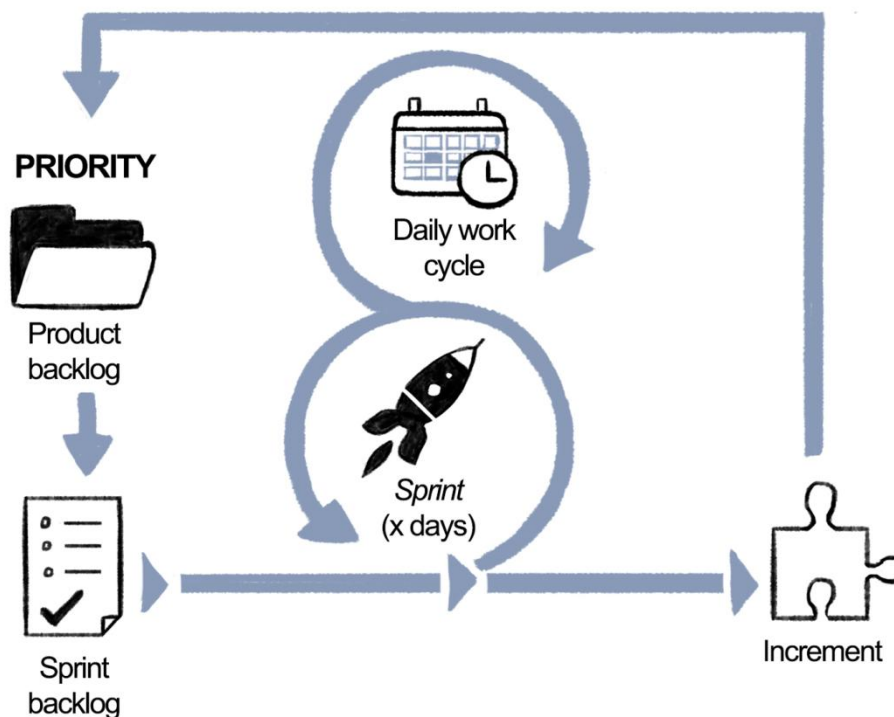
A role that grows with AI

The integration of AI broadens the Scrum Master's scope of responsibility. Beyond facilitating work between people, they now help the team design how work is shared with AI, agree on its limits, and rigorously review what it produces. They also protect the moments of reflection from the pressure of speed and look after people through a change that generates uncertainty.

Some operational tasks of the role, such as preparing agendas, taking notes, or analyzing metrics, can be supported by AI. What AI cannot do is read the state of a team, create psychological safety, or accompany a change in the way of working, and that is where the heart of the craft lies. We will develop this in the third part (→"A role that evolves").

Scrum artifacts

Scrum's artifacts are its tools, its basic building blocks. They support the roles during the events and, above all, they make the work **transparent**: what is to be achieved, what is being done, and what is already finished.



The most widely used artifacts

The standard framework rests on three key artifacts, each tied to a commitment that gives it meaning:

- **Product Backlog:** the ordered list of everything the product may need. At first it reflects the minimum viable product (MVP) or the initial vision, and it grows and evolves during development. Its commitment is the **product vision**, the result the team wants to reach.
- **Sprint Backlog:** the work the developers are going to do during the sprint. Its commitment is the **Sprint Goal**, the value the sprint must deliver.
- **Increment:** the result of the sprint, a part of the product that works, tested and ready to use. Its commitment is the **Definition of Done**, the criteria it must meet to be considered finished.

Related link: [How to build a backlog, Scrum Manager Podcast \(in Spanish\)](#) (→ "Bibliographic references").

Other artifacts

Alongside these three, many teams use other artifacts that we will see later on:

- **Burndown chart:** shows the work remaining in the sprint and how it evolves (→ "Measuring progress").
- **Burnup chart:** shows how much of the product has been built and the forecast for what is left (→ "Measuring progress").
- **Definition of Ready (DoR):** an agreement on when a Product Backlog item is ready to be pulled into a sprint (→ "The Definition of Ready").
- **Definition of Done (DoD):** an agreement on the criteria for considering a piece of work finished (→ "The Definition of Done").

In teams that work with AI, all these artifacts serve an added purpose. They are where the team records which work has been delegated to AI and who has reviewed it, and where stakeholders can check it.

Related link: [Definition of Done, Scrum Manager Podcast \(in Spanish\)](#) (→ "Bibliographic references").

Work items

A "work item" is what the team uses to manage the work within a sprint. The most commonly used ones are user stories, technical stories, and tasks.

- **User story:** a short, simple, specific description of the value the product should deliver to the end user, written from that user's point of view. Stories come out of conversations between the developers and the Product Owner aimed at aligning the vision, and they serve as a reminder of what was discussed.

- **Technical story:** similar to a user story, but for work the end user doesn't see directly, such as reorganizing an existing system or implementing security measures.
- **Task:** a smaller unit, an activity that has to be completed in order to deliver a story, such as writing a text, designing a screen, or running tests.

When a story is too big and too loosely defined, it's usually called an "epic." Epics are agile requirements, but their size and lack of detail make them unusable as work items. Building a new ticket management system, or launching the recruitment campaign for a new course, are examples of epics.

Related link: [How to write a user story, Scrum Manager Podcast \(in Spanish\)](#) (→ "Bibliographic references").

Related link: [5 characteristics of a good user story, Scrum Manager Podcast \(in Spanish\)](#) (→ "Bibliographic references").

When a team starts working with Scrum, it's common to break stories down into tasks, because that makes day-to-day progress visible and keeps less experienced teams from getting stuck. As experience grows, breaking every story down can become tedious; if the team manages its work well directly at story level, forcing a breakdown is unnecessary micromanagement. Which work items to use depends on the context.

Specification by example

User stories were born to communicate with people, who can ask questions, infer the context, and propose alternatives. AI, on the other hand, interprets what it is given literally and does not fill in missing context. When part of the work is going to be produced by AI, the team needs to state more precisely what is expected.

The most useful way to do this is to **specify with examples**: to accompany the story with concrete cases that show the expected result and that serve to check whether it has been achieved. For the story "as a candidate, I want to receive a reply to my application so that I know whether I'm still in the process," a human resources team might add examples such as "if the application does not meet the degree requirement, the reply says so and offers the area's other open positions" or "the reply never mentions data about other candidates."

Examples turn intent into verifiable criteria, both for whoever reviews and for the AI that produces. The level of detail depends on the risk: a quick test to learn something can start from the story alone, while anything that will reach real users needs precise criteria. Specification complements the conversation, but it does not replace it.

The Product Backlog

The Product Backlog is the inventory of everything that is to be built into the product over successive sprints: features, improvements, technical changes, and bug fixes. It represents what the client, the users, and the other stakeholders are expecting, and anything that means work for the team should be reflected in it. A few examples of its items:

- "Let users browse the files published by a given member of the platform."
- "Look up the orders placed by a seller within a date range."
- "Prepare the onboarding guide for people joining the department."

Its essential characteristic is that it **holds living information**, continuously evolving. Rather than a requirements document, it is a tool that makes communication with the team easier. At the start of the project it contains only a handful of items, the ones best understood at that point, and it expands and changes as development progresses. It only needs enough items for a first sprint.

The Product Owner keeps it ordered according to the value and urgency of each item. How detailed an item is should be proportional to its priority: the highest-priority ones need to be detailed enough to be pulled into the sprint, and the rest can wait.

Refinement

Ordering, detailing, and pre-estimating items is called **refinement** or **preparation**. It's collaborative work between the Product Owner and the developers, who can meet for it at any point, as long as it never takes up more than 10% of the team's capacity. Estimates are the developers' call, since they're the ones who will do the work (→ "Estimating").

The best way to maintain the Product Backlog is with simple means that the whole team knows and shares: sticky notes on a board, ordered by priority, or a management tool everyone is comfortable with. It's an information radiator and a tool that supports direct communication.

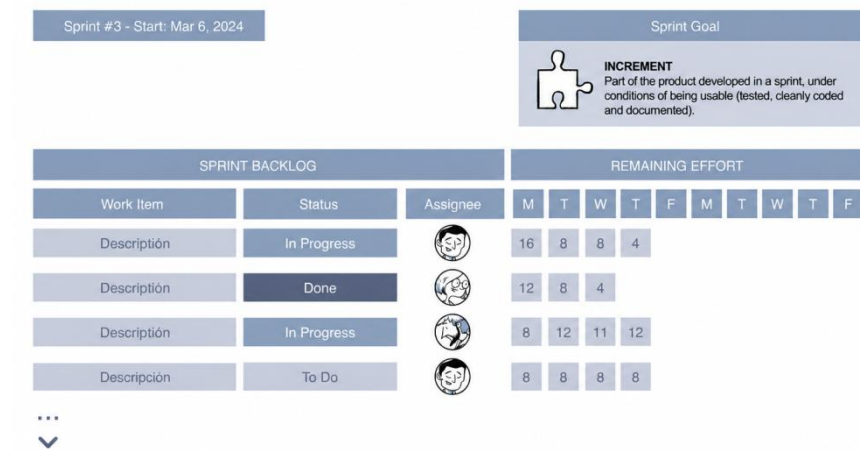
The Product Backlog and AI

When building is fast and cheap, the risk of filling the product with things nobody needs goes up. That's why many teams treat each Product Backlog item as a **hypothesis**: a bet on what problem it solves and for whom, which has to be validated. Every item should be able to answer a simple question: what problem does it solve, and how will we know whether it has solved it?

AI can help draft, group, or spot duplicate items, but the decision on what goes in and in what order still belongs to the Product Owner. The methods for discovering and validating what deserves to be built are developed in Scrum Manager's [Product Owner syllabus](#).

The Sprint Backlog

The Sprint Backlog is the list of all the work items needed to reach the Sprint Goal. In it, stories are broken down, if the team finds it useful, into pieces of the right size to track progress daily and to spot risks and problems without any complicated management process.



The whole team helps put it together during Sprint Planning. If the developers split stories into tasks, a task shouldn't take more than a day's work. If the Product Backlog is the Product Owner's territory, **the Sprint Backlog is the developers' territory**: they're the only ones who can change it during the sprint.

It provides direct visual communication, and the developers use it to review progress every day. Ideally it is in plain sight of everyone, on a physical board or in a collaborative tool, in whatever format suits the team best, with this information:

- The Sprint Goal.
- The list of stories or tasks.
- Who has taken on each work item.
- What state each one is in and how much is left to complete it.

The Sprint Backlog should be aligned with the Sprint Goal, which marks a milestone on the way to the product vision.

The Sprint Backlog and AI

When part of the work is produced by AI, the Sprint Backlog makes visible how it is shared out. Many teams note in each work item which parts AI will produce and which person will review them, so that attribution becomes part of the sprint's transparency.

Above all, the Sprint Backlog reflects that **reviewing is work too**. AI can produce in minutes what will take a person hours to review and integrate, and if the sprint plan only counts what is produced, review becomes an invisible bottleneck. Review and

integration tasks are recorded and planned like any other (→"Planning with AI on the team").

The increment

The increment is the part of the product built in a sprint that is fit to be handed over to the client: finished, tested, and working. **One or more increments** may be produced in a sprint, and all of them must meet the Definition of Done.

Prototypes, loose modules, and anything still awaiting testing don't count as increments. Ideally, every Product Backlog item refers to something deliverable, not to internal work of the "design the database" or "write the table of contents" variety.

The first sprint can be an exception. It's often called "sprint zero" when it has goals such as testing out the platform, the design, or the tools and working methods being considered, which some projects need at the outset.

If the part that's been built requires documentation, or documented validation and verification processes, those also have to be complete before the increment can be considered done, that is, ready to hand over to the client.

A prototype is not an increment

AI makes it possible to produce prototypes in very little time, and that is one of its great contributions: testing an idea with users now costs a fraction of what it used to. But a prototype is for learning and is not ready to be delivered. It lacks the reliability, security, regulatory compliance, and maintainability that don't show up in a demo.

Prototypes are useful, but they are presented for what they are. When an idea validated with a prototype becomes part of the product, it is built or reviewed against the full criteria of the Definition of Done.

The Definition of Done

The **Definition of Done (DoD)** is the team's agreement on the criteria a piece of work must meet to be considered finished. It is the increment's commitment: an increment is not finished until it meets all of its criteria, and whatever doesn't meet them goes back to the Product Backlog. It must be known and shared by the whole team and by the stakeholders.

The DoD provides transparency, because everyone knows what "finished" means without having to ask. It also provides quality, because the criteria are not negotiated sprint by sprint. The team sets them, based on the organization's requirements, and revises them as it matures. For a software team it may include the code being reviewed, tested, and integrated; for a communications team, the text being reviewed by a second person, its sources verified, and approved by the legal department where applicable.

The Definition of Done and AI

With AI, the Definition of Done is the main counterweight to speed. AI produces work that looks finished but may contain subtle errors, made-up data, or solutions that work today and cause problems tomorrow. Without explicit criteria, that work piles up without anyone noticing.

That's why teams that work with AI make their DoD more demanding, not less. The central criterion is **human review**: all work produced by AI has been reviewed by a person who understands what it does, not just that it works or looks right. When review draws on the reviewer's experience, the DoD also protects against the bias of accepting whatever a machine produces.

The reinforced Definition of Done

Consolidating trend (2026). This section describes practices that are gaining adoption quickly but are not yet widespread. It is not covered by the certification exam.

Many teams add to their DoD specific criteria for work produced by AI, known as a **reinforced DoD**. The most common are:

- **Independent verification**: what AI produces is checked by someone or something other than whatever produced it, because AI tends to approve its own work.
- **Visible provenance**: a record is kept of which parts AI produced and who reviewed them.
- **Automated checks**: in software, automated quality and security analysis; in other teams, checks on sources, on personal data, or on consistency with the style guide.

Some teams use a second AI as a first filter before human review, which it never replaces.

Done and outcome achieved

Emerging trend (2026). This section describes a recent proposal in the profession. It is not covered by the certification exam.

Some authors propose distinguishing between work that is **done**, which meets the DoD, and the **outcome achieved**, when there is evidence that the work has produced the benefit that was sought. The proposal responds to a risk that AI makes worse: finishing a great deal of work without knowing whether it serves any purpose.

The Definition of Ready

The **Definition of Ready (DoR)** is the team's agreement on when a Product Backlog item is ready to be pulled into a sprint. A ready item is defined and pre-estimated, can be completed within a single sprint, and has established criteria for considering it finished, along with who will verify them.

The DoR keeps items that the team doesn't understand or can't finish from entering the sprint, and keeps the sprint from being spent clarifying what should have been clarified beforehand. It applies above all to the highest-priority Product Backlog items, during refinement.

It's best not to turn it into bureaucracy. A DoR with too many requirements slows down the entry of valuable work and turns preparation into a formality. The aim is for the team and the Product Owner to agree that the item is ready, not to fill in a checklist.

Preparing the work AI will do

Consolidating trend (2026). This section describes practices that are gaining adoption quickly but are not yet widespread. It is not covered by the certification exam.

When part of the work is going to be delegated to AI, many teams extend their DoR with specific criteria:

- AI has access to the **context** it needs: the documentation, the style criteria, the data, and the background of the work.
- The acceptance criteria are verifiable, ideally with examples (→"Work items").
- It has been decided who will review the result and how long the review will take.

It's the equivalent of preparing well for a new person joining the team. If AI doesn't have enough context, its output will be irrelevant or wrong and will create rework.

Scrum events

Events make up the working routine in Scrum. Each one is a formal opportunity to inspect and adapt, and each has a maximum duration (a timebox) that keeps it from running on unnecessarily. Holding them regularly creates a cadence that reduces the need for other meetings.

- **Sprint:** the container for all the other events and the period in which the increment is built (→"The sprint").
- **Sprint Planning:** marks the start of the sprint and determines its goal and the work needed to achieve it.
- **Daily Scrum:** a short meeting in which the developers review progress toward the goal and adjust their plan for the day.
- **Sprint Review:** the inspection of the increment with stakeholders, and adaptation of the Product Backlog if necessary.
- **Sprint Retrospective:** the team's reflection on the way it works, and the plan for improvements to apply in the next iteration.

One event that isn't part of the traditional framework but has become widespread is the **backlog refinement session**, which we already saw when discussing the Product Backlog. There's no consensus on when to hold it; it often takes place before Sprint Planning, so that the team arrives there with items already detailed and estimated.

The maximum durations we give for the events correspond to a four-week sprint. In shorter sprints, the events are usually proportionally shorter.

With AI, the events don't lose their purpose; their focus shifts. In each one the team decides something about AI: at Sprint Planning, what to delegate to it; at the Daily Scrum, how the work it has produced is going; at the Sprint Review, whether what has been built is the right thing; and at the Sprint Retrospective, how the collaboration with it is working.

Related link: [Scrum events, Scrum Manager Podcast \(in Spanish\)](#) (→"Bibliographic references").

The sprint

The sprint is the key Scrum event for keeping progress continuous: a fixed period of time, **no longer than four weeks**, during which one or more product increments are built. All of a team's sprints usually have the same length, so that they set a cadence. Each sprint has a **Sprint Goal**, the value it must deliver, which guides the team's decisions while it runs.

When you're starting out with Scrum, it's worth treating the sprint as the container for all the other events:

- It sets the daily rhythm of progress, which is made visible and shared at the Daily Scrum.
- It sets a fixed rhythm for checking how the product is developing, at Sprint Planning and Sprint Review.
- Retrospectives are held at that same rhythm, for reflection and improvement.

In more mature implementations, the sprint can be treated as covering only the construction of the increment, leaving the meetings outside it. That gives more flexibility, for instance, to run sprints of different lengths or to decouple the frequency of retrospectives from the frequency of sprints.

Sprint length

Choosing the length is a balancing act. A long sprint gives time to build larger increments, but it delays the moment of inspection and increases the risk of spending weeks building in the wrong direction. A short sprint offers more frequent feedback, but it increases the relative weight of the events.

With AI, many teams feel the sprint has become too long for them, because they can produce increments in days. The useful question isn't how much the team can produce, but how often it needs to inspect and adapt. The availability of stakeholders to review the work, and the team's own capacity to review what it produces rigorously, still limit the pace, however much AI speeds up production.

Cadences in tension

Consolidating trend (2026). This section describes practices that are gaining adoption quickly but are not yet widespread. It is not covered by the certification exam.

Teams working with AI are exploring several cadence options:

- **Micro-iterations within the sprint:** the sprint remains the container for planning and review, with short cycles of production and validation inside it.
- **Shorter sprints**, of a week or less, with more feedback but with a greater relative weight of the events.

- **Continuous flow** with kanban techniques, without fixed-length sprints, at the risk of losing the structured moments of reflection.

None of them is best in every case. What matters is that the cadence is a conscious decision of the team, reviewed at the retrospective, and that, whatever it is, it keeps the moments of inspection and adaptation.

Sprint Planning

Sprint Planning marks the start of every sprint. Taking the client's priorities and business needs as a basis, it determines the Sprint Goal and the work to be done. The Product Owner and the developers attend, and other people the team invites to contribute information may take part. Its length is proportional to that of the sprint, and it lasts no more than eight hours for a four-week sprint.

Preconditions	Inputs
The highest-priority Product Backlog items are ready, detailed enough and with a preliminary estimate. The developers know enough about the technologies and the business to make estimates and to understand what the Product Owner presents.	Product Backlog. Previous increments (except in sprint zero). The team's capacity for the sprint and its recent throughput. The client's business circumstances and the value the Product Owner expects to obtain.

The meeting answers three questions.

Why is this sprint valuable?

The Product Owner sets out how the product's value can be increased by the outcome of the sprint, presents the highest-priority items, and explains the relevant changes to the Product Backlog since the last meeting. The presentation is open to questions, suggestions, and alternatives.

The team then defines the **Sprint Goal**, a single sentence capturing the value that's going to be delivered. Working it out together guarantees that the whole team understands and shares the purpose of the work, and during the sprint it serves as a reference point for decisions.

What can be done in this sprint?

The developers select the Product Backlog items they can complete, taking into account their capacity and recent throughput. Items that need more detail can be refined during the selection. This meeting is one of Scrum's hot spots for cross-pollinating ideas.

How will the increment be achieved?

The developers decide how they'll turn the selected items into a done increment, and they assemble the Sprint Backlog. They can break the items down into tasks of a day or less, which is recommended for less experienced teams because it helps make progress visible and builds a better instinct for what each kind of work involves. The Product Owner fields questions and checks that the goal is understood and shared.

What the Scrum Master does during Sprint Planning

The Scrum Master helps make the meeting useful and keep it within its timebox. In particular, they make sure that:

1. There's a Product Backlog prepared by the Product Owner.
2. The dialogue between the Product Owner and the developers keeps flowing, and they reach agreement on what the sprint will include.
3. The work is broken down and estimated realistically, including any analysis, research, review, or support tasks.
4. By the end of the meeting, the items to be worked on, the Sprint Goal, and the initial Sprint Backlog are clear.

Planning with AI on the team

In a team that works with AI, Sprint Planning includes one more decision: how the sprint's work is shared out between people and AI. It isn't a technical question but a matter of self-management, and it applies the four explicit decisions we saw in "The Scrum Team and AI" to each specific sprint.

The Sprint Goal gains weight

When AI makes it possible to produce almost anything quickly, the risk of losing focus grows. A clear Sprint Goal is the best defense, because it helps decide what won't be done even though it could be. A marketing team that can generate fifty pieces in a week needs more than ever to know what the campaign's purpose is and which pieces serve it.

Review is planned

The part of the work most often underestimated is reviewing and integrating what AI produces. AI can generate in minutes what will take a person hours to check, and if the sprint plan only counts what gets produced, the team will discover halfway through the sprint that it has no time to review the output. That's why review capacity is as important an input to planning as production capacity, and review tasks are recorded in the Sprint Backlog.

Human-AI attribution

Consolidating trend (2026). This section describes practices that are gaining adoption quickly but are not yet widespread. It is not covered by the certification exam.

Many teams make the **attribution** of each work item explicit at Sprint Planning: what requires human judgment or creativity, what AI can produce under review, and which person will be accountable for each part. For the tasks delegated to AI, they also check that the specific readiness criteria are met (→"The Definition of Ready"). Attribution isn't final: if during the sprint it turns out that AI isn't producing what was expected, the team reassigns the work.

The Daily Scrum

In Scrum, the developers monitor how each sprint is going through **very short daily meetings**, of **15 minutes at most**. They're also known as "stand-up meetings" or, colloquially, "dailies." They can be held in person or remotely, always with access to the board or tool that shows the sprint's progress. In them, the developers sync up their work and adjust the plan for the next 24 hours.

Inputs	Outputs
Sprint Backlog and burndown chart or kanban board with the information from the previous meeting. Information on each developer's progress.	Updated Sprint Backlog and burndown chart or board. Impediments identified.

Meeting format

The developers run this meeting. It isn't a check-up: it's a space for the team to collaborate, share where the work stands, and spot difficulties. Other people may attend, though they don't take part. The Scrum Master's role is to facilitate if needed and to take care of clearing impediments after the meeting.



A common format when starting out is for each developer to explain what they've achieved since the last meeting, what they'll be doing before the next one, and whether they have or foresee any impediment. This individual-by-individual format can have negative effects on motivation and team dynamics, and it's worth **evolving toward a format that focuses on the work, not on the people**.

In that format, the developers look at the state of the work, item by item. Seeing what's in flight, they take on the outstanding work in priority order and make sure no bottlenecks form. This shortens the meeting, increases the team's autonomy, and improves collaboration; you see it more often, for example, that several people decide to work together on the top-priority item that's at risk of falling behind.

The Daily Scrum with AI

When part of the work is produced by AI, the Daily Scrum takes on one more question: whether what AI has produced since the last meeting is what was expected, and who is reviewing it. Deviations, produced work waiting for review, and blockages caused by missing context are now impediments as relevant as the usual ones.

Asynchronous formats

Consolidating trend (2026). This section describes practices that are gaining adoption quickly but are not yet widespread. It is not covered by the certification exam.

In very small teams that work together continuously, a formal meeting can feel out of proportion. Some replace part of their Daily Scrums with brief updates in a shared channel, and meet only when deviations or blockages come up. The risk is losing the

conversation that catches problems before anyone writes them down, so it's worth checking at the retrospective whether the format is still doing its job.

The Sprint Review

The Sprint Review is held at the end of the sprint to inspect the increment and gather suggestions. The whole Scrum Team attends, along with any stakeholders who want to. It lasts no more than four hours for a four-week sprint.

At regular intervals, this meeting sets the pace of construction and shows the direction the product is taking, giving stakeholders a chance to suggest how it should keep evolving. The developers present what they planned, what they've finished, what they haven't, and the impediments they ran into, so that expectations are realistic from the start.

Inputs	Outputs
A done increment.	Feedback for the Product Owner. Input for improving the product's value. Revised Product Backlog.

Meeting format

This is an informal working meeting in which the increment is shown working and discussed. Depending on the product, it may also include user or technical documentation. A recommended running order is this:

1. The developers set out the Sprint Goal, the items it included, and the ones that are done.
2. They demonstrate what they've built in action, and the floor is opened for questions and suggestions, which generates valuable input for the Product Owner and the team.
3. With what they've learned, the team and the stakeholders discuss what's best to do next, and the Product Owner adapts the Product Backlog.

The Sprint Review with AI

With AI, the central question of the review shifts from whether what's been built is well built to whether **the right thing is being built**. When producing is cheap, the main risk is delivering a lot that's of no use, and the review is the moment to check that with the people who will use it. That's why it's worth bringing into the review data on how previous increments are being used, rather than only showing the new ones.

It's also worth avoiding overproduction. Presenting more than stakeholders can evaluate carefully overloads the meeting and impoverishes the feedback. And

transparency about provenance applies here as in the rest of the work: the team explains which parts AI produced and how they were reviewed.

The Sprint Retrospective

The retrospective is held after each Sprint Review and before the next sprint's planning. It lasts no more than three hours for a four-week sprint. It's an example of a practice that wasn't in the original Scrum framework but has become established over time. In it, the Scrum Team reflects on the way it works, identifies strengths and weak points, and plans improvement actions.

Because it takes place at the end of the sprint, it's sometimes confused with the Sprint Review, but it's better to handle the two separately because their goals are different. The review analyzes **what** is being built, the product; the retrospective focuses on **how** the team works.

The whole Scrum Team takes part, including the Product Owner, who should see themselves as "team" rather than "client." Their perspective adds a great deal of value, since they're the one who keeps the focus on building what the client needs. If they don't take part actively, the Scrum Master acts as facilitator to secure their involvement.

More necessary than ever

With AI, the retrospective is the event most at risk of being dropped and the one most needed. It's the natural space for assessing how the collaboration with AI is working:

- whether the agreements on what gets delegated and who reviews are still appropriate;
- whether review is rigorous enough or unreviewed work is piling up;
- whether the Definition of Done needs adapting;
- whether the greater output is turning into more value.

There's a pattern that repeats in teams for which AI doesn't work out. They adopt the tool, output goes up, and they drop events to go faster. From then on they lose the ability to spot problems, which pile up until an incident makes them visible. The decisive step is the third one, dropping the moments of reflection, and the teams that thrive do the opposite.

Adapt the format, don't drop it

If a long retrospective isn't working, you can try a shorter one, but not none at all. In very fast workflows, the Sprint Retrospective can be complemented with short, frequent retrospectives, with themed retrospectives on the use of AI, or with incident retrospectives when something has gone wrong. What's not negotiable is that the team devotes time to inspecting and adapting the way it works.

Measuring progress

In Scrum, you measure to make decisions, not for the sake of measuring. **The fewer metrics, the better:** measuring is expensive, and every indicator should serve to deliver more value to the client. The aims of measuring progress are to plan sprints realistically, set the pace, synchronize teams, and forecast delivery dates. For all of them, Scrum measures **the work left**, not the work done.

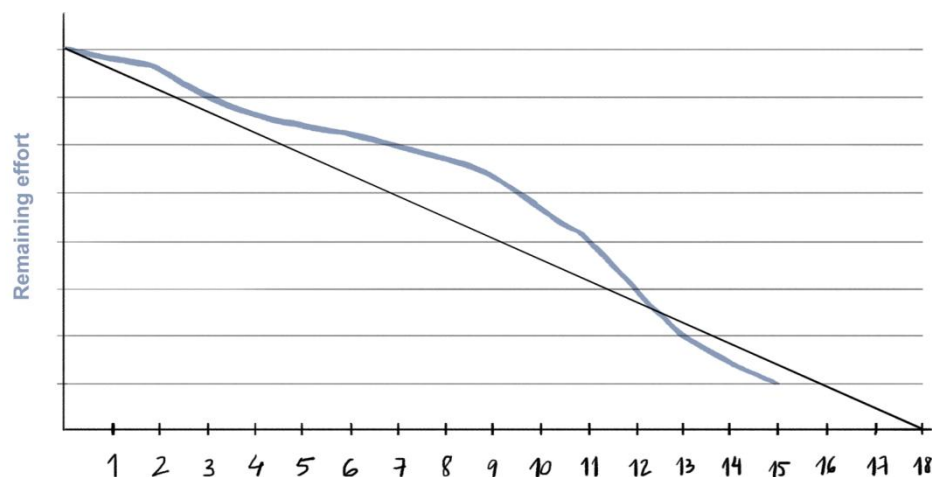
How much longer?

Measuring the work done is easy, but it doesn't help calculate progress. "This was supposed to take a week and three days have gone by, so there are four left" sounds logical, but it rarely holds up, because surprises come up or shortcuts are found. That's why progress is determined by the work remaining, as children know perfectly well when they ask, over and over on a trip, whether there's much further to go.



Burndown and burnup charts

The developers update the **burndown chart** during the sprint, ideally every day, to catch deviations early. The vertical axis shows the work outstanding, and the horizontal axis, the days of the sprint. When you're working with stories rather than tasks, it's updated as each story is completed. If the line stays above the diagonal of ideal progress for several days, the sprint was underestimated; if it drops faster, it will finish early.

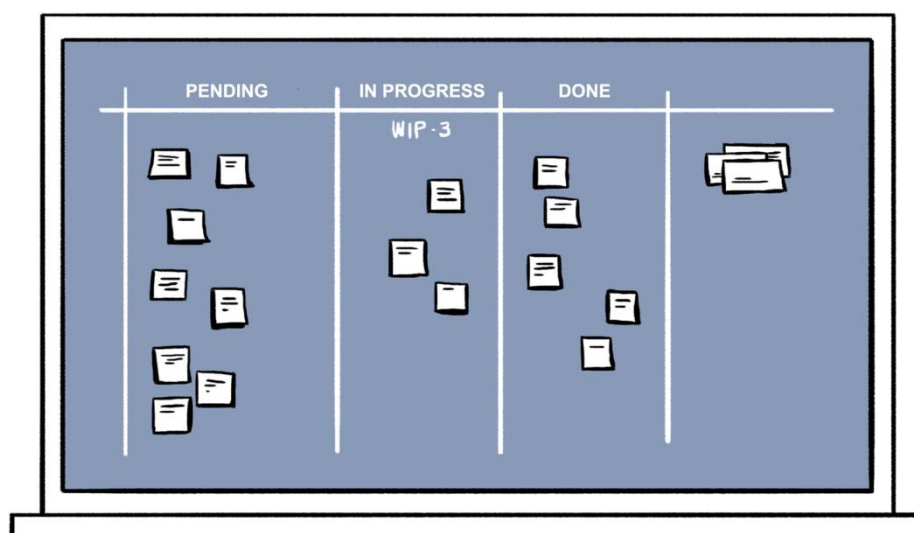


The **burnup chart** is a forecasting tool that belongs to the Product Owner. It represents what has been built and what remains of the Product Backlog over the sprints, with pessimistic, realistic, and optimistic forecasts. It's a living document that shows the forecasts after each change to the Product Backlog, not a stable plan.

Video: [Burndown chart](#) and [Burnup chart](#) (→"Bibliographic references").

Kanban and workflow

Kanban is a technique for visually managing a continuous flow of work, with or without sprints. The team places its work items on a board whose columns show their state ("to do," "in progress," "done," and any others the team needs), so you can see at once where work is piling up.



The key piece is the work-in-progress limit, or **WIP** (work in progress): the maximum number of items that can be in a column at the same time. A WIP of 3 on "in progress" means the team doesn't work on more than three items at once. Limiting work in

progress forces the team to finish before starting something new and reduces **cycle time**, how long an item takes to cross the board from when it's started until it's finished.

Review as the constraint

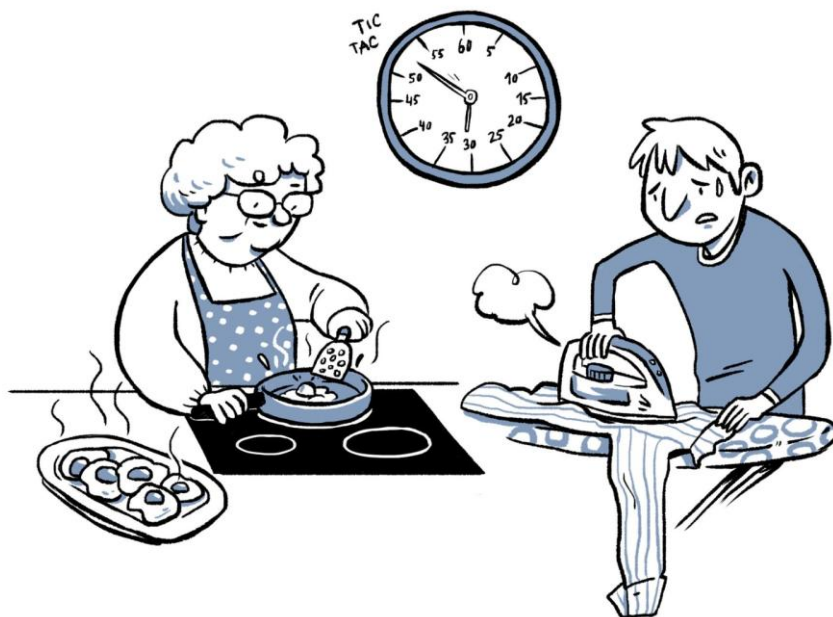
With AI, the board often shows a new phenomenon: work piles up in the review column. AI produces faster than people can check, and that is now the narrow point of the flow. Many teams add an "in review" column with its own WIP limit, so that production doesn't outrun the capacity to review and the backlog of unreviewed work stays visible. Measuring the full cycle time, review included, gives a truer picture of progress than counting what has been produced.

Estimating

Estimating serves to plan realistic sprints and forecast deliveries, but knowledge work rarely has a single solution, and how long it takes varies widely depending on who does it and in what circumstances. That's why agile estimation prefers **relative units**, and the estimates are always made by the people who will do the work.

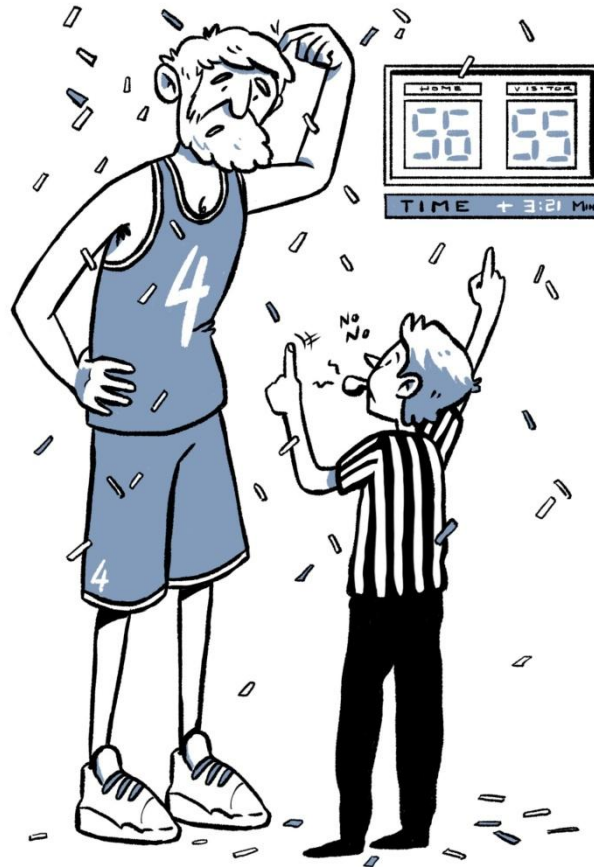
Story points

The most common relative unit is the **story point**: the size of a piece of work compared with another one already known. For a development team, a point might be the equivalent of building a login screen; for a graphic design team, laying out a leaflet. If someone estimates how many "fried eggs" it costs to iron a shirt, the answer depends on their skill at each task, and that's why the estimate is made by whoever will do the work.



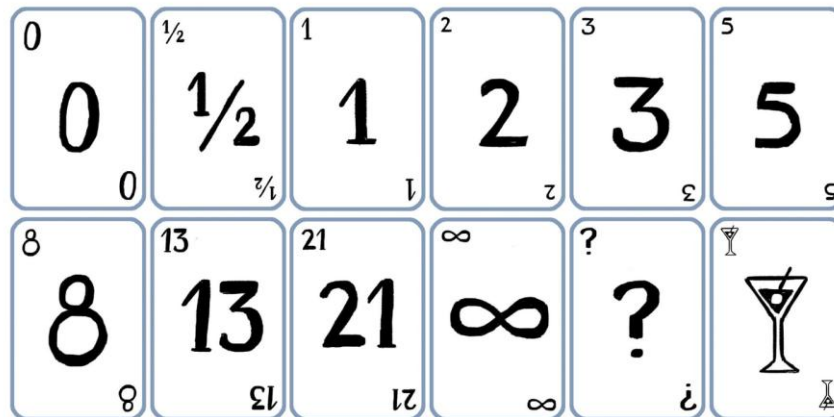
Velocity is the amount of work, in points, that the team completes in a sprint, and it helps forecast how much will fit in the next one. When estimating in time, it's worth

distinguishing **ideal time** (what the task would take while focused and without interruptions) from **elapsed time**, just as the 40 minutes of play in a basketball game take up more than an hour on the clock.



Estimation techniques

Planning poker, devised by James Grenning, is the most widespread technique. Each participant secretly chooses a card with their estimate, usually from a deck following the Fibonacci sequence, in which the values grow further apart as the estimate gets larger, because uncertainty grows too. The cards are shown at the same time and, if the estimates differ widely, the people who gave the most extreme ones explain their reasoning before repeating. It heads off circular arguments and gets everyone participating.



Wall estimation places sticky notes with the Product Backlog items on a wall: the developers set each note's horizontal position according to size, and the Product Owner sets the vertical position according to priority.

Video: [Estimating on the wall](#) (→"Bibliographic references").

Story points are easily misread when they're used to compare teams or as a measure of productivity, because they invite inflated estimates. That's why some teams use simpler alternatives, such as **T-shirt sizing** (XS, S, M, L, XL) or just three sizes (too big, too small, or just right). Others, with a mature workflow, stop estimating altogether (#NoEstimates) and forecast deliveries from the actual rate at which they finish work.

Related link: [To estimate or not to estimate, Scrum Manager Podcast \(in Spanish\)](#) (→"Bibliographic references").

Estimating with AI on the team

AI disrupts estimation from two sides. On the one hand, the work AI produces doesn't fit well into units designed for human effort: a task that used to cost eight points can be produced in minutes. On the other, reviewing that work does require human effort, and it's the part most often forgotten.

The practical consequence is that the team also estimates review and integration, not just production. Forecasting deliveries from data on the actual pace of work, rather than from estimates, is developed in Scrum Manager's [Product Owner](#) syllabus.

Three-level estimation

Consolidating trend (2026). This section describes practices that are gaining adoption quickly but are not yet widespread. It is not covered by the certification exam.

Some teams distinguish three kinds of work when estimating: work that AI produces almost autonomously, which isn't estimated in points but by the review time it will require; human work, which is estimated as always; and review and integration work, which is estimated explicitly because it's what creates the bottlenecks.

Output and outcome

With AI, the amount of work a team produces can grow a great deal without the value it delivers growing with it. That's why it's worth drawing a clear distinction between **output** (what the team generates: documents, features, pieces, lines of code) and **outcome** (the change that produces in users and in the business).

Speed isn't agility

Agility is the ability to respond to change while delivering value, not the speed of production. A team can be very fast and not very agile: it delivers a lot, but without knowing whether it's the right thing or being able to change course when the evidence calls for it. And a team can produce less and be very agile, because it delivers what has been validated, responds quickly to the market, and improves the way it works.

Not all speed is equal. The kind that matters is the speed of validation (how long it takes to find out whether an idea works), of learning, and of adaptation. The kind that can be illusory is the speed of production, if what's produced isn't used, isn't useful, or has to be redone.

Perceived and measured productivity

As we saw in the introduction, the feeling of going faster with AI doesn't always match reality. People tend to overestimate what the tool saves them and to leave out the time they spend reviewing and correcting. That's why productivity is measured with data on finished work, not with impressions, and it's measured at team level, not for each person.

Measuring outcomes

The useful metrics in a team that works with AI measure flow and outcome, not activity. Some questions that help choose them are these:

- Are we delivering more value, or just more work?
- Can we change course when the evidence calls for it?
- Are we learning at the same pace as we produce?
- Is quality holding up or degrading?

Measuring how much AI is used, or how much is produced with it, says nothing about value. Celebrating volume is one of the fastest ways to fall into the trap of producing a lot without making progress. Measuring the product's value is developed in Scrum Manager's [Product Owner](#) syllabus.

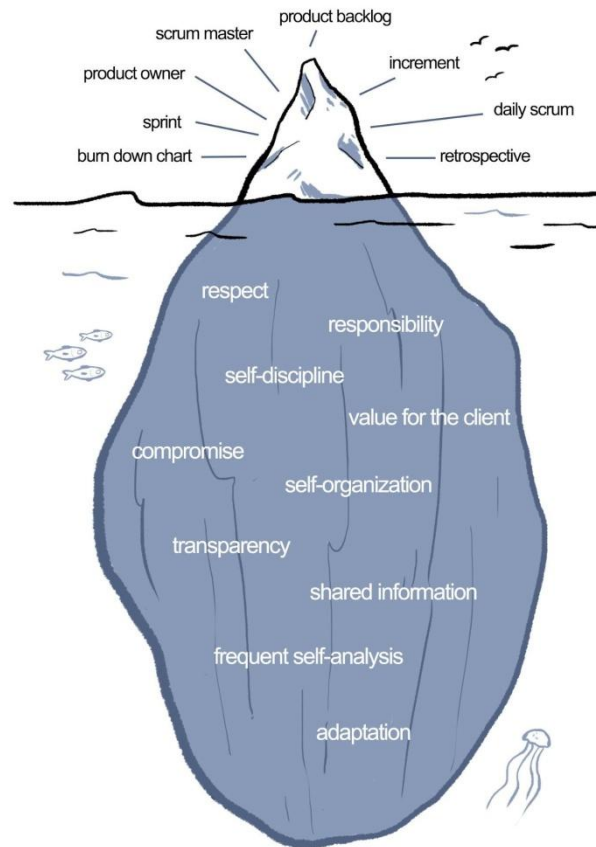
Part II — Principles and values

Internalizing and adapting.



Scrum principles and values

There's something Scrum courses don't usually tell you: the framework we've just studied, the practices and tools covered in part one, doesn't work without principles and values aligned with it.



The reason is that Scrum's practices aren't processes, that is, guidelines that guarantee results no matter who carries them out. Scrum rests on people's tacit knowledge and on a set of organizational values. The practices are the branches of the tree, and without good roots they won't bear fruit. It's worth reminding ourselves of this now and then, so we don't end up focusing on the tools alone.

You'll find companies where the principles are so thoroughly internalized that they develop their own model of agility, with practices different from the ones we've seen, and the result is innovative, high-quality products. And you'll find companies that apply every practice in standard Scrum and end up with nothing but demotivated, stressed people and mediocre results.

It takes flexibility to adapt what you've learned to the reality of each company and each project. Managing well means knowing the tools and fitting them to the team, taking what works and changing or dropping what doesn't, rather than expecting people to adapt to methods they don't believe in. AI is the clearest case: there's no single right

way to integrate it that works for every team, and each team has to find its own, relying on the principles.

Principles and values can be divided up, if only as a matter of semantics, according to whether they sit behind the practices or behind the organization's culture. **Principles** underpin the practices; **values** underpin the culture. In this part we'll look at both, at how the roles, artifacts, and events sustain them, and at a few further practices that, while not part of the standard framework, are commonly used.

The principles of Scrum

The purpose of artifacts, events, and other agile techniques is to ground the work in these eight principles. Alongside each one, we note what AI changes in how it is applied.

Value delivery

Early and continuous delivery of value to the client, which requires the client to work with the team and requires their vision to be shared and understood. With AI, this principle calls for distinguishing more carefully than ever between producing and delivering value, because the volume of what is produced can grow a great deal without any growth in value.

Continuous improvement

Agility means reflecting often on working methods, questioning how effective they are and adapting them, and that same self-critical effort is applied to products and services. Integrating AI is today one of the main subjects of that improvement, and the retrospective is its natural place.

Iterative and incremental development

The product isn't built to a detailed, complete plan drawn up at the start; it starts from a minimum viable product to which increments are added. AI shortens production cycles, but not the time users need to try things out and give their opinion, and the pace of increments has to respect that time.

Sustainable pace

A rhythm of work that avoids Parkinson's law (work expands to fill the time available for its completion) and the pressure that comes from discovering delays too late. AI brings a new risk to sustainability, the mental load of endlessly reviewing what AI produces, which the team has to keep an eye on.

Continuous attention to excellence

Using techniques that guarantee the quality of products and services and make it possible to catch errors early, or as they happen. When part of the work is produced by AI, this principle takes the form of a demanding Definition of Done and rigorous review.

Operational visibility

Information is shared clearly, so that collaboration is easier, impediments are identified early, and the whole team knows the state of the product and can contribute ideas. With AI, operational visibility includes the provenance of the work: what AI has produced and who has reviewed it.

Cadence and global synchronization

This principle matters most when several teams working on related products or services are synchronized. The aim is to make the frequency of meetings and delivery dates predictable, for example by synchronizing the different teams' sprints.

People over processes

The team's collective intelligence, its tacit knowledge, is what's responsible for the quality of the product. AI contributes explicit knowledge on a large scale, but judgment about what is valuable, understanding of the context, and accountability for the result still belong to people.

The values of Scrum

A company's culture is the sum of its organizational and governance characteristics, and it can either accelerate agility or hold it back. In organizations whose culture rests on values drawn from process-based industrial work, agile principles yield far more modest results than in organizations whose values amplify them. These are the six values that Scrum Manager® considers characteristic of an agile culture:

- Assertiveness.
- Valuing talent.
- Clarity.
- Trust.
- A flatter, less hierarchical structure.
- Shared purpose.

In addition, executive backing is essential: the company's leadership needs to have a compatible culture, be involved, and support people with enough training and resources.

Values in the face of AI

AI puts several of these values to the test. **Trust** becomes more complex, because the team has to learn to trust AI to the right degree, without the blind trust that accepts whatever it produces without reviewing it, or the distrust that gives up on its potential. **Clarity** requires decisions about AI (what is delegated, who reviews, what limits it has) to be explicit and known to everyone.

Valuing talent takes on a new meaning, because the talent that matters is the talent AI doesn't provide: judgment, experience, and the ability to work with other people. And **executive backing** now includes giving the team training on AI, clear criteria for its use, and the reassurance that integrating it is not a hidden threat to people.

Agile principles and values at the organizational level have their own syllabus at Scrum Manager: AgiLevel (<https://agilevel.com>).

People above roles

Artifacts and events help develop the agile principles, but people have to take those principles on board first. Without talented people who are committed and aware of their responsibilities, the practices are worth nothing, and without a culture with the values we've seen, neither talent nor commitment can grow.

Ken Schwaber made this point provocatively in a 2006 talk (Schwaber, 2006). With a brilliant, well-equipped team, Scrum makes it possible to build an increment in every iteration; but Scrum also works with a team that does everything badly, because at the end of each iteration it shows starkly where things stand. The framework makes reality visible, but it doesn't change it by itself: what changes it is people.

The Product Owner and the principles

The Product Owner is the person who makes **early and continuous delivery of value** possible. They manage and order the Product Backlog with a proactive stance, make changes whenever they judge it necessary, and communicate the client's vision clearly so that the whole team lines up behind it. Their attitude is key to making communication honest and fluid.

The developers and the principles

The developers are responsible for producing increments in every iteration and for keeping a **sustainable pace**. Committed to their work, they look critically at their results and their methods in order to improve them, and they are aware that what counts is their collective intelligence, not just each person's talent.

The Scrum Master and the principles

The Scrum Master helps the framework work and the team improve, resolving the impediments that hold back its progress. Having a dedicated Scrum Master makes most sense in teams with little experience, or in large organizations with frequent staff rotation or onboarding.

In small, stable teams with established agility, responsibility for running and improving the framework may already be internalized and taken on by the whole team. In those cases, the Scrum Master usually moves toward serving several teams or toward transforming the organization (→"A role that evolves").

Artifacts and events in service of the principles

Artifacts and events are not an end in themselves. Each one exists to make one or more principles possible, and knowing that relationship is what makes it possible to adapt them without losing their meaning. When a team modifies an artifact or an event, for example to integrate AI, the question it should ask is whether the change still serves the same principles.

The artifacts

- **Product Backlog.**
 - Value delivery: it makes it possible to absorb changes in the client's business and concentrate effort on whatever delivers the most value at each moment.
 - Iterative and incremental development: unlike a closed requirements document, it's a living document whose items and priorities change.
 - Operational visibility: it's an information radiator through which the Product Owner and the team share the product vision.
- **Sprint Backlog.**
 - Iterative and incremental development: it bounds the work of a sprint and sets the pulse of progress.
 - Operational visibility: it's an internal communication tool that shows at a glance the state of the sprint and how the work is shared out, including the work done by AI.
- **Increment.**
 - Value delivery: presenting a finished, ready-to-use part of the product at the end of each sprint makes it possible to check whether value is being generated.
 - Continuous attention to excellence: the Definition of Done guarantees that what is finished meets the quality criteria.

The events

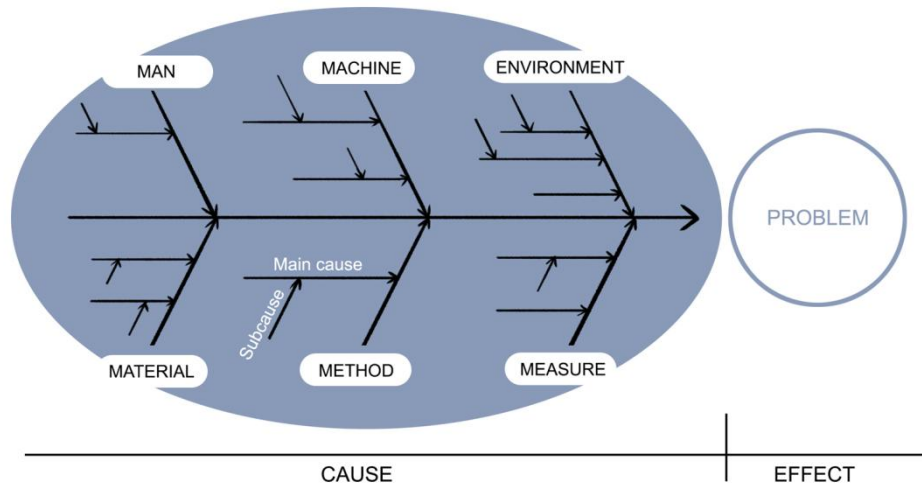
- **Sprint.**
 - Continuous improvement: its short iterations offer regular moments to reflect on the product and on the way of working.
 - Sustainable pace: it sets the pulse of progress.
 - Cadence and global synchronization: it makes meetings and deliveries predictable and allows teams to be synchronized.
- **Sprint Planning.**
 - Value delivery: the team and the Product Owner deepen their understanding of the vision together and agree on the Sprint Goal.
- **Daily Scrum.**
 - Operational visibility: the team shares where its work stands, including the work produced by AI, and spots impediments.
- **Sprint Review.**
 - Value delivery: the team and the stakeholders check whether what has been built is what is needed.
 - Continuous improvement: its conclusions help shape the next sprint.
- **Sprint Retrospective.**
 - Continuous improvement: the team analyzes how it works, including its collaboration with AI, to decide what to keep and what to change.

Ways of working for quality

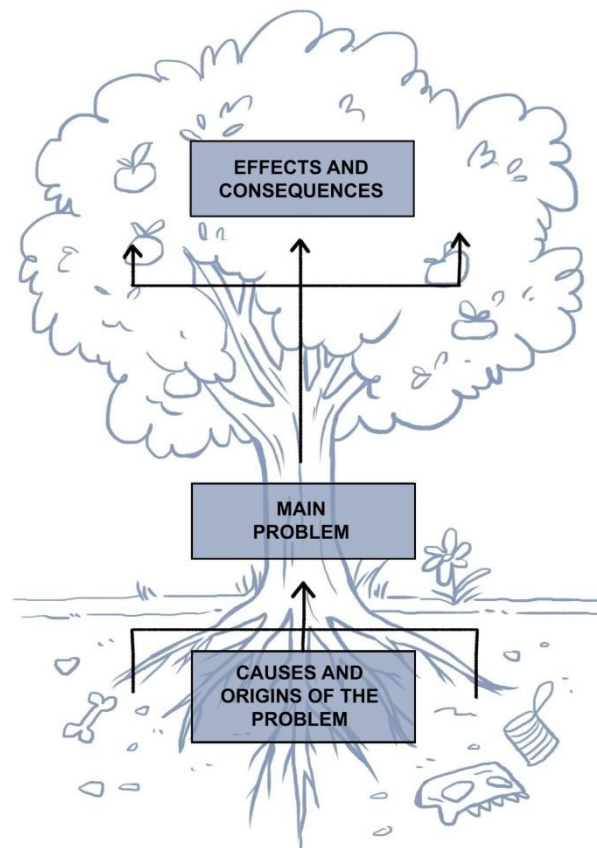
The professional community contributes many practices that complement the standard framework. We already saw the ones for tracking progress and estimating in part one; here we present some that focus on the quality of the work. Our interest isn't in explaining how they work in depth, but in presenting them as examples and encouraging you to look into them and experiment with them.

Retrospective diagrams: fishbone and tree

The **fishbone diagram**, also called the "Ishikawa diagram" or the "cause-and-effect diagram," represents the cause-and-effect relationships at work on a problem. The problem forms the central axis and, around it, its possible causes and sub-causes are listed. It comes from quality management and is very useful in retrospectives.



Tree diagrams help reach conclusions from the analysis of a problematic situation. They use the parts of the tree (roots, trunk, branches, fruit) to represent the causes of the problem, the problem itself, and its consequences, or the means for solving it and their outcomes. There's no single type of tree diagram and no standard system.



Mistake-proofing techniques

This means applying ways of working that prevent errors from occurring. Agility makes use of *poka-yoke* techniques and *andon* control devices, two concepts that come from lean manufacturing. Poka-yoke techniques serve to make human error impossible, like plug sockets that can't be connected the wrong way round, or to make the error obvious as soon as it happens, as spellcheckers do. Andon systems are visual indicators, using lights or graphical displays, that immediately alert the team to a fault in its working environment.

In programming, test-driven development (TDD) is common: you first write the tests the code has to pass, and then the code.

Specification by example as a test

Mistake-proofing techniques take on new importance with AI. The examples that accompany a story (→"Work items") can serve both as an instruction and as a test: they are written before the work is produced and used afterward to check whether what was produced meets them, just as in test-driven development.

The advantage is that the check doesn't depend on whoever produced the work. This matters especially with AI, which tends to accept as good whatever it has produced itself. Before commissioning a text from AI, a communications team can set out examples of what it should and shouldn't say, and then check the draft against them.

Working in pairs

Working in pairs means assigning two people to the same task, which they carry out simultaneously and jointly, usually alternating between doing the work and supervising it. While one carries out the task, the other watches and raises whatever observations are warranted. In programming, it's known as pair programming.

It's a good fit when the quality of the result depends above all on the knowledge of the person doing it. That's why trains run with a driver and an assistant when the technology can't completely rule out human error, and why we feel safer knowing there are two pilots working as a pair in an aircraft cockpit.

The human-AI pair

With AI, many people already work in a different kind of pair, in which AI produces and the person supervises. The analogy is useful, but it has an important limit: in the human-AI pair, supervision always belongs to the person, because AI can't be accountable for the result or reliably detect its own errors. Whoever works with AI keeps the role of active supervisor: they read carefully, question the proposals, and check the facts, rather than simply accepting whatever appears on the screen.

Cross-review

The pair of people still has its place, and in some cases it gains ground. When one person works with AI, having another person review the result brings an independent view that neither the AI nor the person who used it has. In addition, working in pairs between an experienced person and someone just starting out is one of the best ways to pass on tacit knowledge, the kind AI can't teach (→"Learning and developing people").

Part III — The Scrum Master's craft in AI-enabled teams

Supporting the evolution.

A role that evolves

In part one we saw that the Scrum Master is accountable for the Scrum Team's effectiveness. In this third part we'll look at what that accountability means when the team works with AI, and at the new areas the craft takes on.

The Scrum Master role has changed several times since Scrum began. It started out very focused on making sure the team knew and applied the rules of the framework, and over time it has shifted toward facilitation, team coaching, and organizational change. AI is one more step in that evolution, not the end of the role.

What AI takes on

Part of the Scrum Master's operational tasks can be supported by AI: preparing agendas, taking notes and summarizing meetings, suggesting activities for a retrospective, analyzing flow metrics, or spotting patterns in the team's comments. A Scrum Master who defines their value by those tasks alone will watch AI make them dispensable.

In some organizations the Scrum Master title has become less visible and its functions are spread across other positions, such as the people responsible for delivery or for technical teams. But the functions are still needed, because teams still need someone to look after the way they work, and AI makes that way of working more complex.

What grows

What AI can't do is precisely the heart of the craft: reading the state of a team, detecting a latent conflict, creating psychological safety, seeing the connections between problems that seem unrelated, and supporting people through change. To this are added new areas that we'll cover in this part:

- facilitating the work system between people and AI;
- helping the team agree on how AI is used and on its limits;
- promoting the team's AI literacy;
- safeguarding quality against work that only looks finished;
- resolving the impediments specific to AI-enabled teams;
- protecting people's learning and their well-being during the change.

New names for the role

Emerging trend (2026). This section describes a recent trend in the job market. It is not covered by the certification exam.

Some organizations have begun using new titles for functions close to those of the Scrum Master in AI-enabled teams, such as flow facilitator, AI orchestrator, or agile enabler. None has become established as a standard, and their definitions vary from one organization to another. What they have in common is a focus on the work system between people and AI, which is the focus this guide attributes to the Scrum Master.

Facilitating the human-AI work system

The Scrum Master has always facilitated interaction between people. In a team that works with AI, they also facilitate the whole **work system**: how work is divided between people and AI, how it flows from one to the other, where it is reviewed, and how it is integrated. For that they need an overall view, one that sees the flow from start to finish rather than each task separately.

Small batches

AI invites production in large quantities: a complete report instead of a section, fifty campaign pieces instead of five. But large batches are harder to review, hide errors, and delay feedback. Research on software teams that work with AI points to working in small batches as one of the conditions for AI to translate into better results (DORA, 2025).

The Scrum Master helps the team keep batches small even when the tool makes it possible to produce more: splitting the work into units that can be reviewed carefully, integrating them frequently, and asking for feedback early.

Review as the bottleneck

When AI speeds up production, the bottleneck in the flow shifts to review. If the team doesn't manage it, the work produced piles up waiting for review, people review in a hurry, and quality suffers. The Scrum Master helps make that pile-up visible, with a review column on the board and its own work-in-process limit, and helps the team adjust its rate of production to its capacity to review (→"Measuring progress").

Unintended effects

Every intervention in a system produces effects nobody had foreseen. A team that delegates writing up meeting minutes to AI may find that nobody reads them anymore, because they're too long. One that generates more proposals may overwhelm the people who have to assess them. The Scrum Master watches for these consequences and brings them to the retrospective so the team can decide what to do about them.

AI working agreements for the team

Agile teams work with **working agreements**: explicit commitments about how they collaborate, which the team itself sets and reviews. Integrating AI calls for its own agreements, which put into practice the four explicit decisions of "The Scrum Team and AI." The Scrum Master helps the team make them, write them down, and keep them alive.

What the agreements cover

AI working agreements usually set out:

- **What is delegated:** the tasks in which the team uses AI and those it reserves for people.
- **How it is reviewed:** who reviews each type of work produced by AI and against what criteria, consistent with the Definition of Done.
- **What information it is given:** what data can be used with AI and what can't, especially personal, confidential, and client data.
- **How it is made visible:** how the parts of the work produced by AI are indicated.
- **Which tools:** which ones are authorized by the organization and under what conditions.

A human resources team, for example, may agree that AI helps draft job postings and summarize regulations, but never takes part in assessing applications, and that no personal data leaves the tools authorized by the organization.

The organization's framework

The team's agreements sit within the organization's AI policies and the regulations that apply to it. The team decides within that framework, and the Scrum Master helps it understand the framework and resolve situations where the framework doesn't meet a need of the team, taking them to the appropriate people in the organization. Integrating AI across the different areas of an organization is covered in Scrum Manager's [Agile Bedrock](#) syllabus.

Limits for agents

Consolidating trend (2026). This section describes practices that are gaining adoption quickly but are not yet widespread. It is not covered by the certification exam.

When the team uses agents that carry out complete tasks, the agreements also include their **limits of action**: which systems and information they can access, which actions they can take without human approval and which they can't, and how an agent is stopped when its behavior strays from what was expected. These limits are reviewed more often than the rest of the agreements, because agents' capabilities change quickly.

AI literacy

To use AI with sound judgment, everyone on the team needs a basic understanding of how it works and what it can and can't do. This **AI literacy** isn't about knowing how to program AI, but about understanding enough to decide when to use it and how to review what it produces.

What the team needs to know

The basic content of AI literacy for a team is:

- **Its capabilities and its limits:** what kinds of tasks it handles well and where it fails most often.
- **Its characteristic errors:** it can invent data or references that look entirely genuine, reproduce biases, and give different answers to the same question.
- **The use of information:** what happens to the data it is given and what risks that poses for privacy and confidentiality.
- **Calibrated trust:** trusting AI to the extent that its reliability has been verified for each type of task, no more and no less.

An obligation in the European Union

In the European Union, the Artificial Intelligence Act (European Union, 2024) has required organizations that use AI systems to address their staff's AI literacy since February 2025. Following its amendment in 2026 (European Union, 2026), the obligation consists of taking measures to support that literacy, taking into account each person's training and experience and the context in which AI is used. The Scrum Master is a natural driver of those measures within the team.

Avoiding the single expert

A common risk is that a single person on the team becomes the AI expert everyone consults. The team then depends on that person, and knowledge isn't shared. The Scrum Master encourages learning to belong to the whole team: sharing what works in the Daily Scrum or the retrospective, practicing in pairs, and setting aside time to experiment together.

Quality and workslop

AI easily produces work that looks finished: well-written reports, eye-catching presentations, or code that works on a first test. But looking finished isn't being finished. The term **workslop** has been coined for AI-generated work that looks good but doesn't move the task forward, because it's incomplete, lacks context, or contains errors (Niederhoffer et al., 2025).

The cost is passed on

What's characteristic of workslop is that its cost isn't paid by whoever produces it, but by whoever receives it. A report generated in five minutes can cost the person who has to read it an hour to discover that it doesn't answer the question and to redo it. In addition, whoever sends it loses credibility with their colleagues, which erodes trust within the team.

In a Scrum Team, workslop shows up as increments that don't pass review, items that move backward, or stakeholders who stop trusting what they're shown. It's a form of debt that builds up silently.

How to prevent it

Prevention combines several practices we already know:

- a **Definition of Done** that requires human review of what AI produces;
- **acceptance criteria with examples**, which make it possible to check whether the work meets its purpose;
- **visible provenance**, so that whoever receives the work knows how it was produced;
- and a clear team agreement: whoever delivers a piece of work is accountable for it, however it was produced.

The Scrum Master helps the team talk openly about this phenomenon in the retrospective, without blaming anyone, and adjust its agreements accordingly.

Impediments in AI-enabled teams

One of the Scrum Master's classic responsibilities is resolving the impediments that hold back the team's progress. AI doesn't eliminate them; it adds new ones, and the Scrum Master has to learn to recognize them.

The new impediments

These are some of the most common impediments in teams that work with AI:

- **The review pile-up:** the work produced waits to be reviewed and slows down the flow.
- **Insufficient context:** AI doesn't have access to the information it needs and produces irrelevant results that have to be redone.
- **Unauthorized or poorly configured tools:** the team can't use the AI it needs, or uses it without the safeguards the organization requires.
- **Accumulated debt:** work produced in a hurry and reviewed lightly creates problems that surface later.
- **Dependency:** the team depends on a specific tool or provider, or on the only person who knows how to use them.
- **Cost:** using AI has a financial cost that can grow without anyone keeping an eye on it.

How to address them

Like any other impediment, the new ones are made visible, prioritized, and resolved, by the team itself when it's within its reach, or by the Scrum Master when they depend on other areas of the organization. Some have a technical solution, such as giving AI access to the documentation it needs; others require decisions from the organization, such as authorizing a tool or setting a budget.

Many of these impediments aren't visible at first glance. The Scrum Master looks for them in the signals of the flow (work piling up, items moving backward, timelines stretching out) and in the team's conversations, and brings them to the retrospective when their cause lies in the way of working.

Learning and developing people

AI poses a challenge to people's learning that is worth understanding. Reviewing what AI produces with sound judgment takes experience, and experience is gained by doing the work. If AI does the work and people only review it, how do they gain the experience they need to review it well?

The risk for those starting out

The risk is greater for people entering the profession. They used to learn by doing the simplest tasks, which AI now takes over easily, and by asking their more experienced colleagues, who are now often replaced by a query to AI. A team that doesn't pay attention to this may find itself, a few years from now, without people capable of reviewing the work AI produces.

What the team can do

The Scrum Master helps the team look after learning deliberately:

- reserving some tasks for less experienced people to carry out without AI, or with it but under close supervision;
- practicing **working in pairs** between people with different levels of experience, the best way to pass on tacit knowledge;
- reviewing AI's work together, explaining out loud why something is right or wrong;
- setting aside time to learn, including learning to use AI better.

Preserving tacit knowledge

Agility rests on people's tacit knowledge, and that knowledge is formed through practice and through working with other professionals. Looking after the team's learning isn't a secondary activity; it's the condition for the team to remain capable of providing the judgment AI lacks.

Psychological safety and change

Integrating AI is a profound change in the way of working, and like any change it creates uncertainty. Some people experience it with enthusiasm; others, with worry about their job, about their work losing its meaning, or about not knowing how to adapt. The Scrum Master makes sure the team can talk about all of this openly.

Psychological safety

Psychological safety is the confidence that on the team you can speak up, ask questions, disagree, or admit a mistake without fear of negative consequences. With AI, it's more necessary than ever. People need to be able to say that they don't understand how a tool works, that they made a mistake when reviewing what it produced, or that they're worried about the future of their job.

Without psychological safety, problems get hidden: nobody admits to having accepted work without reviewing it, or to not knowing how to use the tool everyone else seems to have mastered.

The tensions of change

Change creates tensions that the Scrum Master helps manage:

- **Fear of replacement**, which is eased by clear information from the organization about its intentions and by the team's participation in decisions about AI.
- **The mental load** of continuously reviewing other people's work, which can be more exhausting than producing one's own.
- **Uneven learning curves**, which can open gaps between those who adopt AI quickly and those who need more time.
- **Conflicts of judgment**, between those who want to use AI for everything and those who distrust it.

Small, reversible experiments

The best way to move forward in a change with so much uncertainty is to experiment on a small scale. The team tries a specific use of AI for one or two sprints, observes the results, and decides in the retrospective whether to keep it, adjust it, or drop it. Small experiments reduce risk, give people time to adapt, and turn the integration of AI into an empirical process, consistent with everything we've seen in this guide.

Antipatterns in AI-enabled teams

An antipattern is a way of acting that seems reasonable but systematically produces poor results. Knowing the most common antipatterns helps the Scrum Master recognize them early and help the team correct them. These are some of the ones that appear in teams that work with AI.

Workflow antipatterns

- **Illusory velocity:** the team celebrates producing more without checking whether it's delivering more value.
- **Producing without reviewing:** AI's work is accepted because it looks right or because it works on a first test.
- **Unplanned review:** the sprint fills up with production and there's no time left to review.
- **Overproducing for the Sprint Review:** more is presented than the stakeholders can assess.

Event antipatterns

- **Dropping the retrospective:** the moments for reflection are eliminated to go faster, just when they're needed most.
- **Shortening the sprint by inertia:** the cadence is changed because AI makes it possible to produce faster, not because the team needs to inspect more often.
- **The Daily Scrum without conversation:** the meeting is replaced by an automatic summary that nobody discusses.

Accountability and learning antipatterns

- **"The AI did it":** accountability for the result is diluted into the tool.
- **Delegating judgment:** AI is asked to decide what belongs to people, such as what is valuable or what risk is acceptable.
- **Implicit agreements:** each person uses AI in their own way, without team agreements.
- **The single expert:** all the knowledge about AI is concentrated in one person.
- **Measuring adoption:** how much AI is used is measured, instead of what results it produces.

Faced with any of them, the Scrum Master's response is the usual one: make it visible, bring it to the retrospective, and help the team decide how to correct it.

Scrum with AI beyond software

Most of what is published about AI in agile teams comes from software development. However, Scrum is applied in all kinds of knowledge teams, and AI has reached them too. This chapter covers what changes when the team doesn't develop software.

Where AI contributes most

AI performs best at writing, synthesis, information analysis, and ideation, which are central to many non-technical teams. In an experiment with product innovation professionals, individuals working with AI achieved quality comparable to that of pairs working without it (Dell'Acqua et al., 2025). In a study of customer support agents, AI improved average productivity, especially for the least experienced people (Brynjolfsson et al., 2025).

Its contribution is smaller in work that depends on personal relationships, on in-depth knowledge of a specific context, or on decisions with significant consequences for people.

What changes compared with software

Teams that don't develop software have two differences that shape how they use AI:

- **They don't usually have automated checks**, like the tests that verify code. Quality depends almost entirely on human review, which makes the Definition of Done and criteria with examples even more important.
- **They often work with sensitive information and regulatory requirements**. In the European Union, for example, AI systems used in selecting or evaluating people are considered high-risk, with specific obligations.

Examples by area

- A **marketing** team uses AI to generate variants of a campaign; the value lies in choosing, and validating with data, the ones that work.
- A **human resources** team uses it to draft communications and summarize regulations, but keeps decisions about people out of its hands.
- A **legal advisory** team uses it for preliminary research, with mandatory review of every reference, because AI can invent them.
- A **teaching** team uses it to prepare materials and activities, and reviews whether they suit the level and the learning objectives.

What we don't know yet

Evidence on the use of AI in Scrum Teams that don't develop software is still scarce: most studies analyze specific tasks or professions, not agile teams. That's why each team should treat its way of integrating AI as an experiment and learn from its own experience. How the impact of AI changes depending on the type of work is analyzed in Scrum Manager's [Agile Bedrock](#) syllabus.

Appendix

Glossary of the syllabus terms, bibliographic references, Scrum Manager's resources and professional community, and quality control of the training.

Glossary

AI working agreements: the team's agreements on what is delegated to AI, who reviews, what information it is given, and what limits it has.

Agility: evolutionary management based on people; it focuses on their tacit knowledge, culture, and talent, and removes tasks that add no value.

AI literacy: a basic understanding of how AI works, what it can do, what its characteristic errors are, and how to review what it produces.

Antipattern: a way of acting that seems reasonable but systematically produces poor results.

Artifacts: Scrum's tools that make the work transparent: Product Backlog, Sprint Backlog, and increment.

The committed: those directly involved in building the product who are accountable for its result; the members of the Scrum Team.

Explicit knowledge: codified knowledge, which can be passed on in formal, systematic language.

Tacit knowledge: personal, context-specific knowledge, hard to formalize and communicate.

Definition of Done (DoD): the team's agreement on the criteria a piece of work must meet to be considered finished.

Definition of Ready (DoR): an agreement on when a Product Backlog item is ready to enter a sprint.

Developer: each of the professionals who build the increment, not only programmers; together they form a cross-functional, self-managing team.

Empiricism: a way of working that makes decisions based on what is observed, grounded in transparency, inspection, and adaptation.

Epic: a story that is too large and poorly defined to serve as a work item.

Scrum Team: the team formed by the Product Owner, the developers, and the Scrum Master; ten people at most.

Specification by example: accompanying a story with concrete cases of the expected result, which serve as both instruction and test.

Planning poker: a technique devised by James Grenning for estimating the size of work items with cards.

Events: sprint, Sprint Planning, Daily Scrum, Sprint Review, and Sprint Retrospective.

Evolutionary management: management that delivers a minimum viable product early and keeps increasing its value.

Predictive management: management that seeks predictable results in time, cost, and scope; also called sequential engineering.

Burndown chart: a chart of the sprint's remaining work against its days.

Burnup chart: the Product Owner's chart showing what has been built and the forecast of what remains.

User story: a short description of the value the product must provide to the user, written from their perspective.

The involved: stakeholders who contribute to the project without being accountable for the sprint's result, such as management or user support.

Increment: a finished, tested, and working part of the product; a sprint may have one or several.

Kanban: a technique for visually managing a continuous flow of work on a board.

Sprint Goal: the value a sprint must deliver; it guides the team's decisions while the sprint is under way.

Provenance: information about which parts of a piece of work AI has produced and who has reviewed them.

Product Backlog: the ordered inventory of everything the product may need; the Product Owner maintains it.

Product Owner: the person who makes the client's decisions and is accountable for maximizing the product's value.

Story points: a relative unit for estimating the size of work, by comparison with a known piece of work.

Refinement: the work of ordering, detailing, and pre-estimating the Product Backlog; no more than 10% of the team's capacity.

Sprint Retrospective: a meeting in which the Scrum Team reflects on how it works and plans improvements.

Sprint Planning: the meeting that starts the sprint and sets its goal and the work needed to achieve it.

Sprint Review: a meeting at the end of the sprint to inspect the increment with the stakeholders.

Scrum: an agile development model with self-managing teams, incremental development, overlapping phases, and quality based on tacit knowledge.

Daily Scrum: a meeting of 15 minutes at most in which the developers synchronize their work.

Scrum Master: the person accountable for the Scrum Team's effectiveness, serving the team, the Product Owner, and the organization.

Psychological safety: the confidence that on the team you can ask questions, disagree, or admit a mistake without fear.

Sprint: a period of four weeks at most in which one or more increments are built.

Sprint Backlog: the list of the work needed to achieve the Sprint Goal; it is the developers' territory.

Task: a work item smaller than a story, taking a day or less.

Cycle time: the time a work item takes from when it is started until it is finished.

Ideal time: working time with no distractions or impediments; elapsed time is usually longer.

Workslop: AI-generated work that looks finished but doesn't move the task forward.

Velocity: the amount of work, in points, that the team completes in a sprint.

WIP: the maximum number of work items that can be in a stage at the same time (work in process).

Bibliographic references

The syllabus's reference works and resources, in alphabetical order.

Beck, Kent (1999). *Extreme Programming Explained: Embrace Change*, Addison-Wesley.

Beck, Kent, et al. (2001). *Manifesto for Agile Software Development*: <https://agilemanifesto.org>

Becker, Joel; Rush, Nate; Barnes, Elizabeth; and Rein, David (2025). *Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity*, METR.

Brynjolfsson, Erik; Li, Danielle; and Raymond, Lindsey (2025). "Generative AI at Work," *The Quarterly Journal of Economics*, 140(2), pp. 889-942.

Cockburn, Alistair (2006). *Agile Software Development: The Cooperative Game* (2nd ed.), Addison-Wesley.

Dell'Acqua, Fabrizio, et al. (2025). "The Cybernetic Teammate: A Field Experiment on Generative AI Reshaping Teamwork and Expertise," *NBER Working Paper 33641*, National Bureau of Economic Research.

DORA (2025). *State of AI-assisted Software Development 2025*, Google Cloud.

European Union (2024). Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act), *Official Journal of the European Union*.

European Union (2026). Regulation (EU) 2026/1744 of the European Parliament and of the Council of 8 July 2026 amending Regulations (EU) 2024/1689, (EU) 2018/1139 and (EU) 2023/1230 as regards the simplification of the implementation of harmonised rules on artificial intelligence (Digital Omnibus on AI), *Official Journal of the European Union*.

Grenning, James (2002). *Planning Poker, or How to Avoid Analysis Paralysis While Release Planning*, Renaissance Software Consulting.

Hackman, J. Richard, and Vidmar, Neil (1970). "Effects of size and task type on group performance and member reactions," *Sociometry*, 33(1), 37–54.

Highsmith, Jim (2001). "History: The Agile Manifesto": <https://agilemanifesto.org/history.html>

Ishikawa, Kaoru (1985). *What Is Total Quality Control? The Japanese Way*, Prentice Hall.

Kerth, Norman L. (2001). *Project Retrospectives: A Handbook for Team Reviews*, Dorset House.

Naur, Peter, and Randell, Brian (eds.) (1969). *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7th–11th October 1968*, NATO Scientific Affairs Division.

Niederhoffer, Kate; Rosen Kellerman, Gabriella; Lee, Angela; Liebscher, Alex; Rapuano, Kristina; and Hancock, Jeffrey T. (2025). "AI-Generated 'Workslop' Is Destroying Productivity," *Harvard Business Review*.

Nonaka, Ikujiro, and Takeuchi, Hirotaka (1995). *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*, Oxford University Press.

Nonaka, Ikujiro, and Toyama, Ryoko (2004). "Knowledge Creation as a Synthesizing Process." In Hirotaka Takeuchi and Ikujiro Nonaka (eds.), *Hitotsubashi on Knowledge Management*, 91–124, John Wiley & Sons.

Orr, Ken (2002). *CMM versus Agile Development: Religious Wars and Software Development*, Cutter Consortium, Agile Project Management Executive Report, vol. 3, no. 7.

Parkinson, Cyril Northcote (1955). "Parkinson's Law," *The Economist*, November 19.

Schwaber, Ken (1995). "SCRUM Development Process," *OOPSLA '95 Workshop on Business Object Design and Implementation*, Springer.

Schwaber, Ken (2006). *Scrum Et Al.*, Google Tech Talks.

Scrum Manager Podcast (2023). *Cómo crear un backlog* [How to create a backlog], episode 1x02, in Spanish: <https://open.spotify.com/episode/7hGoEqV2uZL47uZZN6GXfk?si=2eb790bebe644aa8>

Scrum Manager Podcast (2023). *Cómo se crea una historia de usuario* [How a user story is created], episode 1x03, in Spanish: <https://open.spotify.com/episode/07cSgNf9znGHU93kAm2qaU?si=9813747ed602458b>

Scrum Manager Podcast (2023). *Eventos de scrum* [Scrum events], episode 1x04, in Spanish: <https://open.spotify.com/episode/7basfleLN9L5A6FMOGHaf3?si=7d5e6974f32f49af>

Scrum Manager Podcast (2023). *Definition of Done*, episode 1x06, in Spanish: <https://open.spotify.com/episode/0GjNQprSs7wbQ8s6G0IGXu?si=2d94f9fd7d1f4770>

Scrum Manager Podcast (2023). *Estimar o no estimar* [To estimate or not to estimate], episode 1x10, in Spanish: <https://open.spotify.com/episode/5qa41nYkK9JSmKVg0E5qt0?si=14aafb0778a44d09>

Scrum Manager Podcast (2024). *Las 5 claves de un buen product owner* [The 5 keys to a good Product Owner], episode 2x02, in Spanish: <https://open.spotify.com/episode/5OwC380a6pWE1iV59Mchqo?si=80e27682603a42b4>

Scrum Manager Podcast (2024). *5 características de una buena historia de usuario* [5 characteristics of a good user story], episode 2x07, in Spanish: <https://open.spotify.com/episode/5PMBllhu0cuW9GkA2xNGV5?si=6e579095f4e542d5>

Scrum Manager YouTube (2017). *Burn up chart*: <https://www.youtube.com/watch?v=0yoL-GK251A>

Scrum Manager YouTube (2017). *Purpose of Agile vs. Predictive Management*: <https://youtu.be/mz1lhdHA2Ts?si=9jv5os3kXlZ73geu>

Scrum Manager YouTube (2018). *Managing plans and ideas*: <https://www.youtube.com/watch?v=1VkDbgeNRpk>

Scrum Manager YouTube (2021). *Burn down chart*: https://youtu.be/QQz_ujHV0c?si=qXrjdt1jOBuLu6mc

Scrum Manager YouTube (2021). *Estimating on the wall*: https://youtu.be/zfh7qVhdYzE?si=GxHQVu8s90QAji5_

Scrum Manager YouTube (2021). *Scrum (en)*: <https://youtu.be/aS30AGwaLxw?si=sy7kgdBvzDufUPfs>

Takeuchi, Hirotaka, and Nonaka, Ikujiro (1986). "The New New Product Development Game," *Harvard Business Review*, 64(1), 137–146.

Turner, Richard, and Jain, Apurva (2002). "Agile Meets CMMI: Culture Clash or Common Cause?," in *Extreme Programming and Agile Methods — XP/Agile Universe 2002*, Lecture Notes in Computer Science, vol. 2418, Springer.

Information, resources, and professional community

Scrum Manager maintains an ecosystem of open resources and professional community for continuing to learn after this syllabus:

[Agile Club →](#)[Skill Arena →](#)[Academy →](#)[BoK/Wiki →](#)[Community →](#)[Comic strips →](#)[About the Scrum Manager® certification →](#)[Frequently asked questions about Scrum Manager® courses and exams →](#)

Follow us on  [LinkedIn](#)

Continuous improvement and quality control

Scrum Manager's quality control is based on the evaluations of its students. With your opinion you help us maintain the level of our materials, courses, centers, and teachers.

If you have taken part in a training activity audited by Scrum Manager®, we ask you, and thank you in advance, to rate the quality of the training you received. The information collected is anonymous. You can send us your comments from the "Members' Area": <https://scrummanager.com>