SCRUM MASTER

Scrum Manager®

Scrum Master

Core syllabus 1

Version 4 (2026)

Scrum Manager ®

Version date: july 2026
Version author: Scrum Manager® team
Illustrations: María de la Fuente Soro and Ana Andrés Soria
Acknowledgements: Alexander Menzinsky, Claudia Marcionni y Rubén Álvarez.

Index

守破離

Shu Ha Ri

Learning any skill happens in three stages:
Shu: you pick one technique and, taking it to be the right one, try to imitate it.
Ha: you collect more techniques.
Ri: you experiment and invent new techniques by mixing, combining and modifying what you know.

Shu-stage techniques apply broadly. Ri-stage techniques only work in specific situations, and it takes expert knowledge to know when and how to use them.

"You can't win in a competitive industry using shu techniques."
Alistair Cockburn

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.

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 a range of projects in an agile way.

Although the model first took shape inside technology companies, today you'll find it in innovative environments of every kind. The common thread is always the same: knowledge work, instability, and fast, constant change. Plenty of organizations have found that in industries like these, agile management is simply the best fit.

This syllabus will suit you if you work in any of the so-called "knowledge companies", in a shifting environment, and especially if your work involves **managing cultures and people**. Above all, it will suit you if **the value of your product depends on the talent of motivated people** rather than on the processes and tools they happen to use.

***

The content is divided into three parts:

The **introduction** puts the birth of Scrum in context and defines what agility means in a business setting. It's worth reading, particularly if this is your first time reading about agile management. If it isn't, you can skip it — though you may still pick up something new. It also explains the distinction Scrum Manager® draws between agile "practices" on one hand and agile "principles and values" on the other, which is key to everything that follows.

Part one focuses on the most widespread **Scrum practices**. It covers the "roles", "artifacts" and "events" that have settled into a standard over time and that you'll come across in any environment using this management model.

Part two digs into the **principles and values** those concepts grew out of. It explains what Scrum Manager® calls Scrum "principles" and "values": a freer but more deliberate use of the model. It closes with a few further practices that, while not part of the standard framework, are commonly used and round out the agile management toolkit.

***

Discussions of Scrum are full of terms that take on a specific meaning within the world of agile management. Whenever a term of this kind is introduced, it will appear in quotation

marks — both to make it easier to search for more information and to flag that the word carries a special connotation.

Introduction

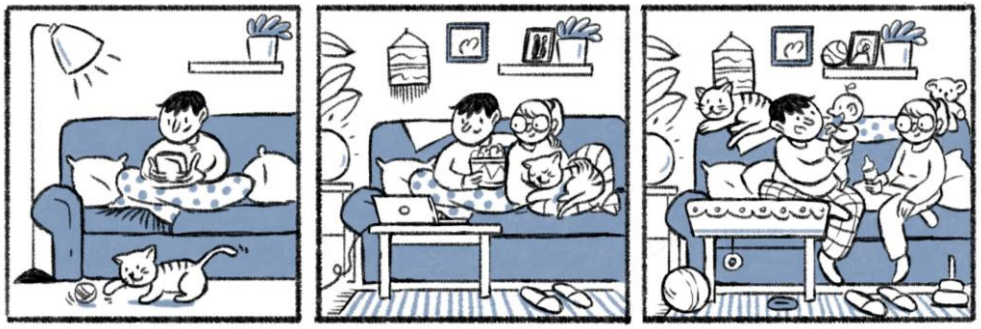
Agility

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 finished project lands on the agreed date, within budget, and with every feature from the original plan, it counts as a success.

Reasonable as that sounds, if you work in an industry defined by constant, rapid change, you'll soon run into its drawbacks. That definition of a successful project 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. Take a client who wants 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.

But today we build and sell products that bear no resemblance to that. For a start, they can be intangible — a film, or a mobile app. You can try new things out as you go, seeing empirically what works and what doesn't. You can adjust on the fly. You can also 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 interesting and prompt you to rethink your product priorities. Staying competitive means being able to respond quickly in uncertain working conditions: where requirements aren't stable when new products or services are first conceived; where clients need to start using the product as soon as possible and improve it continuously; 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 seemed out of step with what the so-called "knowledge

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

The working environment of these companies bears very little resemblance to the one that gave rise to predictive project management.

We now have markets moving so fast that 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. The product is built at the same time as requirements are revised and added. The client starts out with a more or less clear vision, but the level of innovation involved — and the speed at which their business environment moves — makes it impossible to foresee the end result in any detail.

There are product managers today who don't need to know which 200 features the finished product will have, or whether it will be ready in 12 months or 16. There are clients who need a first version with minimum functionality within weeks, not a complete product in one or two years. What matters to them is getting a new concept to market fast and building 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: a resolution and 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 biased in some respect. It then serves as the thesis for a new dialectical movement, and so the process carries on, zigzagging upward in a spiral." (Nonaka 2004)

Agile frameworks didn't emerge as a thesis of knowledge, but as the antithesis of what software engineering had been developing up to that point.

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 scientific discipline that would bring a systematic, measurable approach to the development, operation and maintenance of computer systems. That translated into an attempt to apply process engineering to software, and so "software engineering" was born (Naur & Randell 1969). This first strategy — the thesis — rested on two pillars:

- **Process engineering.** Industrial production environments had a basic 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 or highly qualified people; as long as the processes are good, the result will be good.

- **Predictive management.** A style of management focused on making sure schedules and budgets are met.

While the discipline evolved and refined itself through a succession of process models and project management bodies of knowledge (MIL-Q-9858, ISO 9000, ISO 9000-3, ISO 12207, SPICE, SW-CMM and so on), doubts were surfacing in the software industry and the whole strategy was being called into question.

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 & 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 development — like other knowledge-based work — can 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 (→"Taking project management apart", "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."

In February 2001, 17 software professionals came together at the initiative of Robert C. Martin. Kent Beck, who a couple of years earlier had published the book setting out his new *Extreme Programming methodology* (Beck 1999), was among them. They all had something in common: they were critics of process-based production models.

They met at Salt Lake City to discuss the processes software teams were working with.

It was at that meeting that the term "agile methods" was coined, to describe those emerging as an alternative to formal methodologies such as CMM-SW (the forerunner of CMMI), PMI and SPICE (the initial project behind ISO 15504, itself the forerunner of ISO 33000). These were seen as excessively heavy and rigid, given their prescriptive character and their heavy reliance on detailed planning ahead of development.

The participants boiled down into four statements what has come to be known as the Agile Manifesto: the values these methods rest on. They are the ones set out above and expanded on below. They also established 12 principles, which we'll come to at the end.

"Individuals and interactions over processes and tools"

The most important of the four. There's no question that processes help: they act as a guide for how to operate, and having the right tools improves efficiency. But some work requires talent, and it needs motivated people to bring it.

In process-based production, the aim is for the quality of the result to follow from the processes rather than from the knowledge of the people carrying them out. In agile development, by contrast, processes are only an aid — a support to guide the work.

Defending processes at all costs leads to the claim that they can deliver extraordinary results with mediocre people. The truth is that this doesn't hold when creativity and innovation are what's needed.

"Working software over comprehensive documentation"

The Agile Manifesto doesn't treat documentation as useless — only unnecessary documentation. Documents are a physical medium for recording and communicating information that matters to the project, and legal or regulatory requirements may make them compulsory. But they should matter less than the product.

What does that mean in practice? Being able to anticipate how the finished product will work by looking at prototypes and completed parts is stimulating, rewarding feedback that generates ideas nobody could have conceived at the outset. That's why producing a highly detailed requirements document before starting is so often a waste of time.

You could argue that detailed documentation makes it easier to pass information between the people involved in a project, but that's rarely how it plays out. It lacks the richness and the value that come from direct communication and from interacting with product prototypes. Worse, it doesn't just miss out on those benefits — it gets in the way, creating bureaucratic barriers between departments and between individuals.

So wherever possible, documentation should be kept to the bare minimum. Ideally, anything that consumes effort without adding direct value to the product should go.

"Customer collaboration over contract negotiation"

The goal of an agile project isn't to control execution so that the initial plans are met; it's to continuously deliver as much value as possible to the product.

As we've said, when you're developing products that evolve continuously — a web application, say — you can't set out in a closed requirements document what the finished product should look like. It's more effective to take direct feedback while the product is being built, and redefine and improve the requirements for whatever is still to come.

For the client to be aware of changes as they arise, they need to be alongside the team throughout. The best client–team relationship is one of direct involvement and collaboration, not a contractual one, which tends to mark out who is responsible for what at the start of the project and leave it there.

"Responding to change over following a plan"

The core values of agile management are anticipation and adaptation. They differ from those of orthodox project management: planning and control to guarantee the plan is met.

When you're developing products with unstable requirements, where change and rapid, continuous evolution are a given, the ability to respond is far more valuable than the ability to monitor and safeguard plans.

The 12 principles of the Agile Manifesto

Alongside the four statements above, the Agile Manifesto sets out these 12 principles:

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

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, a scrum is the formation in which both teams crouch and bind together, pushing to win the ball without handling it.

For our purposes, though, we need to travel to 1980s Japan, when researchers Ikujiro Nonaka and Hirotaka Takeuchi gave the term a second meaning.

They identified a novel 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 (Nonaka 1986). They compared the way these self-managing teams worked to the way rugby players advance in formation — hence the term.

Although this way of working originated in technology-product companies and in industrial manufacturing, from 1995 onwards it began to be applied to the software industry too. That year, Ken Schwaber presented a software development methodology based on a Scrum environment at OOPSLA (the annual Object-Oriented Programming, Systems, Languages & Applications conference), using that very term (Schwaber 1995). This first framework laid out a series of phases and "artifacts": pregame, game, postgame, planning, sprints, wrap. Some have stuck around and we'll look at them, but on the whole 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 a community of practitioners who determine which practices prove most useful. The original spirit does hold, though: practices should help teams manage themselves and maintain a continuous flow of progress, producing results iteratively and often. That's the appealing, exciting part of working in companies like these.

Among the events and practices that have been added along the way are retrospectives, Product Backlog refinement meetings, the DoR (Definition of Ready), story maps and more.

Scrum Manager® uses the term "Scrum" in its original sense: the one Nonaka and Takeuchi gave it.

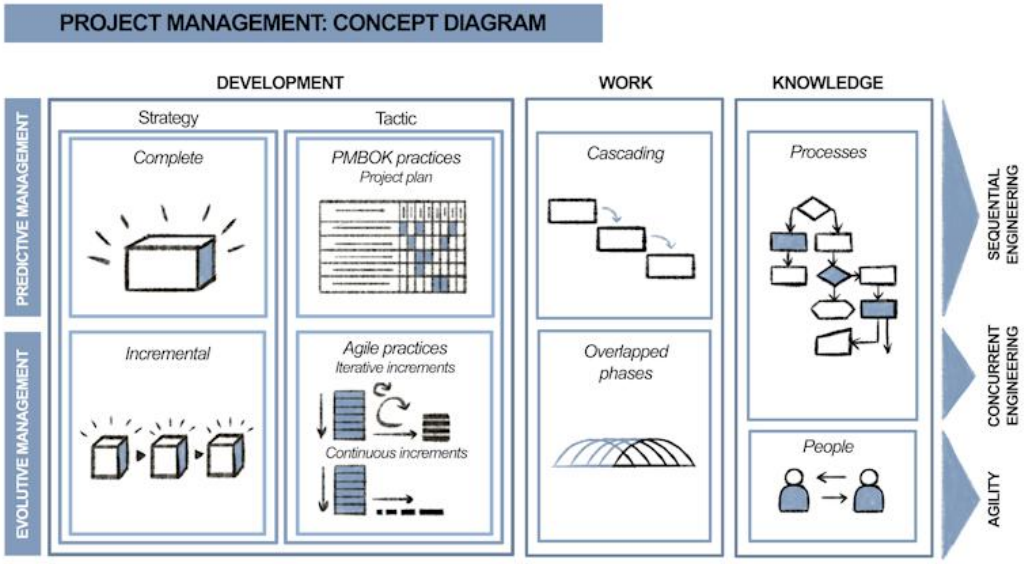
Taking project management apart

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 section we'll look past the labels and boil these frameworks down to their basics: the principles underneath them and the strategies they use to put those principles into practice.

We'll use three concepts and two management models as our coordinates.

- The three concepts are: development, work and knowledge.
- The models, which we've already mentioned, are: predictive management and evolutionary management.

These five ideas clear up and simplify what looks like a maze of frameworks, because in the end every one of them fits somewhere on this map.



1. 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 of the project. It is full and detailed, and it's what estimates are based on. The initial plan is used to schedule the work and the resources needed. During execution, management is about making sure what was planned is delivered.
- **With incremental development,** the full description of what you want isn't available at the start. It's filled in and evolves as development goes along, and it can be managed with two different tactics:

- **Continuous incremental development:** using techniques to achieve a continuous flow of development of the features or parts of the product, which are delivered to the client on an ongoing basis.
- **Iterative development:** using fixed-time or "timeboxing" techniques to keep product increments coming at a steady rhythm. This is the production framework used when applying the standard Scrum framework, which defines each development iteration as a sprint (→"Events"). At the end of each one comes a product "increment": a deliverable part that is ready to use.

2. Work

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

- **Sequential work splits the work into phases.** A new phase starts once the previous one is finished. The most familiar example is the waterfall cycle defined in software engineering, with its phases of requirements definition, analysis, design, coding, testing and implementation.
- **Working concurrently means overlapping those phases in time.** Sticking with the software engineering example, all the phases listed above would be revisited simultaneously and continuously.

3. Knowledge

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

- **In process-based production**, knowledge is explicit. The quality of the result lies, for the most part, in the processes and technology used.
- **In people-based production, knowledge is tacit.** The quality of the result depends on the experience of the people in the organization — not on following a process correctly, but on the people doing the work being motivated and talented.

Here's an example of the difference between explicit and tacit knowledge: dinner made in a bread machine versus 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 case, the cook's talent is what counts. An amateur won't turn out the same dish as a fine-dining chef. Both will benefit from good tools — non-stick pans, well-sharpened knives — 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, refers to knowledge that is transmittable in formal, systematic language." (Nonaka 1995)

Sequential engineering

Sequential engineering, also known as predictive management, aims to deliver predictable results. Under these models, a successful project is one that develops the planned product without exceeding the agreed timeline or resources. Predictive management works with "complete" development and traditional planning practices.

In software, the main reference points for this kind of management are PMI and IPMA, along with process models such as CMMI, ISO 33000 and SPICE. All of them use 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 uses incremental development, achieved either by keeping to a rhythm of short, cyclical iterations or by maintaining a constant development flow.

It can be carried out with process-based production (concurrent engineering) or with people-based production (agility). The distinction matters, because without it you end up with confused situations where simply applying the standard rules of Scrum (an iterative increment cycle with defined roles and artifacts) — or simply using kanban visual management techniques to keep work flowing — gets mistaken for agility.

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

Video: Managing plans and ideas (→"References").

Video: Purpose of Agile vs. Predictive Management (→"References").

Scrum

To recap, we come back to the characteristics of Scrum (→"Where Scrum came from") and see how they fit within the characteristics of agile management:

- It uses an incremental development strategy (which may be iterative, using timeboxing, or continuous).
- It overlaps the different phases of development.
- It bases the quality of the result on people's tacit knowledge and creativity.

On top of that, Scrum is also characterized by work in autonomous, self-managing teams that share their knowledge and learn together. Hence the name, and the metaphor of advancing as a scrum.

Telling practices apart from 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 traditional or standard framework. In this first part we explain what that framework consists of, and which roles, artifacts and events make it up. Think of it as a template rather than a mold. **These aren't rules to be followed** to the letter to guarantee speed or quality; **they're 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 — it's concurrent engineering.** Agility requires trust in the team's capabilities. The tools and processes we use should help us manage the work, not the people, because the end goal is for teams to be able to manage themselves.

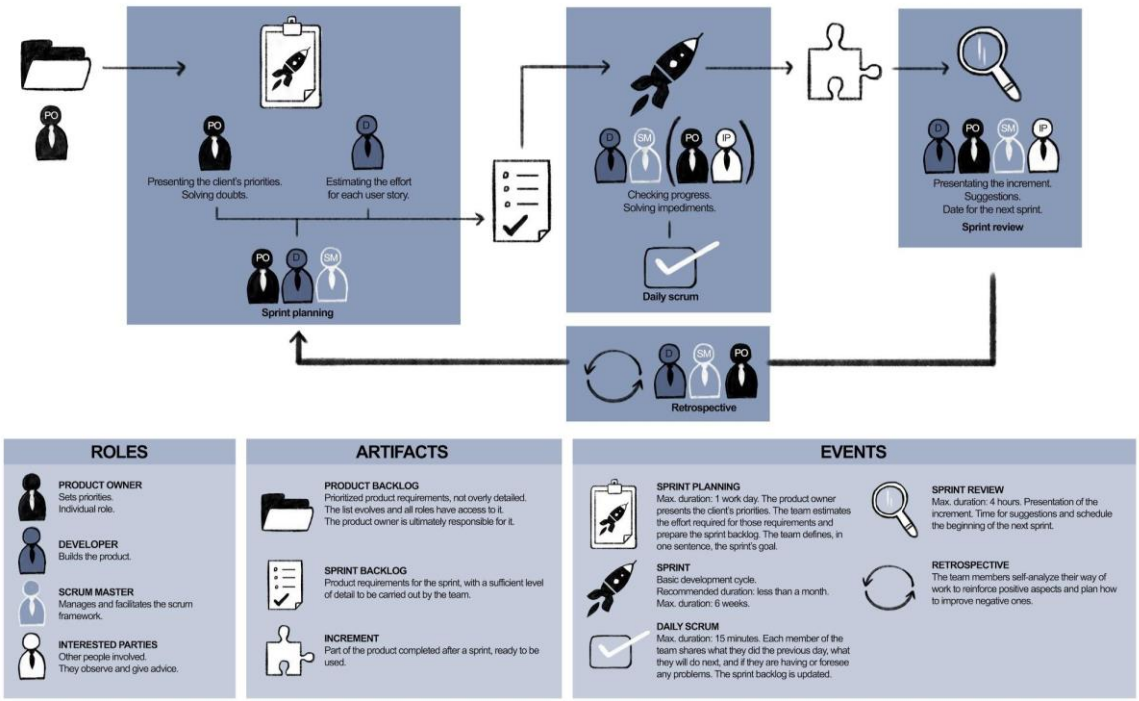
Once you've reached a steady iterative flow, you can start to move beyond a standard Scrum framework. That's the moment to unlearn the practices and lean on Scrum's principles and values instead, adapting Scrum and other techniques and frameworks to the specifics of the project or the team. In most agile companies these practices can be adapted — and they are.

Part one covers the most widespread Scrum techniques, the standard framework you'll find here, on the internet and in other manuals: the rules for applying it, the roles, the events and the artifacts. In part two we'll explain how to take the training wheels off — very useful at first, but they'll hold you back in the long run — so you can keep moving forward.

PART I: THE SCRUM CYCLE

Learning the standard practices

The scrum cycle



As things stand today, the components of the standard Scrum cycle are:

- Scrum Team, made up of the following roles:
    - Developer.
    - Product Owner.
    - Scrum Master.
- Artifacts:
    - Product Backlog.
    - Sprint Backlog.
    - Increment.
- Events:
    - Sprint
    - Sprint Planning.
    - Daily Scrum.
    - Sprint Review.
    - 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 a working part of the product** (an increment). The recommended length for a sprint is between one and three weeks. Most commonly they're always the same length, setting a cadence, but that cadence can evolve or be adjusted.

Video: Scrum (→"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 functionally with everyone involved in the project. The length of the sprint is therefore the longest you can go without discovering that an approach was wrong, could be improved, or that a product feature was misunderstood.

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 sequentially, one after another, overlap in Scrum and move forward at the same time.

Self-management

Predictive management makes the project manager role responsible for managing the project and resolving its problems. 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.

Collaboration

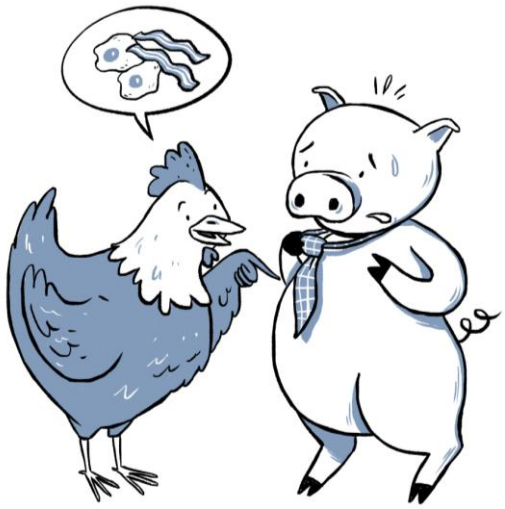
Every team member collaborates openly with the rest, according to what they can do rather than their role or job title.

Between self-management and collaboration, a team can handle perfectly well the work a project manager would otherwise be doing.

The committed and the involved

Plenty of people take part in a project and add value to it, though with different levels of commitment and responsibility. On that basis, a distinction is usually drawn between "the committed" and "the involved".

- **The committed:** those directly engaged in building the product or delivering the service.
- **The involved:** other stakeholders — management, executives, sales, marketing, the people who will operate the system being built, user support, and so on.



***

A chicken and a pig were walking down the street. The chicken said to the pig:
"Do you want to open a restaurant with me?"
The pig thought it over and replied:
"Sure, I'd like that. What would we call it?"
"Ham 'n' Eggs."
The pig stopped, paused, and answered:
"On second thought, I don't think I'll open a restaurant with you. I'd be really committed... whereas you'd only be involved."

***

In Scrum circles it's common to call the committed "pigs" and the involved "chickens" (with no pejorative intent whatsoever). The names come from this story, which illustrates the difference between being committed to a project and being involved in it.

One caveat. The fable was part of the early versions of the *Scrum Guide* and was removed in the 2011 edition, because in practice the labels were being used to shut stakeholders out — "management are just chickens" — which works against the very collaboration Scrum depends on. The distinction between commitment and involvement remains useful; the animals are better treated as a piece of Scrum's history than as current vocabulary.

Roles

Product Owner

The Product Owner is the person who makes the client's decisions. They are responsible for the value of the product.

To keep communication and decision-making simple, this role needs to sit with a single person. If the client is a large organization, or one with several departments, they can handle their internal communication however they see fit — but only one person joins the team. That person represents the client and must have the knowledge and the authority to make the decisions the role calls for.

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

- Developing and administering the Product Backlog.
- Presenting the vision and the user stories, and taking part in Sprint Planning for every sprint (→"Events").

The Product Owner owns the "Product Backlog" (→"Artifacts"). That means they have the final say on what the finished product will look like and the order in which increments are built; what goes in and what comes out; and how the "user stories" (→"Artifacts") are prioritized. They know the product plan, what it's capable of, the investment plan and the expected return on that investment. As the client's representative, they're also responsible for meeting the planned release dates for the product.

The Product Owner has expert knowledge of the client's business environment: they know what the client needs and what the project is meant to achieve. That's essential if they're to share that vision with the team and prioritize requirements. They need to stay on top of the business environment and analyze it continuously — how the market is moving, what competitors are doing, what alternatives exist — and combine that with what comes out of the team as the product takes shape: suggestions, technical alternatives, tests and evaluations of each increment. They need to know Scrum in order to do their job properly. Ideally, they'll also know the developers and have worked with them before. And the organization must respect their decisions, rather than changing priorities or Product Backlog items themselves.

Developer

Each of the professionals who build the "increment" (→"Artifacts") in every sprint. Although Scrum calls them "developers", that doesn't mean we're always talking about

programmers. It's an umbrella term for the people whose work contributes directly to developing — building — the increment.

Between three and ten people is the recommended range. Beyond that, direct communication becomes hard to sustain, and the friction typical of group dynamics — which starts to show from around six people — becomes more pronounced.

That threshold isn't arbitrary. 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 optimal reported satisfaction falling at around 4.6 members.

Developers are cross-functional professionals who work supportively, with shared responsibility. Individual developers may specialize in particular areas, but responsibility for the development rests equally with all of them. Beyond self-management and the use of agile technologies, their main responsibilities are the ones that mark the difference between a "working group" and a "team":

- **In a working group,** people have specific assigned responsibilities and follow a process or set of execution guidelines. Even though the group works in the same organization, each person answers for their own part individually.
- **A team has a collaborative spirit and a shared purpose:** getting the most possible value out of the client's vision. Developers manage themselves so that they work and answer for the results together. There is no manager to define, assign and coordinate the work; the developers coordinate it themselves. They all contribute and work with the Product Owner on the Product Backlog, take part in decisions, and respect each other's opinions and contributions. They share the goal of each sprint and the responsibility for reaching it. And they're all familiar with Scrum.

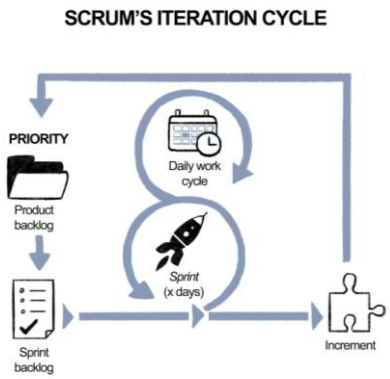
Scrum Master

The Scrum Master is responsible for making sure the rules of the Scrum framework are followed. They make sure the organization understands those rules and works in line with them. They advise and train the Product Owner and the developers, and they set up, shape and continuously improve the organization's agile practices. The aim is for the client and the development team to become capable of organizing themselves and working autonomously.

They're also responsible for facilitating the Daily Scrum, handling any group-dynamics difficulties that come up within the team, and clearing the impediments identified during the Daily Scrum so the sprint can keep moving.

Artifacts

Scrum's artifacts are its tools — its basic building blocks. They support the "roles" during the "events".



The most widely used artifacts

Three artifacts are key to how the standard framework works:

- **Product Backlog:** a prioritized list of the client's needs, expressed as user stories. At the start of the project it reflects the MVP (minimum viable product) or the initial vision. It's a living document: it grows and evolves as development goes on.
- **Sprint Backlog:** the list of work the developers are going to do during the sprint (→ "Work items").
- **Increment:** the result of each sprint. It's a part of the product that works properly; it must be tested and ready to use.

Other artifacts

- **Burndown chart:** shows the work remaining and the rate at which work items are being completed, so you can work out whether they'll all be finished in the estimated time. The developers update it daily.
- **Burnup chart:** where the burndown chart measures what's left, the burnup chart measures how much has been built or completed.
- **Definition of Ready (DoR):** an agreement defining when a Product Backlog item is "ready" to be pulled into a sprint.
- **Definition of Done (DoD):** an agreement on the criteria for considering a piece of work (a task, a story, and so on) finished.

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 a product or service should deliver to the end user, written from that user's point of view. Stories come

out of conversations between developers and the Product Owner aimed at reaching a shared understanding, and they serve as a reminder of what was discussed.
- **Technical story:** similar to a user story, but for functionality that isn't directly visible to the end user — refactoring, say, or implementing security measures.
- **Task:** a smaller work item than either of the above. Tasks are the activities that have to be completed in order to deliver a story: writing code, designing interfaces, running tests.

When a story is too big and too loosely defined, it's usually called an epic. Epics, like stories and tasks, are agile requirements, but their size and lack of detail make them unusable as work items. An example of an epic might be building a new ticket management system.

When a team starts working with Scrum, it's common to break stories down into tasks. That makes day-to-day progress visible, which can be particularly useful in teams with less experienced members — for example, to keep anyone from getting stuck.

As experience grows, though, breaking every story into tasks can become tedious, and can even induce a kind of paralysis. If the developers can manage their work during the sprint at story level, forcing a breakdown into tasks turns into unnecessary micromanagement. Which work items to use depends on the context.

The Product Backlog: the client's requirements

The Product Backlog is the inventory of features, improvements, technology and bug fixes that need to make their way into the product over successive sprints. It represents everything the client, the users and the other stakeholders are expecting. Anything that means work for the team should be reflected here. Backlog entries are most commonly referred to as user stories. A few examples:

- "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."
- "Offer file lookup through a web API."

The essential characteristic of this artifact is that it holds living, **continuously evolving information** — and that it's less a requirements document than a tool for getting information across to the team. At the start of the project the list contains only a handful of requirements: the ones known and best understood at that point, since it will expand and change as development progresses. It's precisely this dynamic quality that lets the product adapt to changing circumstances.

The backlog is usually put together after a meeting in which the client shares an overview of the business goal they're pursuing with the team. Once it has enough stories to run a first sprint, that's enough to get started.

From then on, the Product Owner keeps the stories in the backlog ordered by priority. How urgent an item is depends on how necessary and how valuable that piece of functionality is.

How detailed a user story is should be proportional to its priority. The highest-priority ones need to be detailed enough to be pulled into a sprint; agreeing a DoR can help here.

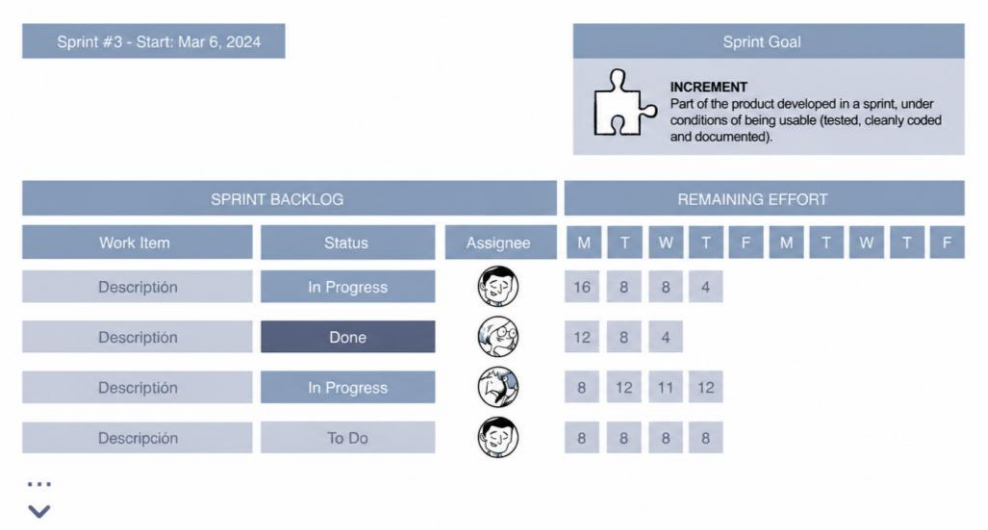
Prioritizing, detailing and pre-estimating stories is usually called refinement. It's collaborative work: the Product Owner and the developers can hold a refinement session at any point, as long as it never eats up more than 10% of the team's capacity. It's worth remembering that estimates — the likely effort for each Sprint Backlog item — are the developers' call, since they're the ones who will do the work (→"Agile measurement and estimation").

Product Backlog items that can be pulled into a sprint are described as ready. The term (or something like it) signals that the Product Owner and the development team agree the story is ready to be selected for a sprint. In other words: it's defined, pre-estimated, self-contained enough to fit within a single sprint, and the criteria for considering it finished — along with who will verify them — have been established.

The best way to build and maintain the Product Backlog is with simple means that the whole team knows and shares. It's a useful information radiator and a tool that supports direct communication. You might write the user stories on sticky notes on a board, ordered by priority, or use a management tool the whole team is comfortable with, something like Trello.

The purpose of the Product Backlog is to describe the state the product will be in later on — the shared vision the team plans against.

The Sprint Backlog



The Sprint Backlog is the list of all the work items (stories, tasks and so on) needed to complete the increment. In it, user stories are broken down into pieces small enough to track progress daily and to spot risks and problems without any complicated management process.

The whole team helps put this backlog together during Sprint Planning (→ "Events"), noting for each work item how much effort it's expected to take. Effort is usually estimated in "points" or "ideal time" (→"Agile measurement and estimation"), using techniques such as planning poker (→"Practices for making Scrum more flexible", "Planning poker").

During Sprint Planning the developers can, if they choose to, split stories into smaller tasks. If they do, 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' — they're the only ones who can change it during the sprint.

It also provides direct visual communication, and the development team uses it to review the sprint's progress every day. Ideally it lives on a board or a wall in the same physical space where the work happens, so that everyone can see it. Common formats include physical boards, shared spreadsheets, and collaborative project management tools such as Trello, Jira, Asana or Notion. Use whatever format suits everyone best, bearing in mind the following:

- It should include only the necessary information:
  - The Sprint Goal.
  - The list of stories or tasks.
  - Who has, for now, taken on each work item.
  - What state it's in, and how much is left to complete it.
- It should be the place where, at each Daily Scrum, the remaining "effort" on each work item is recorded.
- It should make daily, direct consultation and communication easy for the team.

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

Increment

The increment is the part of the product produced in a sprint that is fit to be handed over to the client: finished, tested and working. Prototypes, modules and anything still awaiting testing don't count as increments.

Ideally, in Scrum:

- Every Product Backlog item refers to deliverable functionality, not internal work of the "design the database" variety.
- One increment is produced per iteration/sprint.

The first sprint tends to be the exception, though. It's often called "sprint zero" when it has goals such as validating the platform and the design, which some projects need at the outset. These involve design work or building prototypes to test out the tools and working methods being considered.

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.

The point of the increment is to meet the quality standards the product requires. The Definition of Done must be known and shared by the whole team, and the increment isn't considered finished until it meets that definition.

Events

This section sets out the practices and activities that make up the working routine in Scrum.

Sprint: the fundamental unit of Scrum, the one everything else is organized around. It's sometimes also called an "iteration". It's the name given to each stage of work with a specific goal within the project. Dividing the work into sprints of fixed, constant length — usually referred to as "timeboxing" — is what keeps the rhythm of progress steady.

Sprint Planning: marks the start of every sprint. It determines the goal for that sprint and the work needed to achieve it.

Daily Scrum: a short daily meeting where the team takes stock, confirming that things are moving at the right pace or, if there's an impediment, spotting it and acting on it as soon as possible.

The standard format has each member report on what they did the day before, what they plan to do next, and whether they foresee any impediment.

Everyone updates the time or effort remaining on the work assigned to them in the Sprint Backlog, and that information is then used to update the chart the team uses to monitor the sprint's progress: the burndown chart.

Sprint Review: analysis and inspection of the increment produced, and adaptation of the Product Backlog if necessary.

Sprint Retrospective: a meeting at the end of the sprint in which the team examines the practical aspects of how it works and puts together a plan for improvements to apply in the next iteration.

One event that isn't part of the traditional Scrum framework but has been gaining ground is the **backlog refinement session**, already mentioned in connection with artifacts and estimating work items. There's no consensus on when to hold these; each situation ends up resolving it one way or another. They do seem to happen most often just before Sprint Planning. Whenever they take place, holding them means arriving at Sprint Planning with stories that the developers have already detailed and estimated.

Sprint

The key Scrum event for keeping progress continuous is the sprint: a fixed period of time, recommended not to exceed four weeks, during which a product increment is built.

As we saw under "Artifacts", the increment has to be done: working and useful to the client, in a state where it could be deployed or shipped.

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

- It sets the daily rhythm of progress and makes it visible and shareable at the stand-ups.
- It sets a fixed rhythm for checking how the product is developing, at Sprint Planning and Sprint Review.
- Retrospectives are slotted in at that same rhythm, for reflection and improvement.

In more mature Scrum implementations, however, the sprint can be treated as covering only the construction of the increment, leaving the meetings outside it.

That can be appealing if, for instance, you want more flexibility to run sprints of different lengths, or to decouple the frequency of retrospectives from the frequency of sprints.

Sprint Planning

This meeting marks the start of every sprint. Taking the client's priorities and business needs as a basis, it determines which features will go into the product by the end of the sprint, and how.

The meeting is run by the Scrum Master (or, if there isn't one, by a developer). The Product Owner and the developers must attend, and others involved in the project may also be present. It can take up to a full working day, depending on the volume and complexity of the Product Backlog items (user stories) you want to include in the next increment.

The meeting has to answer three questions:

1. Why is this sprint valuable?

The Product Owner sets out how the product's value can be increased by the outcome of the sprint about to start. This question determines the Sprint Goal.

2. What can be done in this sprint?

Once the Product Owner has shared the increase in value they're expecting, the developers decide which Product Backlog items they're going to take on. In the process, items that need more detail or explanation can be refined.

3. How will the selected work be done?

This part can be optional; it's recommended for less experienced teams. It can be useful for the developers to break each story in the sprint into tasks of a day's work or less. It helps build a better instinct for what different kinds of story involve, and makes day-to-day sprint progress easier to see.

The meeting is best structured in **two roughly equal halves, separated by a break:**

1. What will be delivered at the end of the sprint.
2. How the increment will be achieved, estimating the necessary requirements in story points or however the team prefers.

Preconditions	
The organization has determined and assigned the resources available to carry out the sprint. The highest-priority user stories in the Product Backlog are already "ready": detailed enough, with a preliminary estimate of the work they'll take. The developers know enough about the technologies in use and about the product business to make estimates and to follow the business concepts the Product Owner presents.	
Inputs	**Outputs**
Product Backlog. The product built in previous increments (except in sprint zero). Velocity or throughput in the last sprint, as a basis for estimating how much work to take on. The client's business circumstances, the technology landscape in use, and the value the Product Owner expects to obtain.	Sprint Backlog.

First half: why is this sprint valuable, and what can be done in it?

In this first half, the Product Owner presents the highest-priority user stories, explaining what's needed and what they expect can be built in the coming sprint. If the backlog has changed significantly since the last meeting, they explain why.

The aim is for everyone to understand, in enough detail, the increment the sprint is meant to produce. The presentation should be open to questions, and people can ask for clarification. Any developer can put forward suggestions, changes and alternative solutions, and amend the backlog accordingly.

This meeting is one of Scrum's hot spots for cross-pollinating ideas and adding value to the product vision.

Once the stories in the backlog have been reordered and reworked, the team defines the Sprint Goal: a single sentence capturing the value that's going to be delivered to the client. Except in sprints given over to an assorted collection of unrelated stories, working out this phrase together in the meeting is a guarantee that the whole team understands and shares the purpose of the work — and during the sprint it serves as a reference point for decisions.

Second half: how will the increment be achieved?

This second part should be treated as a team meeting, with all the developers present and doing the breaking down, estimating and assigning themselves. The Product Owner's role here is to field questions and check that the goal is understood and shared.

The team assembles the Sprint Backlog and breaks down any stories that need more detail into smaller pieces. It establishes which stories or tasks will be the priority for the first few days. Prioritization follows the order set by the Product Owner; once that's settled, the developers take on the work items that best fit their profile while respecting that order, aiming to spread the work evenly.

What the Scrum Master does during Sprint Planning

Where a Scrum Master has been assigned, they facilitate the meeting, making sure that:

1. The sprint starts with this meeting.
2. There's a Product Backlog prepared by the Product Owner.
3. The dialogue between the Product Owner and the developers keeps flowing.
4. The Product Owner and the developers reach agreement on what the increment will include.
5. The client's vision and business needs are understood.
6. The work has been broken down and estimated realistically, taking into account any analysis, research or support work that may be needed.
7. By the end of the meeting, the following have been objectively determined:
    - The Product Backlog items that will be worked on.
    - The Sprint Goal.
    - The Sprint Backlog, with every work item estimated, detailed and assigned.

The Daily Scrum

In Scrum, the team monitors how each sprint is going through very short daily meetings, colloquially known as **dailies**. They go by other names too: stand-up meeting, daily scrum, morning roll call. **Fifteen minutes is the recommended maximum**, and while these days they're often held remotely, in person it's best to hold them standing up. Either way, do it with access to a board showing how the sprint is progressing.

In those few minutes, the developers sync up their work and set the plan for the next 24 hours.

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

Meeting format

Hold it standing up, next to a board showing the Sprint Backlog and the burndown chart or a kanban board, so that everyone can share the information and make notes.

All the developers are present, and other people connected to the project or the organization may attend too, though they don't take part.

The developers run this meeting, not the Scrum Master. It isn't a check-up: it's for the team to collaborate and share where the work stands and what might be getting in the way. The Scrum Master's role is to facilitate, and to take whatever steps are needed after the meeting to clear those obstacles.

Commonly, each developer explains what they've achieved since the last Daily Scrum, what they'll be doing before the next one, and whether they're running into any problems or foresee any impediment. At the end, the development team refreshes the sprint

burndown chart (→"Practices for making Scrum more flexible", "Burndown chart") with the updated estimates.

That said, this individual-by-individual format has been observed to have negative effects on motivation and team dynamics. It can work as a starting point, but it's worth evolving towards a format that focuses on the work rather than on the people. That alternative format looks like this: the developers, who run the meeting, look at the state of progress. Seeing what's in flight, they take on the work still outstanding in priority order, making sure no bottlenecks form. Focusing on the stories or tasks rather than going person by person shortens the meetings, increases team autonomy and improves collaboration. You see it more often, for example, that several developers decide to work together on the same high-priority story to make sure it gets over the line if it's at risk of causing delays.

Sprint Review

A one- or two-hour meeting held at the end of the sprint to show the increment and gather suggestions. For larger or more complex increments it can run up to four hours. The whole Scrum Team attends, along with anyone interested in the project who wants to.

At regular intervals, this meeting sets the pace of construction and shows the direction the product vision is taking, giving stakeholders a chance to suggest how it should keep evolving. The developers present the user stories they planned, the ones they've built, the ones they haven't, and the impediments they ran into — important for keeping stakeholders' expectations realistic from the start of the meeting.

Inputs	Outputs
A finished increment.	Feedback for the Product Owner: a checkpoint on project progress and input for improving its value. The invitation to the next sprint's meeting.

Meeting format

This is an informal meeting in which the result of the iteration — the increment — is shown working. Depending on the nature of the project, user or technical documentation may be included as well.

Afterwards, using what came out of the meeting, the Product Owner will deal with any changes that arise.

Recommended running order:

1. The developers set out the Sprint Goal, the list of features that were included and the ones that have been built.
2. They give a general introduction to the sprint and demonstrate the parts they've built in action. The floor is then opened for questions and suggestions. This part generates valuable input for the Product Owner and the team as a whole to improve the product vision.

Sprint Retrospective

A meeting held after each Sprint Review, before the next sprint's planning. One to three hours is the recommended length. 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. Strengths and weak points are identified, so that the former can be reinforced and improvement actions planned for the latter.

Because it usually happens at the end of each sprint, it's sometimes mistakenly thought of as a "sprint review". The two are better handled separately, because their goals are different. The Sprint Review is about analyzing what is being built — the product — while a retrospective focuses on the way of working: the how.

The whole Scrum Team takes part in the Sprint Retrospective, including the Product Owner. It matters that the Product Owner sees themselves as part of the team rather than as the client, and that whoever holds the role takes part actively and knows Scrum's principles and values. If that isn't the case, the Scrum Master should act as facilitator to secure their commitment and participation. Although what's under discussion is the way of working, the Product Owner's perspective can add a great deal of value, since they're the one responsible for keeping the focus on building what the client needs.

Agile measurement and estimation

This section covers the basic concepts of agile measurement and estimation that are worth knowing.

That said: the fewer metrics, the better. The aim of Scrum is to produce as much value as possible, continuously, so it's always worth asking how a given indicator contributes to the value delivered to the client. Measuring is expensive, and it should serve a larger purpose rather than becoming an end in itself.

What we're after with the estimation tools we're about to look at is: being able to plan the length of each sprint realistically, setting the pace of progress (especially in teams just starting to work in an agile way), synchronizing teams, and committing to delivery dates.

Two key ideas:

- You don't measure the work done; you measure the work left.
- You measure using relative units.

Are we there yet?

There are two reasons you might need to measure work: to record what's already been done, or to estimate in advance what still has to be. Either way you need a unit and an objective basis for quantifying it.

Measuring work already done isn't difficult. You can do it in units relative to the product, such as completed stories, or relative to resources, such as cost or working time.

But agile project management doesn't measure work already done in order to calculate progress — that is, by subtracting it from the time budgeted. For example: "This was supposed to take a week. Three days have gone by, so there are four left until it's finished." Logical as that sounds, it rarely holds up in practice. Surprises come up, or shortcuts are found, and the time estimated at the start turns out not to be accurate. With that in mind, progress is determined not by the work done but by the work remaining.

Children understand this perfectly well. When you set off on a trip, what do they ask over and over? How long they've been in the car? No — that's not what they care about. The question is: how much longer?

Other processes in the organization may well need a record of effort spent, but calculating project progress is a different

matter.

Scrum measures remaining work for two purposes. First, to estimate the effort and time expected for particular tasks, user stories and epics. And second, to determine how far the project — and especially each sprint — has progressed.

Relative units: story points

The work needed to complete a requirement or a user story can't be predicted in absolute terms, because there's rarely a single solution. And even where there is, the metric would be too heavy and complex for agile management.

On top of the uncertainty about the work comes the uncertainty inherent in time itself. There's no sense in estimating the amount or quality of work an "average person" does per unit of time, because the differences from one person to the next are simply too great. What's more, the same task done by the same person will take different amounts of time depending on the circumstances.

For all these reasons, agile estimation prefers relative units. The most common work unit is the **point, or story point.**

Every organization institutionalizes its own metric of work — its own "point" — according to its circumstances and judgement. It's the relative size of task the organization tends to work with. What matters is that the meaning of the metric and the way it's applied stay consistent across the organization's measurements, and that everyone knows what it is.

The kind of "point" depends on the organization. For a development team, a point might be the equivalent of building a login screen; for a graphic design team, laying out a leaflet.

The "point" helps in two ways: it lets you size a story or task by comparing it with one you already know, and it brings out how difficult a particular piece of work is for each team member, given their specialisms. To illustrate the second, think about the effort of frying an egg. If you ask how many "fried eggs" it would cost to iron a shirt, the answer depends on the person. Someone might be very handy with a frying pan but hopeless at ironing, and estimate the shirt at "8 fried eggs" — that is, 8 points. Someone well used to housework, on the other hand, might put the same task at one point, or one fried egg.

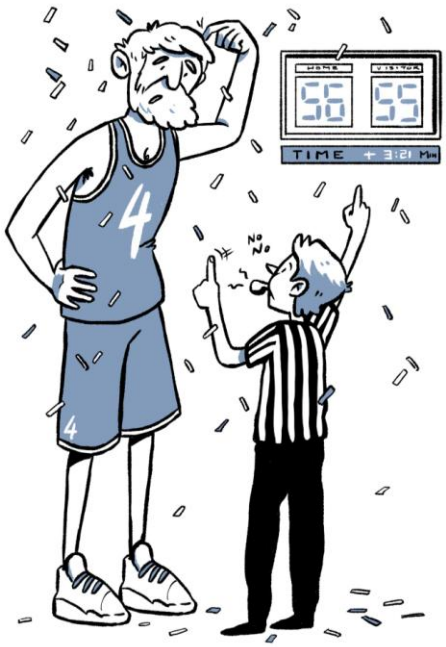
Both are right: the point is that the person doing the estimating should be the person who will do the work. Story points are usually a relative or abstract unit based on something the team knows inside out.

Finally, this lets you estimate velocity: in Scrum, velocity is the amount of work a team completes in a sprint. A velocity of 150 points, for instance, means the team gets through 150 points of work each sprint. Step outside the standard Scrum framework, though, and you may find sprints of differing lengths. When that happens, velocity can be expressed per unit of time rather than per sprint — "the team's average velocity is x points per week".

Measuring velocity with story points has drawn criticism and deserves careful thought, because misusing it can create unwanted dynamics (→"Practices for making Scrum more flexible", "Estimating without story points", "#NoEstimates").

Elapsed time and ideal time

When working out a sprint's schedule, we tend to estimate effort in ideal time: working time under ideal conditions. It's what a task would cost us in a state of flow — focused, with no distractions or obstacles. So it's important to be aware of the difference between ideal time and elapsed time when estimating.

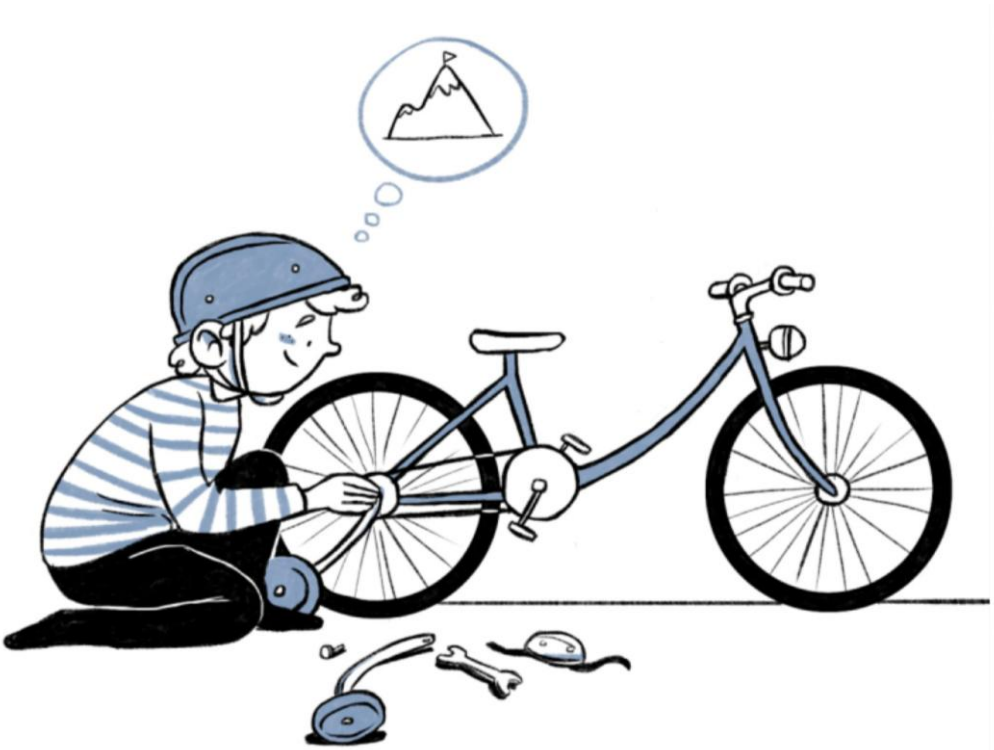


Ask how long a basketball game lasts and the answer, in exact playing time, is 40 minutes under international rules — 48 in the NBA — much as we might say that writing a report takes us an hour. But the elapsed time of a basketball game runs to well over twice that, since it can't end in a draw. It stretches out with time-outs, fouls, half-time and overtime.

We all know it's perfectly normal for a report — something that could be done in an hour of ideal time — to end up taking half a working day. Handling these two notions of time can help us organize work more objectively and avoid the stress of unreachable targets.

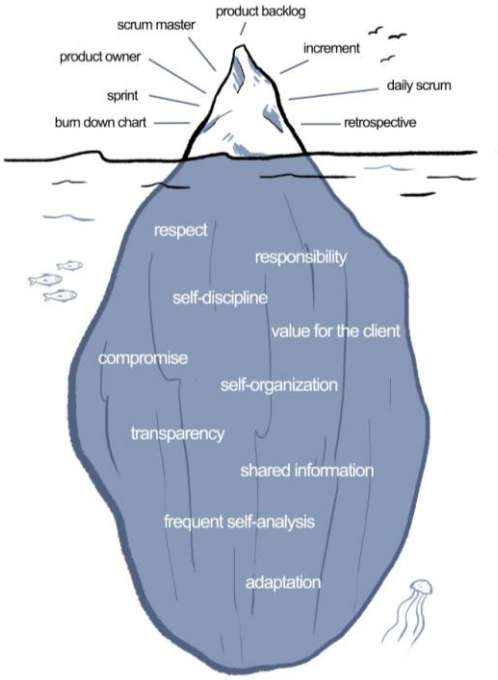
PART II: PRINCIPLES AND VALUES

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.



Why? Because Scrum's "practices" aren't "processes". They aren't 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 only 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 fall into the trap of focusing on the tools alone.

Management models make it easier to develop certain working principles and organizational values — and without those, we're working in a vacuum.

You'll find companies where the principles are so thoroughly internalized that they develop their own model of agility, with practices and techniques different from the ones covered here, borrowed from other agile models or entirely their own, 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 alienated, demotivated, stressed employees and mediocre results.

It takes flexibility to adapt what you've learned to the reality of each company and each project. The goal is for the organization as a whole to be agile — capable of advancing as a scrum in innovative, unstable working conditions.

Managing well means keeping up with the latest tools and then finding a way to fit them to the team: taking what works, changing or dropping what doesn't, rather than expecting people to adapt to methods they don't believe in.

***

Values and principles can be divided up — if only as a matter of semantics — according to whether they sit behind agile practices or behind the organization's culture. Principles underpin the practices; values underpin the culture.

Principles

The purpose of artifacts, events and other agile techniques is to ground the work in these principles.

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.

Continuous improvement

Agility means reflecting often on your working methods, questioning how effective they are and adapting them. That same self-critical effort is applied to improving the products and services on offer.

Iterative and incremental development

The finished 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.

Sustainable pace

Reaching a rhythm of work that avoids Parkinson's Law — work expands to fill the time available for its completion — and avoids the pressure that comes from discovering delays too late.

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.

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.

Cadence and global synchronization

This principle matters most when you're trying to synchronize several teams working on related products or services. The aim is to make the frequency of meetings and delivery dates predictable — by aligning the different teams' sprints, for example.

People over processes

The team's collective intelligence, its tacit knowledge, is what's responsible for the quality of the product.

Values

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 in collective intelligence and innovative value than in organizations whose values amplify what agile principles can do:

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

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

***

Our aim is to have all the tools needed to perform the Scrum Master's role. To that end, we'll now look at the principles associated with Scrum's roles, artifacts and events. We'll close with a few further practices that, while not part of the standard Scrum model, are commonly used.

If you'd like to study these agile principles and values in more depth, Scrum Manager® has produced a second manual, available on AgiLevel: https://agilevel.com.

People and their 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 the influence of certain cultural values (→AgiLevel), neither talent nor commitment can grow.

Ken Schwaber made this point bluntly in his 2006 talk Scrum Et Al.: a team of brilliant engineers with excellent tools, a deep grasp of the technology and the business, and no interruptions can of course use Scrum and build an increment every iteration. But Scrum works just as well with a team that has none of that — it will simply produce something poor every iteration. Either way you find out, at the end of each one, exactly where you stand. That is the whole point: Scrum makes reality visible. What it does not do is supply the talent (Schwaber 2006).

Product Owner

The Product Owner is responsible for the Product Backlog: they manage it and prioritize the requirements, taking a proactive stance and making changes whenever they judge it necessary.

The Product Owner is the person who makes early and continuous delivery of value possible. They represent the client and are responsible for communicating the client's vision clearly to the team, so that everyone lines up behind the same goal.

Their attitude is decisive in making communication honest and easy.

Developers

The development team is responsible for producing an increment with every iteration, and for keeping the pace of work sustainable. Committed to their work, they need to look critically at both their results and their methods in order to improve them. And they need to be conscious of how much their talent and their collective intelligence matter — not just their individual contribution.

Scrum Master

The Scrum Master is the person who makes sure the Scrum framework works: facilitating the Daily Scrum and managing the resolution of the impediments raised there, so that the team keeps moving.

Giving one Scrum Master responsibility for how Scrum runs makes most sense in teams with little experience, or in large organizations with a continuous flow of new and rotating staff. In small, stable teams with established levels of agility, responsibility for running and improving the Scrum framework may already have been internalized and taken on by the team collectively.

Artifacts

The artifacts help put the agile principles into practice.

Product Backlog

Value delivery: it makes it possible to absorb the variability of the client's business environment and concentrate effort on the stories that deliver the most value in the circumstances.

Iterative and incremental development: unlike a closed requirements document, the backlog makes iterative development possible because it's a living document, one that allows the stories it contains — and their priority — to change.

Operational visibility: it's an information radiator through which the Product Owner and the team share the product vision at all times.

Sprint Backlog

Iterative and incremental development: it's the artifact that bounds the work of a single increment (sprint) and sets the pulse of progress.

Operational visibility: it's an internal communication tool for the team, accessible to everyone, that shows the state of the sprint at a glance.

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 actually being generated.

Iterative and incremental development: delivering increments helps set the pace of progress.

Events

Sprint

Continuous improvement: progressing in short iterations makes it easier to identify points at which to stop and reflect on how to improve the quality of the product and of the working system.

Iterative and incremental development: it's the basic unit of time in which each increment is built, and therefore the gear the whole development turns on.

Sustainable pace: it sets the pulse of progress.

Operational visibility: it allows impediments to be spotted early.

Cadence and global synchronization: it sets the cadence of deliveries through timeboxing. It makes the frequency of meetings and delivery dates predictable, and allows different teams' work to be synchronized.

Sprint Planning

Value delivery: in this meeting the team and the Product Owner collaborate directly, deepening their shared understanding of the vision.

Daily Scrum

Operational visibility: in this meeting the team gets its bearings, shares where the work stands and collaborates, contributing ideas and clearing whatever impediments exist.

Sprint Review

Value delivery: in this event the team once again collaborates directly with the Product Owner.

Continuous improvement: the purpose of the meeting is to analyze the increment produced and draw conclusions that help shape the next sprint.

Sprint Retrospective

Continuous improvement: what's analyzed here isn't the increment but the working methods the team has used, so it can decide what to keep and what to change or drop.

Practices for making Scrum more flexible

The professional community contributes practices for shaping product backlogs and user stories, for representing the product vision visually, for running events and meetings, and for estimating work items. The two most widely used when first learning the Scrum framework are the burndown chart and planning poker. We'll look at those along with several others:

- **Tracking progress:**
  - Burndown chart.
  - Burnup chart.
  - Kanban.
- **Estimation:**
  - Planning poker.
  - Wall estimation.
  - Estimating without story points.
  - #NoEstimates.
- **Ways of working:**
  - Retrospective diagrams: fishbone and tree.
  - Mistake-proofing techniques.
  - Working in pairs.

For our purposes there's no need to go into how these practices work in depth. The point is to present them as examples and encourage you to investigate and experiment. They're only a few of the agile management techniques that can help you tailor the management model to a project or a team.

Tracking progress

Burn down chart

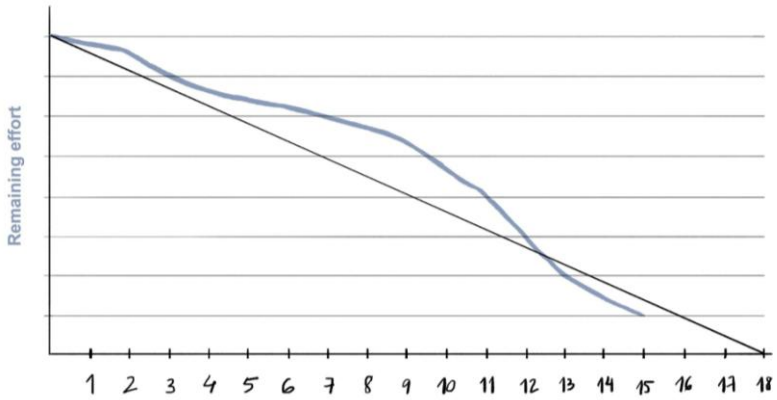
Developers update it during the sprint, ideally every day, to monitor the pace of progress. It's useful for catching deviations early — the kind that could jeopardize delivery at the end of the sprint.

- The Y axis shows the work points still outstanding.
- The X axis shows the days of the sprint.

Day by day, each developer estimates in the Sprint Backlog the effort remaining on each work item, until it's finished. That information is used to update the chart, which reflects the total effort outstanding each day.

When you're working with user stories rather than tasks (which, remember, should be a day or less), the burndown chart can be updated per completed story instead of by the estimated effort remaining on each story day by day. The story becomes the main measure of progress, and all the points estimated for a story are

subtracted in one go, when it's completed.La historia se convierte en la medida de avance principal, y se restan todos los puntos estimados para cada historia una sola vez, cuando ésta se completa.



The ideal progress of a sprint would be the diagonal that reduces outstanding effort continuously and gradually until the last planned day. As you'd expect, that isn't how it usually goes. You can keep a perfectly healthy pattern of progress without the diagonal being perfect.

If the progress line stays well above the diagonal for several days, it's a sign that the sprint was underestimated and will need more time. When the opposite happens and the line drops faster than the diagonal, the sprint will finish ahead of schedule.

Video: Burn down chart (→"References").

Burn up chart

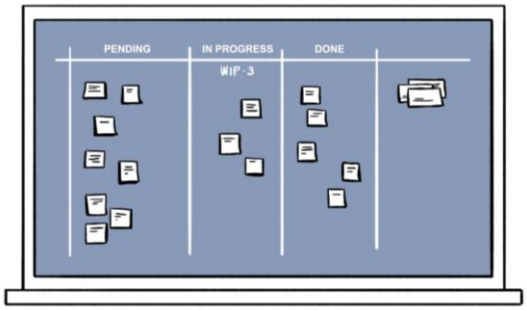
The burnup chart is a planning tool that belongs to the Product Owner. It represents the likely progression given the team's velocity. Three estimates are usually produced: pessimistic, realistic and optimistic.

The projection is plotted on a Cartesian diagram with the estimated effort to build the various Product Backlog stories on the vertical axis and time, measured in sprints, on the horizontal one. To plot the forecast, each release is placed on the vertical axis at the point corresponding to the estimated effort of building all the stories it contains.

Video: Burn up chart (→"References").

This is a tool for agile development — a living document that shouldn't be used to represent stable plans, but rather the forecasts that follow each evolution of the Product Backlog.

Kanban



Kanban is a popular technique for visually managing a continuous flow of work: a flow without timeboxing, without dividing the work into sprints of predetermined length.

The team writes down the stories or tasks and places them on a board. Where they sit determines their state. The most common columns on a kanban board are the ones that mark the order of progress: "to do", "in progress" and "done", laid out from left to right. The format of any given board depends on the circumstances of the product and the team; it might include additional states such as "tested" or "validated".

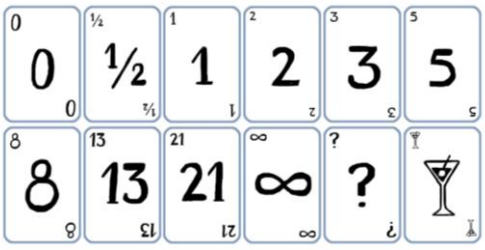
The board doesn't just help you manage work clearly and visually and keep up the pace — it's also a useful way of sharing information within the team. With every update it shows immediately what state each work item is in and whether bottlenecks are forming.

The absence of time-based milestones like sprints heads off Parkinson's Law. On the other hand, that same absence could cause delays, through perfectionism or procrastination. But the WIP limit, along with the structure and visibility the kanban board provides, mitigates that too. WIP (work in progress) is the maximum number of work items that can be in the same stage at the same time. A WIP of 3 on the "in progress" column means the team can't work on more than three stories or tasks simultaneously.

The visibility the board offers lets the whole team spot bottlenecks and idle time early, so priorities can be adjusted or rethought. What it reveals can prompt suggestions for improving flow and for how team members are distributed.

Estimation

Planning poker



James Grenning devised this planning game, which is used to run the meetings where the effort and duration of work items are estimated (Sprint Planning). Grenning's original set consists of eight cards, with the values ½, 1, 2, 3, 5, 6, 7 and infinity.

Each participant has a set of cards and, when estimating an item, shows the combination that adds up to the effort they've estimated.

The most widespread use, though, is with a deck of numbers following the Fibonacci sequence, as in the illustration. In that case numbers aren't combined: a single card is shown for each estimate, the one with the closest figure. This variant rests on the idea that as a task or story gets bigger, so does the margin of error.

So if someone thinks the right size for a story is 6, they're forced to reconsider. They can accept that some of the uncertainty they perceived isn't really there and play the 5, or accept a more conservative estimate and play the 8.

It's common to add a card with a question mark or some other symbol of doubt, to indicate that for whatever reason no estimate can be given. You can also include a card with an image suggesting the player needs a break. The infinity symbol comes out when the story exceeds the maximum estimable effort, to signal that it should be split into smaller items.

When estimates are wildly divergent, the facilitator has several options. They can ask the people at the extremes to explain their reasoning and then repeat the process. They can set the item aside for now and estimate it again later. They can ask the Product Owner to break it down so that each resulting sub-item can be assessed separately. Or they can decide to go with the most optimistic estimate, the most pessimistic one, or the average. It depends on the work being estimated and on each team's management style.

Beyond being fun and giving meetings some energy, planning poker heads off the circular arguments that tend to break out between competing implementation options. It gets everyone in the room participating, helps reach consensus without arguments, and cuts down the time spent estimating each piece of functionality.

Wall estimation

A technique used to estimate and prioritize a list of user stories, usually the Product Backlog. It relies on visual management: sticky notes with the various stories are placed on a wall. The developers set each note's horizontal position according to the size of the story — smaller ones to the left, larger ones to the right. The Product Owner sets the vertical position according to priority: the higher up, the higher the priority.

Video: Estimating on the wall (→"References").

Estimating without story points

Estimating with story points has drawn criticism because of how easily their purpose is misread. Using them to measure the velocity of several teams within a company, for instance, can create friction. Story points aren't a good measure of productivity: they're relative and depend on each team's context. A higher number of points doesn't imply more effort or more quality — but that isn't necessarily intuitive. Teams can start inflating estimates and pulling stories into the Sprint Backlog not on priority but to accumulate points, either to hold on to what they see as their average velocity or to compete with others. All of this works against the project and stifles collaboration.

There are alternatives that let you estimate while avoiding these problems:

- **T-shirt sizing:** an intuitive estimation system that keeps a degree of granularity, marking each story as XS, S, M, L or XL.
- **Goldilocks:** using only three sizes for stories — too big, too small, or just right.

Both require alternative measures for synchronizing the pace of work across teams and for identifying bottlenecks and other impediments in advance. But they allow intuitive prioritization and make it easy to spot stories that need more analysis before they go into production.

#NoEstimates

This is another reaction to the misuse of estimates in agile: disowning them altogether. In some cases — high-performing teams accustomed to working in an agile way and sustaining a continuous flow of work — not estimating can make sense. It helps guard against Parkinson's Law, for instance, which holds that work expands to fill the time available for its completion.

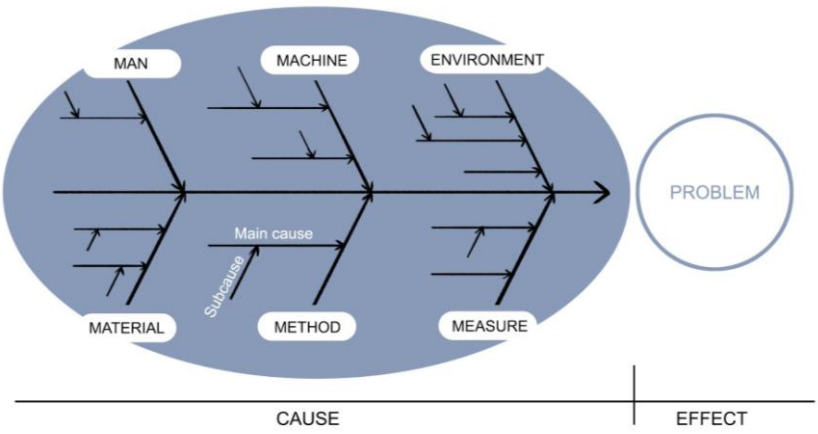
As with the alternatives to story points, this way of working means finding other ways to synchronize deliveries.

Ways of working

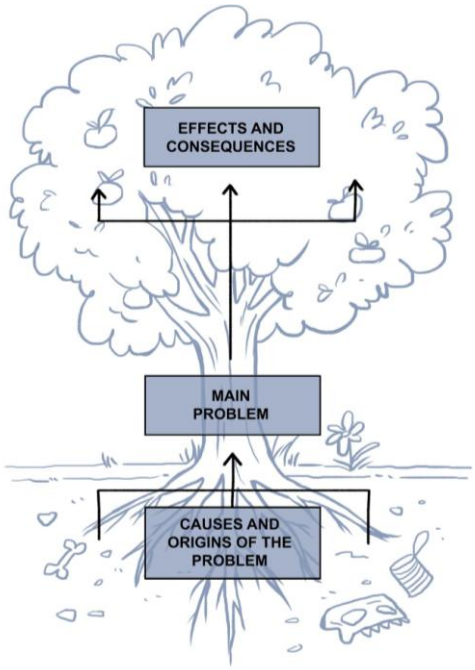
Retrospective diagrams: fishbone and tree

The **fishbone diagram** is also known as the Ishikawa diagram or the cause-and-effect diagram. It represents the cause-and-effect relationships at work on the problem being analyzed.

Once the problem has been identified — it forms the central spine of the diagram, a horizontal line through the middle — possible causes that might explain it are listed around it, and these can in turn have sub-causes. It's commonly used in quality management.



***



Tree diagrams tend to be used to reach positive conclusions from the analysis of a problematic situation. The parts of the tree (roots, trunk, branches, fruit) are used to represent the means or resources for solving a problem and the possible outcomes. There's no single type of tree diagram and no standard system, but as an example we suggest looking at how Khurram Bhatti develops it.

Mistake-proofing techniques

This means adopting ways of working that make errors impossible in the first place. In agile programming, test-driven development (TDD) is common: you write the tests the code has to pass first, and the code afterwards.

More broadly, agility makes use of poka-yoke techniques and andon control devices. Both concepts come from lean manufacturing.

Poka-yoke mistake-proofing techniques can be designed either to:

- **Make human error impossible.** Plug sockets designed so they can't be connected the wrong way round are one example.
- **Make the error obvious as soon as it happens.** This is what a word processor's spellchecker does, or a programming language's syntax checker.

Andon systems belong to the production process itself. They usually take the form of indicators — lights in different colors, or graphical displays — showing that the system is running normally or flagging possible faults. They sit in the working environment itself, alerting the team immediately and visually.

Working in pairs

This idea is most established in programming teams, where it's known as pair programming.

It means assigning two people to carry out the same task, simultaneously and jointly, usually alternating between doing the work and supervising it. While one carries

out the task, the other watches what they're doing and raises whatever observations are warranted. The practice can be a good fit when the quality of the result depends above all on the knowledge of the person doing it.

It's the same reasoning behind having two people in a train cab when the technology in use can't rule out human error entirely — or behind the reassurance of knowing there are two pilots working as a pair in an aircraft cockpit.

APPENDIX

Information, resources and professional community

Agile Club →

Skill Arena →

Community →

BoK/Wiki →

Observatory →

Comic strips →

About Scrum Manager® certification →

FAQs on Scrum Manager® courses and exams →

Social and professional networks

LinkedIn

Youtube

Continuous improvement and quality control

Scrum Manager's quality control is built on its students' feedback. Your opinion helps us maintain the standard of our materials, courses, centers and instructors.

If you've taken part in a training activity audited by Scrum Manager®, we'd be grateful if you would 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

References

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

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

Bhatti, Khurram (2019). "The tree retrospective model", Khurrambhatti.com: https://khurrambhatti.com/2019/03/08/tree-retrospective-model/

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

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

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

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, & 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.

Nonaka, Ikujiro, & Takeuchi, Hirotaka

- (1986). "The New New Product Development Game", *Harvard Business Review*, 64(1), 137–146.
- (1995). *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press.
- (2004). *Hitotsubashi on Knowledge Management*. 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.

Palacio, Juan (2019). "Estimaciones ágiles, métricas de productividad, #NoEstimates y otros animales" [Agile estimates, productivity metrics, #NoEstimates and other creatures], Scrum Manager Blog, in Spanish: https://www.scrummanager.com/blog/2019/12/estimaciones-agiles-metricas-de-productividad-noestimates-y-otros-animales/

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

Schwaber, Ken

- (1995). "SCRUM Development Process", *OOPSLA '95 Workshop on Business Object Design and Implementation*. Springer.
- (2006). *Scrum Et Al.* Google Tech Talks.

Schwaber, Ken, & Sutherland, Jeff (2020). *The Scrum Guide: The Definitive Guide to Scrum. The Rules of the Game*: https://scrumguides.org

Turner, Richard, & 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.

Scrum Manager Youtube,

- (2017). *Burn up chart*
- (2017). *Purpose of Agile vs. Predictive Management*
- (2018). *Managing plans and ideas*
- (2021). *Burn down chart*
- (2021). *Estimating on the wall*
- (2021). *Scrum (en)*