Agile Bedrock

Agile Bedrock

Scrum Manager

Version date: July 2026
Author: the Scrum Manager team.
Publisher: Uncovering Better Ways SLU

Table of contents

守破離 — Shu Ha Ri

Learning any skill goes through three stages:

- **Shu:** a technique is chosen and, assuming it is correct, one tries to imitate it.

Ha: more techniques are collected.

Ri: one experiments and invents new techniques by mixing, combining and modifying.

Shu-stage techniques are generally applicable. Ri-stage techniques only work in specific cases, and require expert knowledge to know when and how to apply them.

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

Preface

This text forms part of the core body of knowledge of Scrum Manager®. It is new content incorporated into the training and certification framework to fill the gap that current developments have made necessary: the emergence of intelligent assistance is changing the value-production equation in knowledge companies, and with it, the criteria by which a professional decides how to work.
The speed at which the knowledge spiral turns today — the succession of thesis, antithesis, and synthesis described in the introduction — makes it essential to know the frame of reference from which each professional can compose their own synthesis. That is what this guide offers: not a closed model subject to obsolescence with every turn, but the foundations and professional criteria that enable adaptation as the context changes. It is, in the literal sense, the firm substrate — the bedrock — on which each professional builds their development and evolution.
The document serves a dual purpose. It is a guide for teams and organizations that need to design their way of working in this new landscape, starting from empiricism and adapting to their context: whether or not they develop software, whether they use generative assistance or autonomous agents, whether they need an iterative or a continuous cadence. And it is, at the same time, the reference syllabus for professionals who wish to accredit their up-to-date knowledge through the core certification specific to this body of knowledge.

Scrum Manager Certification

It does not prescribe a single framework. It offers an inventory of knowledge, true to Scrum Manager®'s agnostic stance, so that each professional can develop their own judgment. It is aimed at those who perform, or aspire to perform, the role of facilitation and improvement in knowledge teams; the name of that role matters less than its purpose: enabling the team to deliver value empirically, sustainably, and transparently.
When discussing agility, many anglicisms and terms are used that take on a specific meaning within management. These words are marked with « » quotation marks to make it easier to search for information and to indicate that they carry a special connotation.

Introduction

The knowledge spiral

It is worth starting with the frame that gives meaning to everything that follows. Knowledge does not advance in a straight line, nor by accumulation, but by tracing a spiral. Nonaka and Takeuchi, the researchers who gave scrum its name, formulated it drawing on a dialectical idea: knowledge progresses through thesis, antithesis and synthesis.

A **thesis** is a valid answer to a situation, built with the knowledge and the circumstances of its moment. It works, it spreads, and it becomes the established way of doing things. But over time new circumstances appear that expose its limits and contradictions: this is the **antithesis**, which is not an error of the thesis, but the signal that the context has changed. From the confrontation between the two comes neither the victory of one nor the simple sum of both, but a **synthesis:** a new understanding that reconciles what remained valid in the thesis with what the antithesis has revealed.

> *It is at this point that the earlier thesis and antithesis are reconciled and transcended. In time, however, even the synthesis will prove one-sided in some respect. It will then serve as the thesis for a new dialectical movement, and so the process continues, zigzagging in a spiral.*
>
> *— Nonaka, 2004*

That is the key: the synthesis is not a point of arrival, but the start of the next turn. Each synthesis becomes the thesis of a new cycle, which will meet its own antithesis and give rise to a new synthesis. Knowledge advances this way, in a spiral, without stopping.

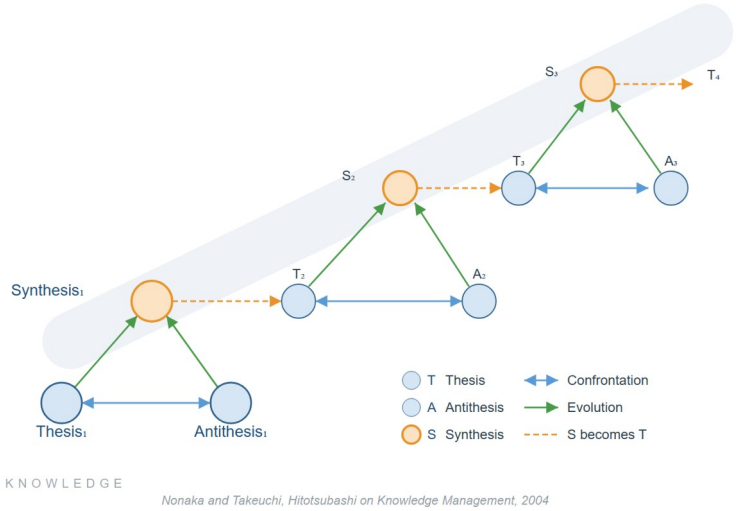

Figure 1. The knowledge spiral of Nonaka and Takeuchi. The synthesis reconciles thesis and antithesis, and becomes the thesis of the next turn.

This manual stands at a specific point of that spiral. Agile management was, in its day, a synthesis. Intelligent assistance is the antithesis that now puts it to the test. And what this manual gathers is not a closed doctrine, but **the synthesis that is taking shape right now:** the reconciliation, still under way, between the agile body of knowledge that remains valid and the new circumstances that artificial intelligence has introduced. Whoever studies it should read it with that awareness: they are not learning a final destination, but the current state of an evolution that will continue. The introduction that follows serves to know where we come from; the rest of the manual, to understand where the next thesis is taking shape.

Agility as a response to complexity

Agile management emerged as the antithesis of an earlier model, which will be referred to frequently: predictive project management. Both have their virtues and prove more useful in different industries. The predictive approach focuses on planning, calculating a budget and setting deadlines. If the project is finished on the agreed date, without exceeding cost and with all the features of the initial plan, it is considered a success.

That strategy works in stable environments, with products that result from scrupulous attention to processes and protocols. **Predictive management** is a child of the Industrial Revolution: it comes from construction, from the automotive industry, from the factory. If what the customer wants is a house, it must be built solid, safe, fit for the needs of those who will live in it and, ideally, on time and on budget.

But much of what is produced today does not resemble a house. First, because it can be abstract: a film, a mobile app, a marketing campaign, a legal report. Things can be tried out during development, checked empirically to see what works, adjusted along the way, starting from a minimal sketch and growing from there. The scenario changes, and a feature that seemed essential at the start may be obsolete by the delivery date.

Competing in these industries requires the ability to respond quickly in uncertain scenarios, without stable starting requirements, with customers who need to start using the product as soon as possible and improve it continuously. These are the **knowledge companies:** those that develop products or services based on knowledge rather than on tools and processes. Their environment bears very little resemblance to the one that gave rise to predictive management.

Predictive and evolutionary management

Since the nineteen-eighties, so many models and practices for managing projects have accumulated that the landscape can be overwhelming. A map that puts them in order helps. All of project management can be understood by crossing three concepts with two management models: the concepts are development, work and knowledge; the models, predictive and evolutionary management.

1. Development

The development of a project can be complete or incremental.

- **In complete development**, the description of what is to be obtained is available and detailed at the start and serves as the basis for estimating. With the initial plan, schedule and resources are laid out, and during execution what is managed is compliance with the plan.

In incremental developments, that description is not complete at the start: it is complemented and evolves during development. It can be managed with two different tactics: **continuous increment**, which achieves a continuous flow of delivery as parts are completed; and **iterative increment**, which keeps production at a fixed rate through "timeboxing", and which is the mode of the standard scrum framework, where each iteration is called a *sprint*.

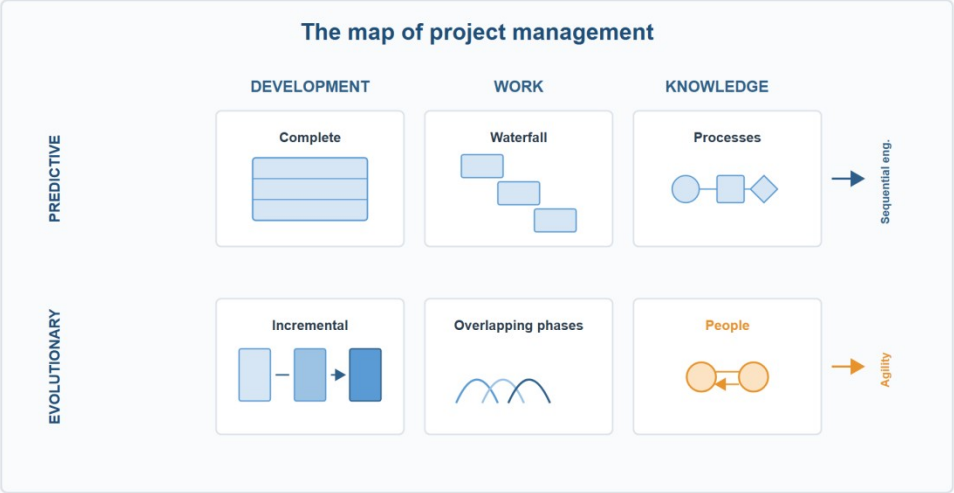

Figure 2. The map of project management. The accent marks the agile end: knowledge residing in people.

It is worth retaining this double tactic of incremental development —iterative and continuous—, because it will be one of the hinges of the whole manual.

2. Work

The way of working can be sequential ("waterfall"), chaining phases one after another, or concurrent, overlapping them in time.

3. Knowledge

Knowledge can reside in processes —explicit knowledge, where quality depends on process and technology— or in people —tacit knowledge, where quality depends on the experience, motivation and talent of those who do the work.

> *Tacit knowledge is personal, context-specific, and therefore hard to formalize and communicate. Explicit or "codified" knowledge, on the other hand, is knowledge that is transmittable in formal, systematic language.*
>
> *— Nonaka, 1995*

Sequential engineering. The combination of complete development, waterfall work and knowledge residing in processes gives rise to sequential engineering, also called predictive management. Its goal is to deliver predictable results: to develop the planned product without exceeding schedule or resources.

Evolutionary management. The combination of incremental development, overlapping phases and knowledge residing in people gives rise to evolutionary management, whose goal is to deliver a minimum viable product early and increase its value continuously. It can be carried out with process-based production —concurrent engineering— or with people-based production —agility—. The distinction matters, because without it, agility gets confused with the mere application of scrum rules or the mere use of a *kanban* board.

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

Whoever focuses on following the process, even if they use iterations and boards, is doing concurrent engineering; agility demands trust in the team's capabilities and real self-management. This difference, subtle in appearance, will be central throughout the manual.

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, seventeen software professionals met in Salt Lake City. They had something in common: they were critical of process-based production models, which they considered too heavy and rigid. From that meeting came the term "agile methods" and the four statements of the Manifesto, together with twelve principles that develop them.

Individuals and interactions over processes and tools. This is the most important statement. Processes help and good tools improve efficiency, but there are tasks that require talent and motivated people to provide it. In process-based production, quality is meant to be a consequence of the process; in agile development, processes are only a support to guide the work.

The working product over comprehensive documentation. Necessary documentation is useful, but its relevance must be lower than that of the product. A closed, exhaustive requirements document written before starting is usually a waste of time.

Customer collaboration over contract negotiation. The goal of an agile project is not to control execution so that the initial plan is fulfilled, but to continuously deliver the greatest possible value.

Responding to change over following a plan. The values of agile management are anticipation and adaptation, as opposed to the planning and control of orthodox management.

The twelve principles that accompany these statements will be the touchstone when we reach the antithesis. In particular: early and continuous delivery of value, welcoming change, delivering frequently in short periods, building around motivated individuals, direct conversation as the most effective form of communication, working software as the measure of progress, sustainable pace, technical excellence, simplicity, self-organizing teams and periodic reflection to adjust behaviour.

The origin of scrum

Scrum is an agile development model characterized by autonomous, self-managed teams that share knowledge and learn together; by an incremental development strategy instead of complete planning; by basing quality on people's tacit knowledge and creativity, not on processes; and by overlapping the phases of development instead of chaining them in a waterfall.

The word comes from rugby, where "scrum" names the formation in which both teams push together for the ball. The term leapt into management in the Japan of the nineteen-eighties, when Nonaka and Takeuchi studied the manufacturing companies that achieved the best results in innovation and time to market (Nonaka 1986) and compared their way of working in self-managed teams to the advance of rugby players moving in formation.

From 1995 onward, Ken Schwaber presented at the OOPSLA conference a software development methodology based on a scrum environment (Schwaber 1995). There is no authority that determines what is scrum and what is not: the framework has changed and will keep changing with the contributions of the community. The original spirit remains: practices should help teams self-manage and keep a continuous flow of progress, producing results iteratively and frequently. Scrum Manager® uses the term in that original sense, not as a closed rulebook of ceremonies.

The traditional scrum framework in brief

For more than two decades, the scrum framework established itself as the reference standard of iterative agile development. It was, in terms of the spiral, an achieved synthesis: it reconciled the need to deliver early with the need to keep empirical control of progress. It is worth knowing, not to imitate it blindly, but to understand which problems it solved and which of them remain relevant now that this synthesis has become the thesis that intelligent assistance puts to the test.

Traditional roles. The *product owner*, the single person responsible for maximizing value; the *development team*, the self-managed, cross-functional professionals who deliver the increment; and the *scrum master*, facilitator of the framework, responsible for scrum being understood and applied correctly.

Traditional artifacts. The product backlog, a prioritized, emergent list of everything the product needs; the sprint backlog, the subset selected for a sprint together with the plan to deliver it; and the increment, the sum of the finished items on top of the value of the previous ones.

Traditional events. The sprint, a fixed-length iteration that contains the other events; planning, which decides what will be done and how; the daily scrum, a brief synchronization; the review, which inspects the increment; and the retrospective, which reflects on the process.

This framework worked, and still works, in thousands of teams. But it was designed in a context where people were the only actors in the production of value. Intelligent assistance forces the question of whether these roles, artifacts and events are still the most efficient form of empiricism, or whether in some contexts they have become a ritual that simulates control while reality flows through other channels.

The fourth wave: intelligent assistance as antithesis

Since 2022, the irruption of generative artificial intelligence —and since 2024, the rise of autonomous assistance agents— has altered the relative speed between human decision and assisted execution. It is not a passing technological fashion, but a change in the value-production equation that forces a re-reading of the agile framework from its roots. In the terms of the knowledge spiral, it is the antithesis that puts to the test the thesis that scrum had come to be.

The question is not whether intelligent assistance will change work: it is already doing so. The question is whether that change will erode empiricism —the ability to inspect and adapt based on reality— or whether assistance will be used to strengthen it.

In this manual, intelligent assistance is understood as a **spectrum**, not as a switch. At one end, **generative assistance:** tools that propose drafts, summaries, code or content suggestions, while the person decides, edits and validates. At the other, **autonomous agents:** systems that execute complete tasks —from writing code to drafting documents— with minimal human intervention in the execution. A marketing team that generates campaign drafts is at one pole; a software team that delegates the writing of microservices to an agent is at the other. The working framework must be able to contain both without losing its essence.

This antithesis has an effect that should be understood before going any further, because it organizes all the analysis that follows: intelligent assistance acts as an **amplifier**.

Assistance amplifies, it does not fix

Artificial intelligence is not, by itself, either the salvation or the ruin of agile management: it magnifies the conditions that already exist in the team. Where there are good practices, a culture of quality and solid foundations, assistance reinforces the workflow and accelerates the delivery of value. Where there are fragile processes, technical debt and broken coordination, assistance exposes and aggravates those problems, which now show up harder and faster. The same tool produces opposite results depending on the team's starting condition.

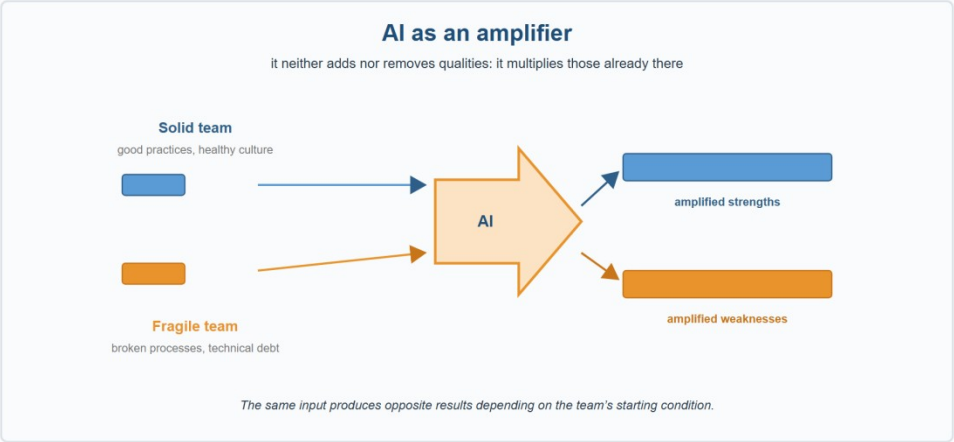

Figure 3. Assistance amplifies the starting conditions: it empowers solid teams and aggravates the weaknesses of fragile ones.

The mechanism is understandable. Assistance makes it possible to generate work much faster, which tends to produce larger batches, harder to review and more prone to introducing defects, unless there are control systems that contain that volume: automated tests, disciplined review, fast feedback loops. Without that base, speed does not improve the result: it accumulates problems at a higher rate. Assistance, in short, does not fix teams; it amplifies them.

This observation carries a reassuring reading regarding the foundations, and it is the reason why this manual places them in its first part. If assistance amplifies what already exists, then the mechanisms that make it possible to see early what is happening and correct in time —transparency, frequent inspection, adaptation— do not lose value with artificial intelligence: they gain it. Empiricism is not a victim of the antithesis; it is precisely what distinguishes a team that harnesses the amplification from one that suffers it.

The bottleneck shifts

There is a second, more conceptual effect that explains why assistance transforms teams. In all knowledge work two different things coexist: deciding what needs to be done and executing what has been decided. Throughout the entire history of agile management, the bottleneck was almost always in execution: writing the code, drafting the document or producing the piece took human time and effort, and so work was organized around that effort, estimated and measured by it.

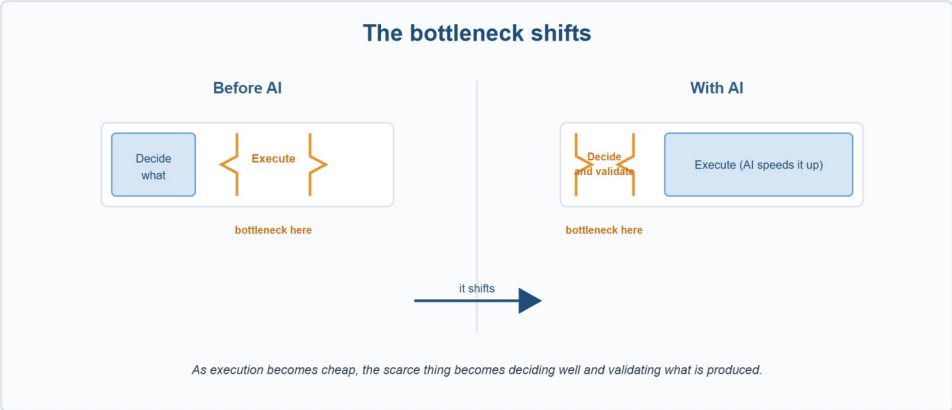

Figure 4. As execution becomes cheap, the narrow point of work moves from producing to deciding and validating.

This shift explains a transformation already visible in the teams that have adopted the most assistance: they tend to become smaller. Where a large team with specialized roles used to be needed, a small group capable of owning the vision, the design, the construction and the validation begins to suffice, supported by tools that execute the mechanical part. And since in knowledge work the real brake was never the number of hands but the speed of decision, smaller teams decide sooner and with less bureaucracy. The consequence for the framework is uncomfortable but clear: the machinery that existed to coordinate the execution of many people loses weight, while everything devoted to deciding well and validating gains it.

What the antithesis challenges and what it consolidates

Before entering the synthesis, it helps to separate clearly what the antithesis acts upon. Because intelligent assistance does not affect the whole framework equally: it challenges certain specific ways of working, above all in flow environments, and at the same time it consolidates —when it does not amplify— the foundations that give agility its meaning. This separation is, in itself, the first step of the synthesis.

What the antithesis challenges are specific forms, not principles, and it does so more strongly in environments that lean toward continuous flow than in iterative ones; most markedly in the assisted lane of software development. The fixed-length iteration as the only option stops being self-evident when part of the work is produced in minutes. The full battery of ceremonies always held in the same way loses meaning when the team is tiny or when part of the work is executed by agents. The story point as the universal unit of measure does not fit the work of a machine. The large team with highly specialized roles becomes unnecessary wherever execution becomes cheap. None of these forms

disappears —they remain valid in many contexts, above all non-software—, but none can be taken for granted for everyone anymore.

What the antithesis consolidates, and often amplifies, is everything that was never a ceremony but a foundation, and that the synthesis must keep preserving: early and continuous delivery of value; transparency about the real state of the work; frequent inspection of the result and of the way of working; fast adaptation to what is discovered; and the centrality of human judgement, which decides what is valuable, validates quality and takes responsibility. Above all, the first statement of the Manifesto, which assistance does not repeal but makes more literal than ever: individuals and interactions over processes and tools. Assistance is the most powerful tool that has ever entered the field, and for that very reason it reminds us that a tool, by itself, contributes no tacit knowledge and takes no responsibility.

The synthesis this manual develops consists, precisely, in rebuilding the way of working on what is consolidated, letting go without nostalgia of what the antithesis has made obsolete in each context. The first two parts establish those foundations that hold; parts III to V rebuild, on top of them, the architectures, the practices and the decision criteria.

Part I — Foundations of empiricism

The roots that do not change, though the branches mutate

The first two parts of this manual develop what the antithesis consolidates: the foundations on which any synthesis has to rest. They come before any rhythm, practice or tool, and they outlive them all.

Empiricism: transparency, inspection, adaptation

Empiricism is the non-negotiable base of any agile development. It is not a technique, but an attitude toward reality: it asserts that valid knowledge comes from experience and from observing what happens, not from theoretical planning.

Scrum describes itself as an empirical framework because it rests on three pillars:

- **Transparency.** The significant aspects of the work must be visible to those who answer for the result. Visibility covers what is being done, the problems that arise and the decisions that are made. Without transparency there is no possible inspection.
- **Inspection.** Those who take part in the work must frequently inspect the artifacts and the progress toward the goal, to detect undesired deviations. Inspection is not police-style control, but a critical and constructive look.

Adaptation. If something is detected drifting beyond acceptable limits, it must be adjusted as soon as possible to minimize larger deviations.

These three pillars are not negotiable, and they are indifferent to the rhythm of work. It does not matter whether the team works in two-week iterations, in continuous flow or with agents that deploy several times a day. If there is no transparency about what is being built, frequent inspection of whether it generates value and fast adaptation when it does not, there is no agility: there is fast chaos or slow bureaucracy. The amplification that intelligent assistance introduces makes these pillars more necessary, not less, because the faster things are produced, the sooner one must be able to see whether what is produced is of use.

Value as the north star

The goal of agile management is not to deliver faster, but to deliver **more value, earlier**. Speed is a means; value, the end. This distinction, which was always important, becomes critical in an environment where assistance can multiply production: producing a lot is not the same as delivering value, and sometimes it only adds volume, or even noise, to the product.

Value means different things in different contexts. For a software customer, a feature that solves a real problem. For a legal department, a reviewed contract that reduces risk without delaying the operation. For a marketing team, a campaign that connects with the audience at the right moment.

Predictive management measures success by compliance with the plan: whether it arrived on time, whether the budget was not exceeded, whether what was agreed was delivered. Agile management

measures it by impact: whether something changed for the better, whether something previously unknown was learned, whether the team is able to adapt. Keeping value as the north star is what prevents speed from becoming an end in itself.

People, interaction and tacit knowledge

The Manifesto places individuals and interactions above processes and tools. This is not romanticism, but an economic observation: in knowledge work, the quality of the result depends more on people's tacit knowledge than on the quality of processes.

A process can indicate how to draft a contract, but only a person with legal experience and knowledge of the client knows whether that contract truly protects the interests of the business. An assistance agent can generate the draft in seconds; the decision on whether that draft is adequate remains human. This is the line that assistance does not cross: it executes and proposes, but it contributes no tacit knowledge and takes no responsibility.

Direct interactions remain the richest way to communicate complex information, because of their nuance, their tone and their immediate feedback. With distributed teams and digital assistance they are not always possible, but the principle remains: seek the richest interaction the context allows. And self-managed teams remain the heart of agility, not because they need no direction, but because that direction must come from knowledge of the work and not from hierarchy. When those who do not do the work decide how it must be done, the capacity for local adaptation is lost.

Intelligent assistance: spectrum and principles of integration

Intelligent assistance is not just another tool, as spreadsheets or digital boards were in their day. Those helped manage the work; assistance can perform part of the work. That is why it should be integrated with judgement, understanding it as the spectrum already introduced: from generative assistance, which proposes so that the person decides, to autonomous agents, which execute complete tasks under supervision.

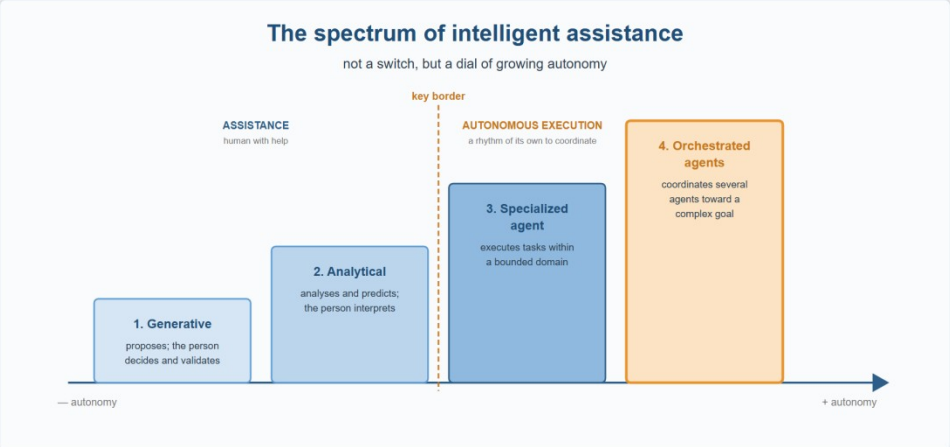

Figure 5. The spectrum of intelligent assistance. The border between the second and third levels —from assistance to autonomous execution— is the one that most conditions the way of working.

Whatever the point on the spectrum, its integration into an agile team is governed by four principles:

- **Empiricism rules over speed.** Assistance is adopted if it strengthens the capacity to inspect and adapt, not because it accelerates. Speed without empiricism is only faster chaos.
- **Responsibility is indivisible.** If an agent generates work that fails, the responsibility lies with the human team that validated it. Assistance cannot be the scapegoat.
- **Transparency extends to assistance.** The team must know which decisions and which outputs were human and which were assisted. If the provenance of a decision cannot be inspected, there is no empiricism about it.
- **Human interaction remains the axis.** Assistance does not replace the conversation between those who define value and those who build it; it replaces transcription, repetition and mechanical execution.

  The violinist and the bow. *A bow capable of playing perfect notes at unreachable speed does not turn just anyone into a musician: the music still belongs to the violinist. If the violinist stops listening, feeling and deciding, what comes out is fast noise, not music. Assistance is the bow; empiricism, the ear that judges.*

Part II — Functions of the agile team

Who does what, and why

In the era of intelligent assistance, titles matter less than functions. An agile team needs to cover four essential functions, which one or several people may exercise and which in some cases may be supported by autonomous systems. The final responsibility for each function, in any case, is human.

The shift of the bottleneck toward decision, described in the introduction, runs through this whole part: the functions that decide and validate gain weight, those that executed lose it, and a new function appears that the traditional framework did not contemplate.

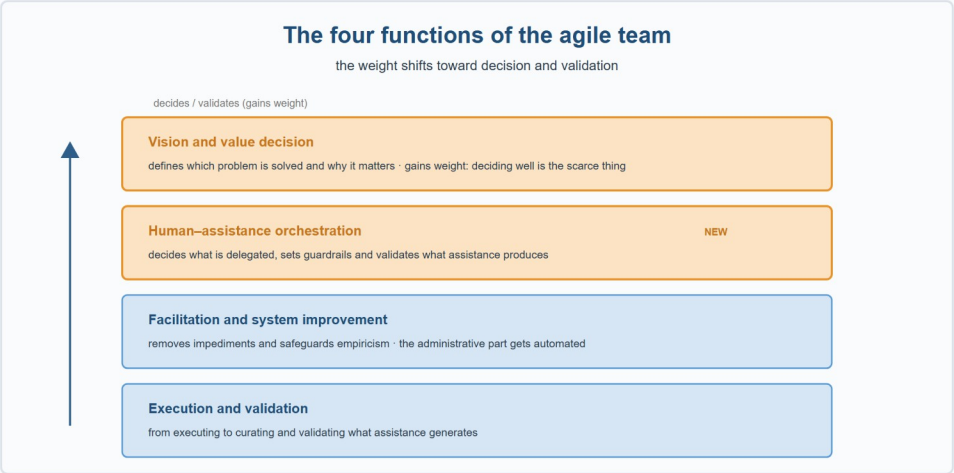

Figure 6. The four functions of the agile team. In amber, those that gain weight with the shift of the bottleneck, including the orchestration function, which is new.

The vision and value-decision function

Purpose

Define which problem is being solved, for whom and why it matters. Maximize the value delivered to the customer and the business.

It is the function traditionally exercised by the product owner. But it is not a title, it is a responsibility: in a legal team it may fall to the senior lawyer who understands the business risk; in marketing, to whoever knows the customer; in software, to whoever understands the market.

Key responsibilities

- Maintain a needs inventory that is prioritized, transparent and understandable for the whole team.
- Make prioritization decisions based on value, risk and dependencies.
- Communicate the vision so that the team can make aligned tactical decisions.

Validate that what is delivered solves the stated problem, not merely that it “works”.

With intelligent assistance

Assistance can analyse usage data, customer feedback and market trends to inform prioritization, and generate drafts of stories or of a roadmap. But the decision of which problem to solve remains human, because it involves business judgement, risk tolerance and strategic vision that the machine does not possess. This is the function most strengthened by the shift of the bottleneck: when executing becomes cheap, deciding well what to build becomes the most valuable and scarcest activity. The profile evolves from manager of a list to strategist of value.

The execution and validation function

Purpose

Build the product or service increment and ensure that it meets the quality and completion criteria.

It is the function traditionally exercised by the development team. But “developing” is not just programming: it is creating, and that covers drafting, designing, analysing, coding, testing and verifying.

Key responsibilities

- Select from the needs inventory the work that can be tackled in the next cycle or stretch of flow.
- Break down, estimate and self-assign the work.
- Build the increment with the agreed quality.
- Validate that the increment meets the completion criteria before delivering it.

With intelligent assistance

In software, agents can generate code, tests, documentation or infrastructure, and people become **curators and validators:** they review, understand, integrate and guarantee. In non-software teams, assistance accelerates the production of drafts, analyses or designs, and people validate the professional judgement, not just the form. Self-assignment and estimation remain human, because they depend on the team’s tacit knowledge of its own capacity and context. Technical knowledge, far from being surplus, becomes more necessary: more judgement than ever is needed to validate what a machine produces at speed.

The facilitation and system-improvement function

Purpose

Get the team to work without friction, remove impediments and continuously improve the system of work.

It is the function traditionally exercised by the scrum master. But it is not a guardian of rites; it is a facilitator of empiricism and an optimizer of the flow of value.

Key responsibilities

- Facilitate the team's synchronizations so that they are efficient and productive.
- Identify and remove the impediments that slow the flow of value.
- Protect the team from external interruptions that break the rhythm.
- Foster a culture of continuous improvement, where the system of work is inspected and adapted.
- Teach agile principles rather than impose practices.

With intelligent assistance

Assistance can automate the collection of metrics, the generation of meeting summaries or the detection of blocking patterns. But the facilitation of conflicts, the accompaniment of people and the decision to change the system of work remain human: they require empathy, intuition and judgement. The facilitator must also watch that assistance does not turn meetings into a formality: they should happen because the team needs them, not because a tool schedules them.

The human–assistance orchestration function

Purpose

Decide which work is delegated to assistance, which is reserved for people and how what assistance produces is validated.

This function is new. It did not exist in traditional scrum because there were no systems that executed work. Today it is indispensable in any team that uses assistance beyond the spell-checker, and the shift of the bottleneck has made it, in many software teams, the most critical of the four: when the scarce thing is no longer writing but validating, whoever orchestrates that validation occupies the centre of the system.

Key responsibilities

- Define the **guardrails** of assistance: what it may do on its own, what requires approval, what is forbidden.
- Maintain **algorithmic transparency:** the team knows which decisions or outputs come from assisted systems.

- Periodically validate the quality of what assistance produces, especially when the underlying models change, a phenomenon known as **model drift** that can silently degrade results that used to be reliable.
- Manage the **assisted lane** in dual-channel teams, ensuring that the machine's speed does not disconnect the human.

Who exercises it

It does not necessarily require a dedicated person. In small teams it falls to the facilitator or to a technical person of reference; in mature software teams it may be a specialized role; in non-software teams it is usually taken on by the senior professional who uses the assistance and knows its limits. What it does not admit is being left without an owner: when nobody orchestrates, assistance stops being a tool of the team and starts setting the pace on its own.

Part III — Rhythm architectures

How the team keeps time

On the foundations of the previous parts, the concrete way of working is now rebuilt. The first configurable element is the rhythm. Choosing it is not an aesthetic preference, but a decision that depends on the team's context, the nature of the work and the level of assistance in use. Remember that the two tactics of evolutionary management —iterative and continuous— are both legitimate: the question is not which is more agile, but which fits each case better.

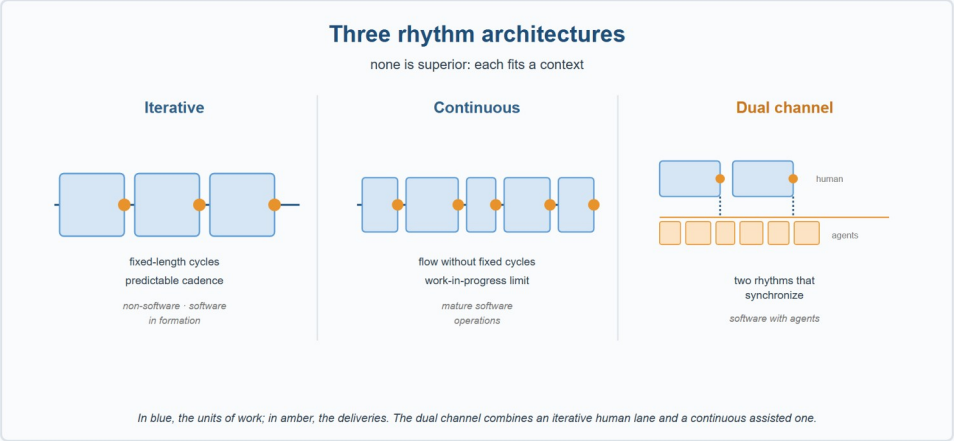

Figure 7. The three rhythm architectures. The dual channel, highlighted, combines an iterative human lane with a continuous assisted lane.

Iterative rhythm

The iterative rhythm divides the work into fixed-length cycles —iterations or sprints—, at the end of which a tangible result is produced and a full synchronization of the team takes place.

Characteristics

- **Fixed time (time-boxing).** Time is fixed; scope is negotiable. If everything cannot be finished, scope is reduced, time is not extended.
- **Full synchronization.** At the end of each cycle the team meets to review the increment, receive feedback and adapt the plan.
- **Predictable cadence.** External stakeholders can trust that there will be a review moment every certain number of weeks.

When it is appropriate

- Teams learning to work together that need the discipline of limits.
- Environments where requirements change frequently and a regular inspection point is advisable.

- Non-software teams where coordination with external departments —legal, customers, regulators— demands predictable milestones.
- The **human lane** of dual-channel teams.

The iterative rhythm is the mode where the traditional framework described in the introduction remains valid almost in its entirety. Assistance enters here mostly as support within each cycle, without altering its logic.

Continuous rhythm

The continuous rhythm does not divide the work into cycles. The flow is constant: a unit of work enters, is processed, validated and delivered. There is no delivery “at the end of the sprint”; there is delivery when it is ready.

Characteristics

- **Pull system.** Work is taken from the system when there is capacity, instead of being pushed into it.
- **Work-in-progress (WIP) limits.** How many things can be in progress at once is limited explicitly. This avoids multitasking and exposes bottlenecks.
- **Quality gates.** Each stage of the flow has clear entry and exit criteria. Nothing advances without meeting them.
- **Flow metrics.** Cycle time, lead time, throughput, defect rate.

When it is appropriate

- Homogeneous, predictable work, where each unit resembles the previous one.
- Mature teams, with high technical discipline and continuous delivery capability.
- The **assisted lane** of dual-channel teams, where agents operate without constant human synchronization.
- Support, maintenance or operations teams, where work arrives unpredictably.

Control is non-negotiable. Here the lesson of the amplifier is critical. If assistance magnifies volume and the main risk is instability, an uncontrolled continuous flow becomes dangerous: much produced, little reviewed, defects accumulated. That is why the continuous rhythm is only safe on a base of mature practices —strict work-in-progress limits, automated tests, continuous integration, disciplined review, quality gates—. The work-in-progress limit stops being a visual management technique and becomes the mechanism that prevents speed from overwhelming the capacity for review.

Dual channel

The dual channel (“dual track”) is an architecture in which two modes of work coexist and synchronize at specific points, not at every step. It is the answer that best preserves the value of the framework when people and agents that execute work autonomously coexist in the same team, because their natural rhythms are incompatible and forcing them into the same beat degrades both.

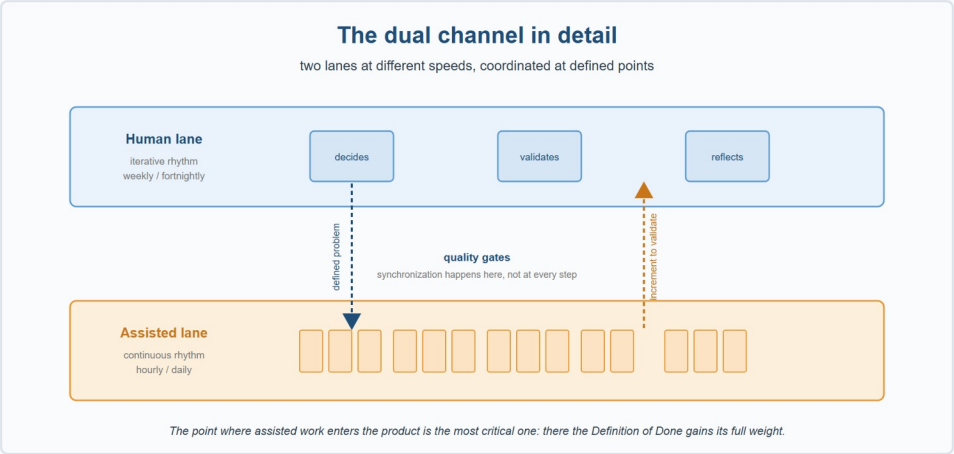

Figure 8. The dual channel in detail. The human lane feeds the assisted one with defined problems and validates what it returns, at the quality gates.

Synchronization points

The human lane feeds the assisted lane with well-defined problems and clear acceptance criteria. The assisted lane returns finished increments to the human for validation. Synchronization happens at the quality gates, not at every step. That validation point —where the work of the assisted lane enters the product— is the most critical place in the whole system, and it is where the Definition of Done acquires a weight it did not have before.

Risks of the dual channel

- **Disconnection.** The assisted lane goes so fast that the human stops validating, and technical or content debt accumulates invisibly.
- **Empty formality.** The human lane holds formal meetings while the assisted one already delivered days ago.
- **Loss of learning.** If assistance solves everything, the human team stops learning and becomes dependent.

The fine-grained management of the dual channel is still territory for experimentation, not settled practice, and should be approached in the spirit of *ri*: expert knowledge to know when and how to apply it.

Transitions and choosing a rhythm

No rhythm is intrinsically superior. The choice depends on an honest diagnosis of the team and the organization.

Factor	Iterative	Continuous	Dual channel
Type of work	Complex, variable, creative	Homogeneous, predictable	Mixed
Technical maturity	Medium	High	Very high
Level of assistance	Low to medium	Medium to high	Very high (agents)
External coordination	Frequent, predictable	Minimal	Defined points
Team size	3–9 people	Flexible	2+ humans, N agents
Typical sector	Non-software, immature software	Mature software, operations	Software with agents

How to transition

- **From iterative to continuous:** requires technical maturity, test automation and a culture of quality. It cannot be skipped.
- **From continuous to iterative:** may be necessary when the team grows, the work becomes more complex or external coordination demands milestones.
- **Toward the dual channel:** only when the team is clear about the orchestration and validation functions.

The rhythm must serve empiricism, not the other way round. *If the chosen rhythm prevents the team from inspecting and adapting, it is the wrong rhythm, however modern it may look.*

Part IV — Working practices

The configurable pieces of the system

The practices that follow are the components with which each team assembles its own concrete system. They are not obligations, but pieces that fulfil a function; what changes from one rhythm to another is their form, not their purpose.

The needs inventory

The needs inventory —the product backlog, in the traditional vocabulary— is the prioritized source of everything that adds value. It is a living document: it reflects what is known at each moment and evolves as learning happens.

Its purpose is twofold: to order the work according to value and to serve as a tool for communicating the vision to the whole team. It is maintained by the vision and value-decision function, and its essential characteristic is prioritization: the most valuable and urgent, first.

With intelligent assistance. Assistance can analyse feedback, usage data or trends to suggest needs, and generate drafts of stories or specifications. But the decision to include something in the inventory, and its priority, remains human: assistance informs, it does not decide. The inventory should make visible, when relevant, whether a need was identified by a person or proposed by a system. This is not excessive zeal, but empiricism: if the team cannot inspect the source of an idea, it cannot adapt its selection criteria.

The work board

The board is the visual representation of work in progress. Its purpose is to make the state of the flow transparent, identify bottlenecks and facilitate self-organization.

In iterative rhythm it is usually a board with columns —to do, in progress, in review, done— where each item is a unit of work of the current cycle and developers self-assign by moving cards. **In continuous rhythm** it is more complex: it includes all the stages of the flow with explicit work-in-progress limits in each column; if a column reaches its limit, no more work is pushed into it; instead, the team helps empty it.

With intelligent assistance. Agents can update the board according to their state, and it is advisable to include a human-validation column as a mandatory quality gate. The orchestration function must be able to see at a glance how much work is executed by people and how much by assistance. A board that "lies" —because assistance updates states without the human knowing— leaves the team blind exactly where it most needs to see.

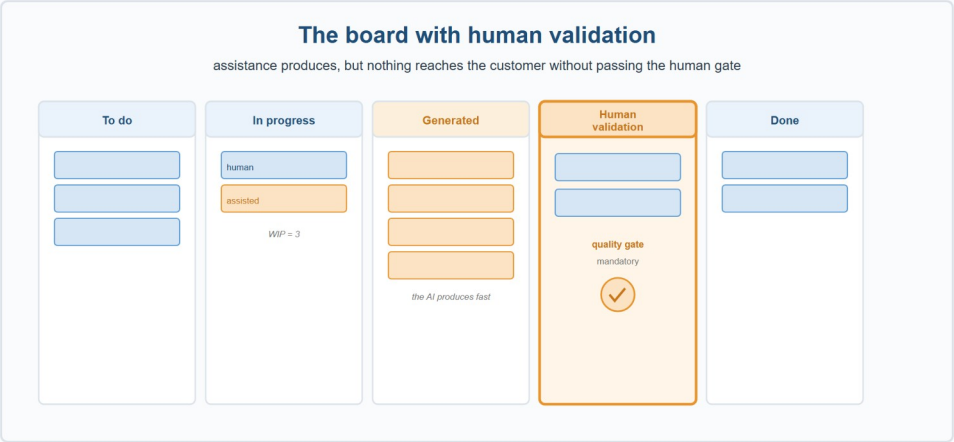

Figure 9. A board adapted to assistance: the AI piles up work in the "Generated" column, but nothing moves to "Done" without passing through human validation.

Increments and completion criteria

An increment is a portion of value that is finished, tested and ready for the customer to use. It is not "eighty per cent done": it is "done".

The Definition of Done is an agreement of the team, not an external imposition. It must be clear, shared and verifiable. In software it usually includes code written, tests passed, peer review, documentation updated and deployment. In non-software it may include a reviewed draft, customer approval, regulatory compliance and publication.

With intelligent assistance, this practice gains a weight it did not have before. When assistance produces more and faster, the Definition of Done becomes the barrier that separates "a lot has been generated" from "something that adds value has been delivered". It is a decisive distinction: greater volume is not greater value, and an increment generated without validation may not only fail to add value, but add noise to the product —code nobody understands, content that dilutes the message, documentation that gets in the way—.

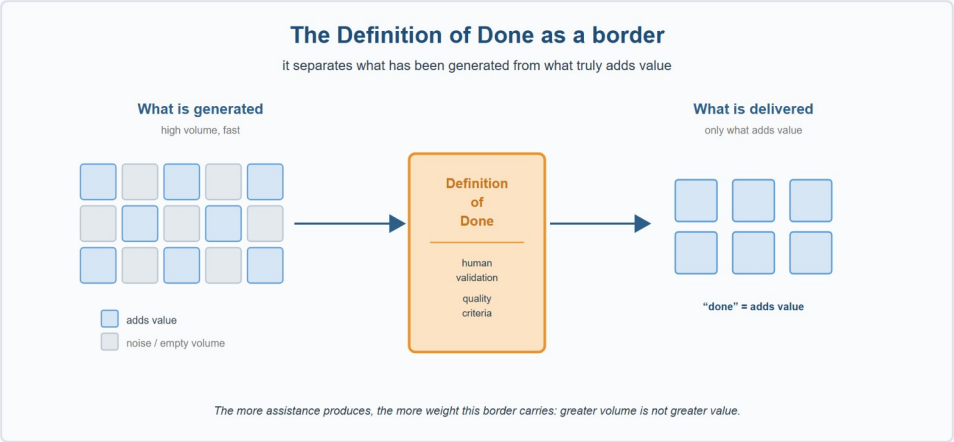

Figure 10. The Definition of Done acts as a border: it filters the generated volume and only lets through what truly adds value.

That is why the Definition of Done must explicitly include the **human validation** of any generated or assisted work: if an agent wrote the code, a person must review it; if an agent drafted the document, a person must verify the facts and the tone. Assistance can automate the tests, but it cannot sign off completion. The signature is human.

Team synchronizations

Synchronizations are meetings with an empirical purpose: align, inspect, adapt. They are not mandatory rituals; if a synchronization adds no value, it is modified or removed. Their form changes with the rhythm.

Planning. In iterative rhythm, at the start of each cycle: it decides what is tackled and how, in a conversation between the vision function and the execution function, and it consumes no more than ten per cent of the cycle's duration. In continuous rhythm there is no cycle planning, but continuous refinement of the inventory: when a unit is ready to enter the flow, there is a brief discussion of who takes it. In dual channel, the human lane plans which problems will feed the assisted lane, which needs no planning but clear specifications.

Daily synchronization. In iterative rhythm, a brief meeting —fifteen minutes at most— that is not a status report but a tactical alignment: the team inspects the board and adjusts the plan for the coming hours. In continuous rhythm it can be less frequent, or a simple visual review of the board: the rule is to synchronize when there is deviation, not by the clock. In dual channel, the human lane synchronizes and the assisted one is monitored, with the orchestration function reviewing the agents' logs. In all cases it is better for the meeting to focus on the work, not on the people: an evolution the agile community was already recommending before assistance, and which assistance accelerates.

Review of the result. In iterative rhythm, at the end of the cycle: the team shows the increment to the stakeholders and receives feedback, in a conversation about the product. In continuous rhythm there is no cycle review; each increment is reviewed as soon as it is ready, and feedback is continuous. In dual channel, the assisted lane delivers continuously to the human, who validates before taking anything to the customer.

Reflection on the system. In iterative rhythm, the retrospective at the end of the cycle: the team reflects on how it worked, not on what it built. In continuous rhythm, reflection is periodic or triggered by an anomaly: if the flow breaks, one stops and reflects. In modes with intense assistance, this reflection incorporates a new object: **the debugging of the working system with assistance itself.** When an agent produces something useless, the improvement does not consist in correcting a person, but in reviewing and rewriting the instructions, the controls and the criteria with which the tool operates. The retrospective thus widens its scope: no longer just the human dynamics, but also the configuration of the automated work, and the uncomfortable question of whether assistance is being used to avoid difficult conversations or whether human validation has become a mere formality.

Estimation and prediction

Agile estimation does not seek absolute precision, but **shared understanding.** When the team estimates, what is valuable is not the number that comes out, but the conversation it generates about complexity, dependencies and risks.

Relative units

Story points are a relative measure of effort, complexity and uncertainty; each team defines its own scale and they are not comparable across teams. **T-shirt sizes (XS, S, M, L, XL)** are useful when numerical precision generates sterile debates. **"Goldilocks" sizing** —too big, too small, just right— helps detect units that should be broken down or grouped.

No estimation (#NoEstimates). In mature teams with continuous flow, estimation may become unnecessary: if the work is homogeneous and the cycle time is stable, prediction is done by flow statistics, not by case-by-case estimation. It is not a fashion, but a consequence of maturity; a team that cannot yet predict by flow still needs to estimate in order to learn.

With intelligent assistance. Systems can predict effort from historical data, but that is a data point, not a verdict. Human estimation remains valuable because it forces conversation, aligns expectations and uncovers hidden risks. The facilitator must prevent assistance from cutting that conversation short: if an algorithm says "eight points" and the team accepts without discussion, the learning has been lost, and learning was the only thing estimation was after. And when the work is executed by an agent, the story point loses its meaning —it was conceived to measure the relative effort of a person—, so other units are used, such as computational cost, and above all metrics of quality and stability.

Metrics and radiators

Metrics must serve empiricism, not hierarchical control, nor competition between teams, nor empty formality.

In iterative rhythm: velocity (points completed per cycle) serves to predict, not to pressure; if it is used to compare teams or demand more, it becomes corrupted. The **burndown chart** shows the remaining work and detects early deviations; the **burnup chart** shows the work accumulated toward a larger goal.

In continuous rhythm: lead time measures from the moment a need enters until it is delivered; **cycle time**, from the moment a unit is activated until it finishes; **throughput**, the units completed per unit of time; and the **defect rate**, the proportion of work that needs redoing.

Information radiators. Visible charts, updated by the teams themselves, showing the real state. In assisted teams they should include the provenance of the work —human, assisted or mixed—, in line with algorithmic transparency. The most important metric is not how much is produced, but how much of what is produced generates value for the customer. Measuring is costly: every metric must justify itself by answering how it helps inspect and adapt. If the answer is "so that management sees that work is being done", it is not an agile metric, but an instrument of control.

Part V — Context and decision

How to design your team's system of work

This part does not prescribe. It offers a method for the professional to diagnose their context and make informed decisions on how to combine everything above. And it begins with the distinction that most conditions any decision: whether the team develops software or not.

The structural asymmetry: software and non-software

This is not one factor among several. It is the axis that organizes the entire design of the system of work, because it determines how much and in what way intelligent assistance affects the team. Treating it as a detail leads to applying to one team recommendations designed for the other, which is the most common mistake of this era.

In software development, the disruption is at its highest, for a concrete reason: the product coincides with what assistance is most competent at. Artificial intelligence writes code, tests it, refactors it and documents it; agents can chain those tasks with autonomy. That is why it is in software where the extreme cases appear —tiny teams that multiply their delivery capacity, cycles that go from weeks to hours— and where the pressure on the iterative framework is most intense. It is the natural terrain of the continuous rhythm, of the dual channel and of the orchestration function.

In the rest of knowledge work, the impact is real but of a different nature. In marketing, legal affairs, human resources, finance or consulting, assistance enters as a powerful tool that automates specific tasks —drafting documents, reviewing texts, analysing data—, but it rarely challenges the framework. The reasons are structural: coordination with clients, courts, regulators or management imposes rhythms that cannot be compressed at will; the product is not executable end to end by an agent; and professional judgement remains at the centre of every delivery. For these teams, the iterative rhythm remains a valid starting point, and assistance is integrated within it without overflowing it.

The practical consequence is that **there is no single answer**, and that the first step of any diagnosis is to place oneself honestly on this axis. What follows should always be read through this distinction.

Diagnosing the team

Before choosing rhythm, practices or tools, five questions should be answered honestly.

- **What type of work do we do?** If it is complex, variable and creative, it leans toward the iterative rhythm. If it is homogeneous, predictable and repeatable, it leans toward the continuous one. If it is mixed —part creative, part mechanical—, it leans toward the dual channel.
- **What sector are we in?** This is the question of the asymmetry above. In software, assistance has reached a level that challenges fixed time-boxes and allows agents to operate in continuous flow, even though validation, value decisions and external coordination remain iterative. In non-software, assistance is integrated as a tool within a framework that remains iterative.

- **What is our technical maturity?** Is there test automation, continuous integration, continuous deployment — the requirements for continuous flow? Is technical or process debt low, knowing that assistance amplifies weaknesses? Does the team know how to measure and act on flow metrics?
- **What is our organizational culture?** Does management trust teams to self-organize, without which no rhythm works? Are metrics used to improve or to control? Is there willingness to experiment and to fail?
- **What level of assistance do we use?** Generative, with minor impact on the rhythm? Analytical, with medium impact? Autonomous agents, with major impact, pushing toward the dual channel or continuous flow?

The honest diagnosis matters more than the sophisticated answer. An organization that adopts assistance without clear guardrails usually finds that speed increases but stability and customer satisfaction do not: it is, once again, the amplifier at work. If the team is fragile, assistance makes it more fragile, faster.

Crossing the two most decisive questions —the type of product and the level of assistance— yields a map that orients without constraining:

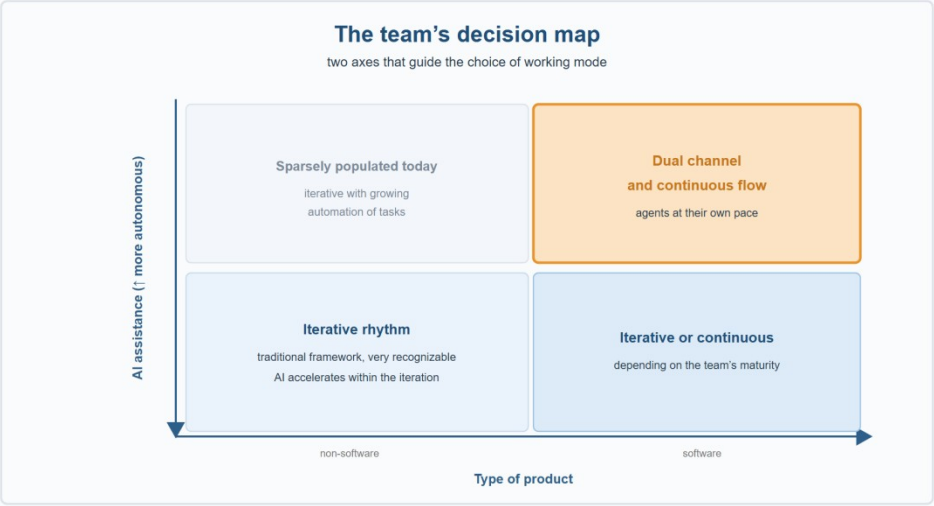

Figure 11. The decision map. The accent marks the quadrant where the change is deepest: software with autonomous agents.

Most non-software teams sit today in the lower half, and for them the iterative rhythm remains an excellent starting point. Most software teams are moving to the right and upward, and for them the choice of working mode is a living decision. The map assigns no fixed destinations: a team can occupy different positions depending on the project, and move over time.

Choosing a rhythm architecture

Based on the diagnosis, the team can opt for one of these architectures.

A. Pure iterative rhythm. Cycles of one to three weeks with planning, daily synchronization, review and reflection. Appropriate for non-software teams, software teams in formation, teams with high external coordination or with generative assistance. It brings discipline, predictability and alignment; its risk is rigidity and empty formality if meetings become perfunctory.

B. Pure continuous rhythm. Flow with work-in-progress limits, without sprints, with delivery when something is ready. Appropriate for mature software teams, support or operations teams, or homogeneous work. It brings flexibility, lower coordination cost and real speed; its risk is the lack of periodic reflection, procrastination without fixed time-boxes and the loss of strategic alignment.

C. Dual channel. An iterative human lane —planning, validation, reflection— plus a continuous assisted lane —execution, testing, deployment—. Appropriate for software teams with autonomous agents, research with massive simulations or content with assisted generation. It maximizes at once the speed of assistance and human judgement; its risk is disconnection between lanes, empty formality in the human lane and the loss of validation.

D. Adaptive hybrid. The team alternates between iterative and continuous depending on the phase of the product or project: iterative in discovery and design, continuous in scaling and maintenance. Appropriate for teams in transition or organizations with several products in different phases. It brings maximum flexibility; its risk is confusion, lack of consistency and the cost of switching modes.

There is no correct architecture, only one adequate to the current context. And that context changes, so the choice should be revisited periodically, not consecrated.

Integrating assistance: levels and guardrails

Assistance is not switched on all at once; it is integrated by levels, each with its own guardrails. The principle that runs through them is one: the greater the autonomy of the assistance, the greater the rigour of the human guardrails must be. Trust is not blind, but calibrated, verifiable and reversible.

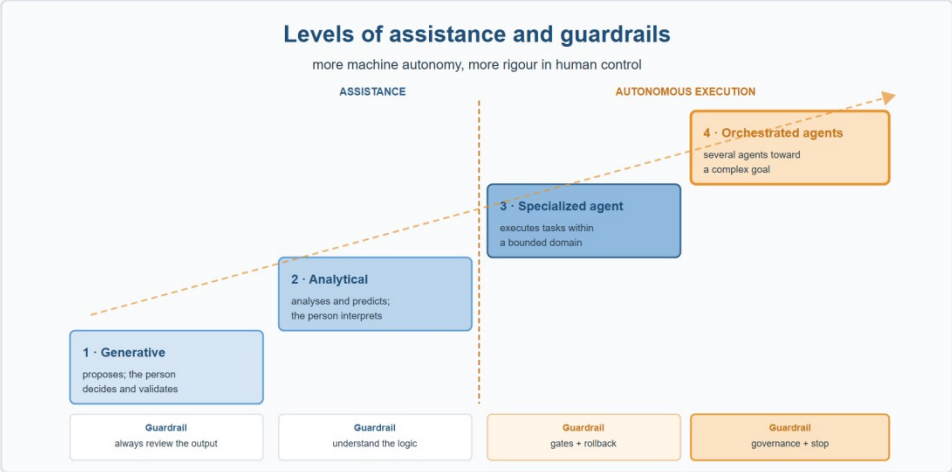

Figure 12. The four levels of assistance and their guardrails. As the machine's autonomy grows, human control intensifies.

Level	What the assistance does	Guardrails
1. Generative	Proposes drafts, summaries, suggestions. The person decides, edits and validates.	The human always reviews. No output is accepted as finished without validation.
2. Analytical	Analyses data, detects patterns, predicts trends.	The human understands the logic of the prediction. No decision is made solely because the algorithm suggests it.
3. Specialized agent	Executes complete tasks within a bounded domain.	Mandatory human quality gates. Transparent logs. Immediate rollback.
4. Orchestrated agents	Several agents coordinate complex goals with minimal human intervention.	Formal governance. A human responsible for each agent. Periodic audit. Manual stop.

The first two levels are assistance; the last two, autonomous execution. The border between the second and the third weighs the most, because it is where work stops being "human with help" and acquires a rhythm of its own that has to be coordinated.

The field of possibilities

In 2026, the debate on the future of agility has no single answer. Three stances coexist, which the professional should know as coordinates on a map, not as settled truths. Seen in the light of the knowledge spiral, they are three different readings of the same synthesis in formation.

Empiricism remains the core. For part of the community, the pillars of agility —transparency, inspection, adaptation, delivery of value, people over processes— are more valuable than ever. Assistance accelerates complexity, but does not eliminate it, and empirical guardrails matter more in a fast world, because speed without direction is only faster chaos. From this reading, assistance is one more tool within the framework: it changes the speed of execution, not the logic of coordination.

The framework evolves from within. Another part proposes integrating assistance into specific activities —augmented estimation, assisted planning, automated facilitation, functions redefined for hybrid teams —. Roles mutate and artifacts adapt, but the empirical pillars remain.

The traditional framework loses its meaning with autonomous agents. An influential minority holds that agile frameworks were designed for a world where people were the only bottleneck. If assistance removes that bottleneck, cycles, daily meetings and retrospectives are unnecessary for the machine, and they propose continuous flows, dual channels or pattern languages that replace entire frameworks. From this reading, traditional frameworks completed their mission for the previous era, and now different intelligences in the team need different rhythms of coordination.

These three stances are not three answers to choose from, but three tensions of one same synthesis that is still taking shape. The agility professional should not be a die-hard defender of any of them: they should know the three coordinates, diagnose what kind of team and organization they work in, and help their team experiment consciously, keeping empiricism as the beacon. That is, deep down, the ri attitude: not applying a recipe, but building one's own way of working with judgement. And, faithful to the spiral, doing so knowing that the synthesis built today will tomorrow be the thesis that a new antithesis will put to the test again.

Part VI — Principles and values

The roots beneath the ice

If architectures, practices and functions are configurable pieces, principles and values are the criterion that says whether the resulting configuration is still agile. They are what the antithesis consolidates, not what it challenges, and that is why they close the manual: they are the firm ground on which any synthesis, present or future, is built.

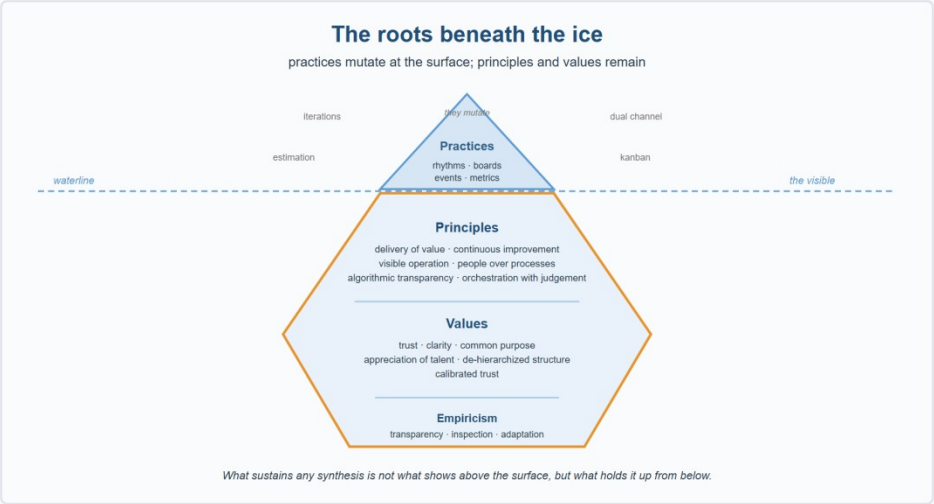

Figure 13. Practices are the visible tip that changes with each turn of the spiral; principles, values and empiricism are the mass that holds them up.

Principles

Principles are the support of practices. Without them, tools are only noise.

- **Delivery of value.** Early and continuous delivery of value to the customer, which requires the customer to collaborate with the team and the vision to be shared and understood.
- **Continuous improvement.** Frequent reflection on the methods of work, questioning their effectiveness and adapting them. The same self-critical effort is applied to improving products and services.
- **Iterative, incremental or continuous development.** The product is not built according to a closed initial plan, but starting from a minimum viable product to which increments are added, or flowing continuously.
- **Sustainable pace of work.** Reaching a pace that avoids both the expansion of work to fill the available time and the pressure of discovering delays too late. The speed that assistance brings must not become an expectation of permanent overexertion.
- **Continuous attention to excellence.** Use of techniques that guarantee quality and allow errors to be detected early. This principle gains importance when assistance generates more work,

faster: without mature control, volume accelerates the appearance of defects instead of preventing it.

- **Visible operation.** Information is shared clearly to facilitate collaboration, identify impediments early and allow the whole team to know the state of the work and contribute ideas.
- **Cadence and synchronization.** Making the frequency of meetings and deliveries predictable, and synchronizing the work of different teams or lanes. The dual channel is, at bottom, a problem of cadence: coordinating two different rhythms.
- **People over processes.** The collective intelligence of the team, its tacit knowledge, is responsible for the quality of the product. Processes and assistance are aids, never substitutes. Far from weakening, this principle becomes the most decisive one in the era of assistance.

To these classic principles, the arrival of intelligent assistance adds two that previously did not need stating.

- **Algorithmic transparency.** Classic transparency dealt with the state of the work; this one deals with its provenance. The team must be able to know which decisions and outputs were human and which were assisted, and which guardrails and limits of autonomy govern the assistance. It is not about understanding the machine's inner workings, but about being able to inspect the origin of what is produced. If where a decision comes from cannot be inspected, no empiricism about it is possible.
- **Orchestration with judgement.** The machine executes; the person judges. To orchestrate is not to operate, but to decide what is delegated, what is validated and what is discussed, without the chain of human responsibility ever being hidden. From this follows a practical rule: what assistance produces and no human can explain or take ownership of should not reach the customer.

Values

A company's culture is the sum of its organizational and governance characteristics, which can accelerate or slow down the development of agility.

- **Assertiveness.** The ability to express what one thinks directly, honestly and respectfully.
- **Appreciation of talent.** Recognizing that the value of the product depends on people's tacit knowledge.
- **Clarity.** Transparency in information, in decisions and in goals.
- **Trust.** The basis of self-organization. Without trust there is no team, only supervision.
- **De-hierarchized structure.** Decisions are made where the knowledge is, not where the rank is.
- **Common purpose.** The team shares a goal that transcends individual interests.
- **Calibrated trust.** Neither blind adoption of assistance nor rejection out of distrust, but the ability to assess when it amplifies the team's capabilities and when it exposes its weaknesses. Trust in

the machine is built the same way as trust in a colleague: with transparency, inspection and adaptation.

These principles and values are the roots that remain and support the new practices. That is the ultimate meaning of a training that does not seek to teach a model, but the ability to build one's own: each team's own field of scrum, at each moment of the spiral.

Appendix

Information, resources and professional community

- Latest version (and other Scrum Manager resources).
- Skill Arena: professional upkeep platform.
- About the Scrum Manager® certification.
- Frequently asked questions about Scrum Manager® courses and exams.

Continuous improvement and quality control

Scrum Manager's quality control is based on the assessments of its students. With your opinion you help us maintain the standard of our materials, courses, centres and teachers. If you have taken part in a training activity audited by Scrum Manager®, you can send us your comments from the "Members area".

About the context sources

The diagnosis of the antithesis and the inventory of proposals of the synthesis draw on the state of professional and academic knowledge as of mid-2026. The text deliberately prioritizes principles and underlying dynamics over specific tools, which change rapidly. Among the context sources used are the DORA 2025 report on AI-assisted development, the annual reports on the state of agility, the analyses on the evolution of agile roles and the shrinking of teams, and the proposals for AI-augmented life cycles and frameworks that have emerged in the community. Scrum Manager® maintains an agnostic stance: it offers the inventory of available knowledge so that each professional can develop, with judgement, their own model of work.