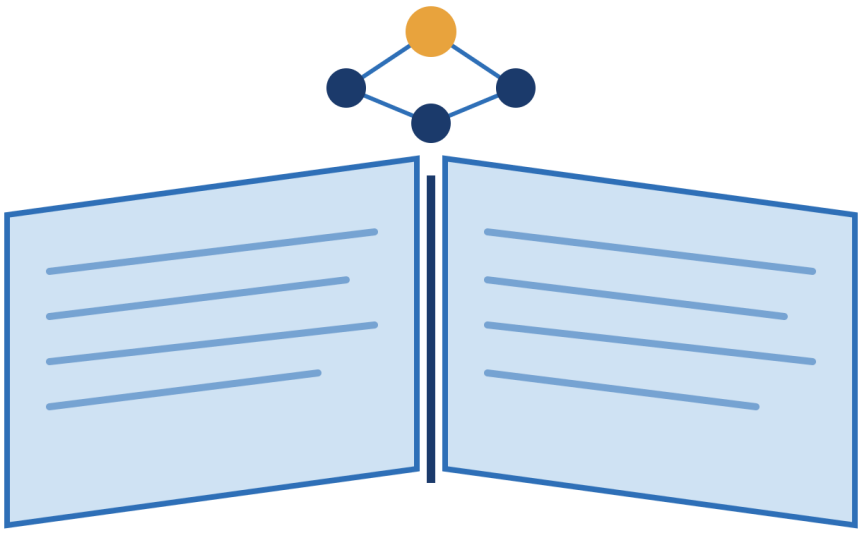
Scrum Manager
Uncovering better ways

Guide

AI applied to agile work

In-depth development of the State of the art topics



Updated June 2026

Scrum Manager® · Skill Arena

About this document

This document is an in-depth development of the topics in the reference map (State of the art) for bringing artificial intelligence into agile work with sound judgement, which is available at: Skill Arena.

Use it as reference material to keep your practice aligned with, and checked against, current professional knowledge.

It is complemented by the training and assessment platform in Skill Arena. In the AI applied to agile work area you can take training tests to check and improve your level of knowledge and, if you wish, you can also obtain a diploma that provides curricular accreditation of professional competence and currency in this area.

State of knowledge

Knowledge about AI applied to agile work is a field in full ferment that evolves extremely fast: practices and tools appear, mature or become obsolete within months, and only a part of the fundamentals can be considered settled. Throughout the document, the labels (ESTABLISHED, CONSOLIDATING, EMERGING) help identify the maturity of each concept:

ESTABLISHED settled consensus; knowledge taken as required.

CONSOLIDATING gaining adoption fast; not yet universal but already relevant.

EMERGING recent frontier; high relevance and high volatility.

Contents

Chapters of the guide, in the order of the State of the art.

Block A · Fundamentals of interacting with AI

1. Context engineering · CONSOLIDATING
2. Structured, reusable prompting · ESTABLISHED
3. Verification and reliability of responses · ESTABLISHED
4. Trust boundary: instructions, data and privacy · ESTABLISHED

Block B · Working with AI agents

5. From chat to agent: delegating verifiable work · CONSOLIDATING
6. Designing human oversight (human-in-the-loop) · CONSOLIDATING
7. Agentic ecosystem standards (MCP, AGENTS.md) · EMERGING

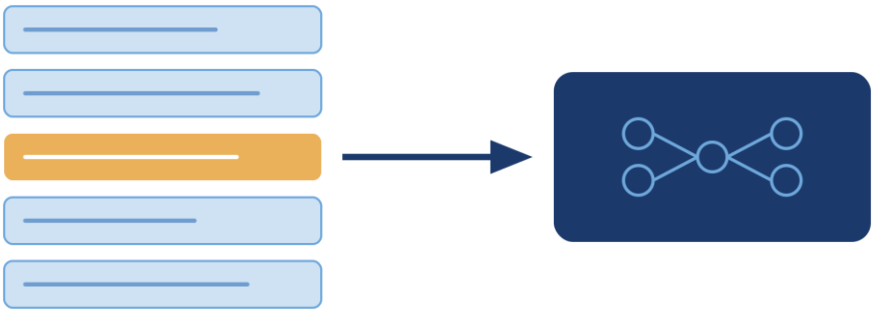
Block C · Application to the agile work cycle

8. AI in the product and backlog flow · ESTABLISHED
9. AI in facilitation and team dynamics · CONSOLIDATING
10. AI in quality and delivery — development roles · ESTABLISHED

BLOCK A · FUNDAMENTALS OF INTERACTING WITH AI

Chapter 1. Context engineering · CONSOLIDATING

Deliberately designing what information the model receives and in what order, not just how the one-off instruction is worded.



Context engineering governs what information enters the model, in what order and what is left out.

Introduction

During the first years of working with language models, professional effort concentrated on writing the perfect instruction: finding the words and structure that produced the best response. That skill —prompting— is still necessary, but it has ceased to be the centre. As work with AI has moved from the single-turn conversation to sustained tasks and the use of agents, a broader competency that encompasses it has emerged: context engineering.

Context engineering is the set of strategies for deciding what information is made available to the model at each moment, how it is ordered and what is left out. The shift in focus is subtle but decisive: the question stops being “how do I word this request?” and becomes “what configuration of information makes the result I am after more likely?”. This chapter develops that competency, its relationship with prompting and the practical strategies that articulate it.

It is a consolidating topic: the conceptual framework is settled and adopted by the main players in the sector, but the concrete techniques continue to evolve rapidly. That is why it is worth mastering the stable principles —which are the ones this chapter develops— and keeping active watch over the tools, which change faster.

1.1 From the instruction to the context

Prompting deals with a specific unit: the instruction written at a given moment. Context engineering deals with everything the model “sees” when it generates a response, which is considerably more than the instruction: it includes the system prompt, the conversation history, the retrieved documents, the results of tools the model has used and the examples it has been given.

The most widespread analogy is that of working memory. If the model is understood as a processor, its context window is the working memory: a finite space that holds only a limited amount of information, and where whatever is included competes for the model's attention. Context engineering is the deliberate management of that scarce space.

Core idea. Prompting is a component of context engineering, not a rival discipline. For simple tasks —a one-off query, a short text— a good prompt is enough. Context engineering becomes indispensable when the work is sustained, when agents are involved or when the relevant information exceeds what fits comfortably in a single instruction.

1.2 Why more context is not better context

Intuition suggests that, since the model feeds on information, it is best to give it all that is available. Practice proves the opposite, for two well-documented reasons.

The attention budget. Models do not weight all elements equally. Like a person with limited working memory, they prioritise recent or more salient information, and important details dilute into the noise when the input is very long. Adding irrelevant context is not neutral: it degrades the model's ability to attend to what matters.

Degradation in long contexts. As the context grows, the reliability of responses declines. The model tends to contradict itself, to omit instructions or to fabricate information. A related phenomenon is “lost in the middle”: information placed at the beginning and end of the context is processed more reliably than that buried in the middle. Hence order matters as much as content.

The practical consequence is a design principle: the goal is not to maximise information but to curate it. Include just what is needed, order it well and remove what has ceased to be useful.

1.3 The four strategies: write, select, compress, isolate

The sector has converged on a framework of four complementary strategies for managing context. They are not mutually exclusive: most serious systems combine all four. Knowing them gives a common vocabulary for reasoning about the problem.

1. **Write.** Store information outside the context window so that it is available without consuming space. The typical pattern is the “scratchpad”: the agent records decisions, findings or progress in an external store and retrieves them when needed, instead of dragging everything along in the conversation.
2. **Select.** Bring into the window only the information relevant to the current step. It is the retrieval strategy (RAG and similar): instead of including everything, what is pertinent is identified and provided for each interaction, through semantic search or targeted retrieval.
3. **Compress.** Retain only the tokens necessary for the task, reducing size without losing meaning: summarising long conversations, extracting the key facts, condensing a tool's lengthy outputs before they enter the context.
4. **Isolate.** Separate distinct contexts so that they do not interfere with one another. The most advanced form is multi-agent architecture: instead of a single agent holding everything in one huge window, several specialised agents, each with its focused context.

Common production pattern. A multi-agent system (isolate) in which each subagent retrieves what it needs (select), keeps a progress notebook (write) and summarises long tool outputs so they fit in its window (compress). The four strategies operating together.

1.4 Context engineering in agile work

For an agile professional, this competency is not a technical matter of infrastructure but a way of framing any AI-assisted work. Before asking a model for something, the useful question is not “what do I write to it?” but “what does it need to know to do this well, and what is superfluous?”.

In everyday practice this translates into concrete habits: providing the minimum sufficient context and not dumping whole documents “just in case”; placing the instruction and the critical data in high-attention positions; separating distinct conversations instead of dragging along a single endless thread that mixes topics; and, in long tasks, relying on external notes rather than trusting the model to “remember” everything said. These are direct applications of the four strategies to work without code.

What you should be able to do

Having mastered this chapter, a professional is able to:

- Distinguish context engineering from prompting, and explain why the former encompasses the latter rather than replacing it.
- Recognise when a task requires context management (sustained work, agents, extensive information) and when a good prompt is enough.
- Justify why adding more context can degrade the result, appealing to the attention budget and to degradation in long contexts.
- Identify which of the four strategies —write, select, compress, isolate— solves a specific context problem.
- Apply the principle of minimum sufficient context and the ordering of information to a real request from their work.

Common mistakes and antipatterns

Confusing context engineering with writing longer prompts. Lengthening the instruction is not managing context; it often makes it worse, because it adds noise. Management is about curating, not accumulating.

Dumping all available information “just in case”. It is the most common and most costly mistake: it dilutes the model's attention and degrades reliability. The correct practice is deliberate minimisation.

Ignoring the ordering of information. Burying the critical instruction in the middle of a long context reduces the likelihood that the model will attend to it. Decisive instructions go in high-attention positions.

Dragging a single thread for everything. Mixing distinct topics and tasks in one endless conversation causes interference and confusion. Isolating distinct contexts is often the simplest and most effective solution.

Further reading

· Anthropic — Effective context engineering for AI agents

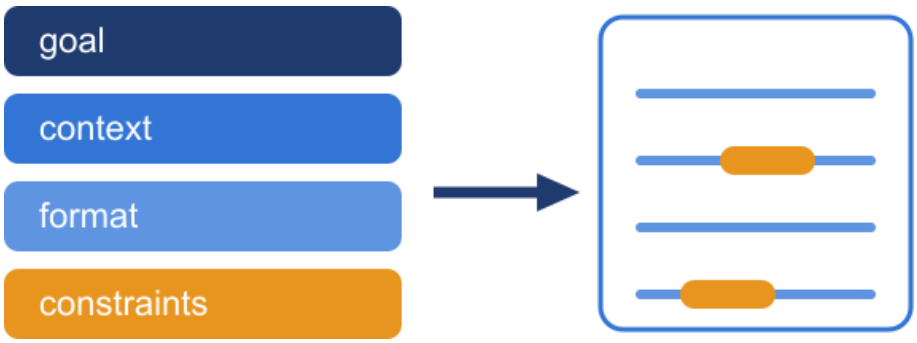
· LangChain — Context engineering for agents

· Anthropic — Prompt engineering documentation

BLOCK A · FUNDAMENTALS OF INTERACTING WITH AI

Chapter 2. Structured, reusable prompting · ESTABLISHED

Specifying a request with the components that make its result predictable and verifiable.



An effective request is made up of stable parts that become a reusable template.

Introduction

Prompting is the base competency of all work with language models: the ability to formulate a request so that the result is useful, predictable and verifiable. Although context engineering frames it within a broader framework (chapter 1), prompting remains the foundation: without a well-constructed instruction, no context strategy rescues a badly asked-for result.

This chapter develops the components of an effective prompt, the distinction between instructions that produce consistency and those that do not, and the practice of turning useful prompts into reusable templates at team scale.

2.1 The components of an effective request

A predictable prompt is built from four explicit components. Their absence is the most frequent cause of erratic results.

5. **Goal with success criteria.** What is wanted and how it will be recognised that the response is good. A goal with no observable criterion (“do it well”) does not guide the model.
6. **Minimum sufficient context.** The information the model needs for the task, no more and no less. Here it links with context engineering: excess degrades as much as deficiency.
7. **Output format.** The expected form of the response: length, structure, tone, type of elements. It is the component that contributes the most consistency.
8. **Explicit constraints.** The limits: what not to do, what to avoid, what conditions to respect. Constraints reduce the space of unwanted responses.

2.2 Structural versus qualitative instructions

A decisive operational distinction: structural instructions —those that define the form of the response— produce consistency; qualitative ones —“be more thorough”, “do it better”— do not. Asking for “a list of five points, each one sentence long” gives a reproducible result; asking to “be more complete” depends on an interpretation that varies with each run.

Practical rule. When a prompt does not give the expected result, the correct reflex is not to add qualitative adjectives but to ask which structural component is missing. Improvement comes from specifying the form better, not from insisting on quality.

2.3 Breaking down overloaded requests

A prompt that asks too much at once produces results of uneven quality: the model attends to some parts and neglects others. The correct practice is to break the request down into sequential steps with explicit priority, or to split a complex task into several chained interactions. This connects with the isolation strategy of context engineering.

2.4 From prompt to reusable template

The leap that distinguishes professional use from occasional use is reuse. An effective prompt is not disposable text: it is an artefact worth saving, versioning and sharing with the team, so that different people obtain comparable results. Treating prompts as team assets —reviewed and maintained like code— is the practice that stabilises quality at scale and links directly with the specification methodology (SDD) and with the reusable instruction files of the agentic ecosystem.

What you should be able to do

Having mastered this chapter, a professional is able to:

- Build a prompt with a goal, observable success criteria, minimum context, output format and explicit constraints.
- Diagnose why a prompt fails by identifying which structural component it lacks, instead of adding qualitative instructions.
- Distinguish structural from qualitative instructions and explain why only the former produce consistency.
- Break down an overloaded request into sequential steps with explicit priority.
- Turn an effective prompt into a reusable template that gives comparable results across different people.

Common mistakes and antipatterns

Responding to a poor result by adding adjectives. “Be more thorough” rarely improves things; what is missing is almost always a specific structural component.

Asking too much in a single instruction. Overload produces uneven attention. Breaking down is more reliable than trusting the model to cover everything.

Treating prompts as disposable text. Failing to save and version effective prompts condemns the team to reinventing them and to obtaining inconsistent results across people.

Omitting the output format. Without specifying the expected form, each run produces a different structure, which prevents comparison or automation.

Further reading

· OpenAI — Prompt engineering guide

· Anthropic — Prompt engineering overview

BLOCK A · FUNDAMENTALS OF INTERACTING WITH AI

Chapter 3. Verification and reliability of responses · ESTABLISHED

Critically assessing what a model produces before acting.

Verify before acting: separate facts from inferences and detect unwarranted precision.

Introduction

It is the most timeless competency in the area and, probably, the most important. Language models produce fluent, convincing text regardless of its accuracy: fluency is not a sign of truthfulness. Verifying is the practice of subjecting what is generated to scrutiny before acting on it. Far from losing relevance as models advance, it gains it: the more work is delegated, the more critical it is to know how to check what the system delivers.

3.1 Separating facts, inferences and recommendations

A model's response often mixes three things of different natures: verifiable facts, the model's inferences from those facts, and recommendations for action. The first discipline of verification is to separate them, because each demands a different level of checking. A fact is checked against a source; an inference is examined in its logic; a recommendation is evaluated against the professional's judgement and context.

3.2 Asking for the assumptions and the evidence

An effective technique consists of not accepting the conclusion without asking for what supports it: the starting assumptions and the evidence. Asking the model to make explicit what it relies on, or to distinguish what it knows from what it infers, reveals the real solidity of the response and exposes the weak points that fluency concealed.

3.3 Unwarranted precision

A characteristic risk pattern is “unwarranted precision”: specific figures, percentages or categorical certainties presented with no backing. Concreteness produces a sense of rigour that does not always correspond to reality. Faced with a precise datum, the verification question is always the same: where does it come from? If there is no source, the precision is decorative and must be treated as suspect.

Guiding principle. The professional's capacity for judgement is the irreplaceable human value in work with AI. Verification is not a quality-control formality: it is the place where professional judgement contributes what the model cannot contribute on its own.

3.4 The “zero trust” approach applied to information

The safest stance towards a model's output is zero trust by default: it is not assumed valid until it has been verified in proportion to the risk of acting on it. This does not mean distrusting everything equally —that would be paralysing— but calibrating the verification effort according to the consequences of an error: an idea for a brainstorm does not require the same as a datum that will feed a business decision.

What you should be able to do

Having mastered this chapter, a professional is able to:

- Separate, in a response, the verifiable facts, the model's inferences and the recommendations for action.
- Ask for and examine the assumptions and evidence that support a claim, instead of accepting the conclusion.
- Detect unwarranted precision and treat as suspect any figure or certainty without a source.
- Calibrate the level of verification in proportion to the risk of acting on the information.
- Recognise that the fluency of the text is no indication of its accuracy.

Common mistakes and antipatterns

Confusing fluency with truthfulness. A well-written, self-assured text can be false. Form does not vouch for substance.

Accepting precision as rigour. A specific figure without a source is no more reliable than a vague claim; it is often more dangerous, because it inspires confidence.

Verifying everything equally or verifying nothing. Both extremes fail: the first paralyses, the second exposes. Calibration by risk is the correct practice.

Delegating verification to the model itself without external judgement. Asking the model whether it is sure does not replace checking against a source or professional judgement.

Further reading

· NIST — AI Risk Management Framework

· Anthropic — Reducing hallucinations

BLOCK A · FUNDAMENTALS OF INTERACTING WITH AI

Chapter 4. Trust boundary: instructions, data and privacy · ESTABLISHED

Treating external content as untrusted and protecting sensitive information.

The trust boundary separates the professional's instructions from untrusted external data.

Introduction

This chapter brings together two security competencies that share the same logic of distrust by default: the separation between instructions and data —to prevent the injection of malicious instructions— and the protection of sensitive information before sharing it with a model. What in conversational use was a defensive good practice becomes a central competency when agents execute actions on real systems.

4.1 Instructions versus data: prompt injection

A model does not reliably distinguish between the instructions it must obey and the data it should merely process. If a document, a web page or an email that the model reads contains text worded as a command, the model may execute it. This is instruction injection (prompt injection): malicious content embedded in apparently innocuous data that hijacks the model's behaviour.

The defence is to treat all content of external origin as untrusted data by default, and to separate it explicitly from legitimate instructions. The risk escalates with agents: an injected instruction that previously only produced an improper response can now trigger a real, unintended action on a system.

Why it matters more with agents. When the model only responds, an injection produces, at most, a bad response that the human reviews. When the agent executes —sends emails, modifies data, calls tools—, the injected instruction becomes an action with real consequences. The trust boundary moves from hygiene to critical security.

4.2 Privacy and handling of secrets

The second competency is the protection of the information shared with a model. The principle is minimisation: providing only the data necessary for the task, and effectively anonymising anything that could identify people or organisations. Effective anonymisation is not cosmetic: replacing a name but leaving data that allow re-identification does not protect.

A specific mistake is relying on labels or promises: marking a text as “confidential” within a prompt does not prevent the model from processing or reflecting it, and the guarantees about data handling depend on the specific service, not on the user's wish. The practical rule: do not share with a model anything you would not be willing to see outside your own control, except under verified contractual and technical conditions.

What you should be able to do

Having mastered this chapter, a professional is able to:

- Explain why a model does not reliably distinguish instructions from data, and what instruction injection is.
- Treat content of external origin as untrusted by default and separate it from legitimate instructions.
- Justify why the risk of injection escalates when an agent executes actions instead of only responding.
- Apply the principle of minimisation and effective anonymisation before sharing information with a model.
- Recognise that confidentiality labels and system promises do not replace contractual and technical guarantees.

Common mistakes and antipatterns

Treating external content as trusted instructions. Reading a document or a web page without assuming it may contain embedded commands opens the door to injection.

Apparent anonymisation. Replacing a name while leaving re-identifiable data does not protect; it is a false sense of security.

Relying on the “confidential” label. Marking something as secret within the prompt does not change how the system treats it.

Dumping sensitive data “because it is an internal tool”. The sensitivity of the data does not depend on the convenience of the channel; it depends on the verified guarantees of that channel.

Further reading

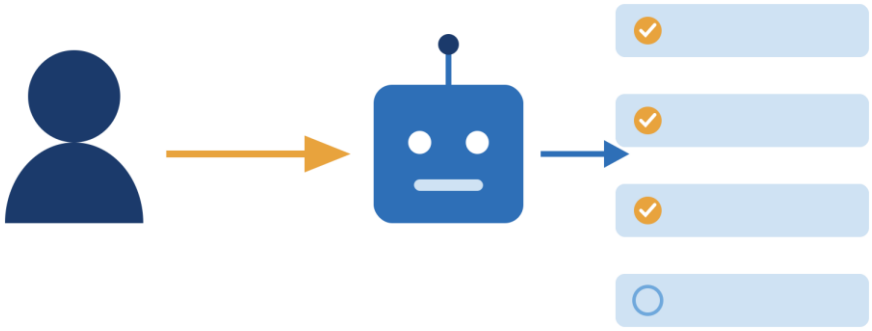
· OWASP Top 10 for LLM Applications

· EU AI Act

BLOCK B · WORKING WITH AI AGENTS

Chapter 5. From chat to agent: delegating verifiable work · CONSOLIDATING

Moving from using AI to think better to delegating work to it, knowing when and with what limits.



From chat to agent: the professional moves from executing to delegating work with acceptance criteria.

Introduction

The most relevant change of recent months is not that models are more capable, but that they have moved from responding to executing. A conversational assistant answers questions; an agent completes multi-step tasks autonomously: it searches, decides, uses tools and produces a result. This chapter develops what that leap means for the agile professional and how one decides what to delegate.

5.1 The difference between responding and executing

An assistant that responds always leaves the action in the human's hands: it suggests, and the person acts. An agent acts on its own within the limits set for it. The difference is not one of degree but of nature: it changes who does the work and, therefore, where the risk lies and where the value lies. The professional's role shifts from “doing” to “deciding what is delegated, with what limits and with what acceptance criteria”.

5.2 Verifiable tasks and non-delegable tasks

Not all work can be delegated with the same safety. The useful distinction is verifiability: a task whose result can be checked against a clear criterion delegates well; a task whose “well done” depends on tacit judgement or unwritten context is a poor candidate. It is best to delegate what can be verified and to reserve what demands irreplaceable human judgement: decisions with ethical implications, those that affect people, novel ones that fall outside any known pattern.

Delegation criterion. Before delegating a task to an agent, two questions: can I verify the result against a clear criterion? and are the consequences of an error bearable and reversible? If both answers are affirmative, it is a good candidate. If either fails, it is best to keep the human in the loop (chapter 6).

5.3 Impact on agile ceremonies and artefacts

Delegation reconfigures agile work. The unit of work stops being only “the task I do” and includes “the task I orchestrate and supervise”. This affects planning (estimating orchestrated work is different from estimating manual work), tracking (progress includes what the agents do) and the definition of done (which must account for the verification of what AI produces). Knowing how to delegate well becomes a team competency, not just an individual one.

What you should be able to do

Having mastered this chapter, a professional is able to:

- Explain the difference in nature between an assistant that responds and an agent that executes.
- Decide which tasks are delegable to an agent according to their verifiability and the reversibility of their consequences.
- Identify the tasks that demand irreplaceable human judgement and should not be delegated.
- Reframe their role from executor to designer of the delegation: limits, acceptance criteria and supervision.
- Anticipate the impact of delegation on the team's planning, tracking and definition of done.

Common mistakes and antipatterns

Delegating non-verifiable tasks. If the result cannot be checked against a clear criterion, delegation shifts the risk without control.

Delegating decisions that demand human judgement. Ethical decisions, those that affect people or novel ones are not candidates for autonomy, however capable the agent.

Treating the agent as an infallible executor. Autonomy does not remove the need for verification; it shifts it to the design of the control points.

Not adjusting team practices. Keeping planning, tracking and DoD as if all work were manual ignores the orchestrated part and creates blind spots.

Further reading

· Anthropic — Building effective agents

· Anthropic — Effective context engineering for AI agents

BLOCK B · WORKING WITH AI AGENTS

Chapter 6. Designing human oversight (human-in-the-loop) · CONSOLIDATING

Deciding where an AI-assisted flow should stop so that a person can review, approve or redirect.

Human oversight introduces pause and approval points into the AI-assisted flow.

Introduction

If delegation (chapter 5) is giving work to an agent, human oversight is its necessary counterpart: deciding where the flow stops so that a person intervenes. The agile professional designs the points where the human takes part, rather than executing each step by hand. This chapter develops the oversight models, the criteria for placing approval gates and the compliance dimension that accompanies them.

6.1 Three oversight models

Oversight is not an all-or-nothing switch but a spectrum with three recognisable positions that can coexist within the same flow:

9. **Human in the loop.** The system stops and requires human approval before executing an action. It is the strictest control, for high-risk or irreversible decisions.
10. **Human on the loop.** The human supervises the flow and can intervene if anomalies are detected, but does not approve each individual action. Suitable for medium risk.
11. **Human out of the loop.** The agent acts autonomously; the human reviews, if at all, in a deferred and aggregated way. Reserved for low-risk and reversible actions.

The level of oversight is a property of the decision, not of the system: it is determined dynamically according to risk, context and policy. The same agent can operate out of the loop for the trivial and require in-the-loop approval for the delicate.

6.2 Where to place approval gates

The practical question is which actions require human approval before being executed. The consolidated criteria: irreversibility (transactions, data deletion, changes in production), write access to sensitive systems, the materiality of the consequences for customers or regulators, and the novelty or ethical sensitivity of the decision. When any of these factors is present, the action is placed behind an approval gate.

Nominal oversight is not real oversight. Placing someone “in the loop” without giving them the information, the authority and the time to genuinely influence the decision is not oversight: it is a

formality that mimics control. A reviewer who approves without real capacity for judgement is worse than the absence of control, because it creates a false guarantee.

6.3 The compliance dimension

Human oversight has ceased to be only good practice and has become a legal requirement in many contexts. The EU AI Act, in its Article 14, requires effective human oversight for high-risk AI systems, applying from 2 August 2026. The scope is extraterritorial: it affects any organisation whose systems are used in the EU. Traceability —that each agent action be attributable to a recorded human authorisation— is the evidentiary basis of that oversight.

What you should be able to do

Having mastered this chapter, a professional is able to:

- Distinguish the three oversight models (in the loop, on the loop, out of the loop) and choose the appropriate one according to risk.
- Place approval gates by applying the criteria of irreversibility, sensitive access, materiality and novelty.
- Recognise when oversight is merely nominal and why that is worse than its absence.
- Explain the human-oversight requirement of Article 14 of the EU AI Act and its timeline.
- Justify traceability as the evidentiary basis of oversight.

Common mistakes and antipatterns

All-or-nothing oversight. Applying the same level of control to every action ignores that risk varies; it saturates the reviewer on the trivial and wears them out for what matters.

A reviewer without information, authority or time. Nominal oversight meets the form but not the substance, and generates a dangerous false guarantee.

Forgetting traceability. Without a record of which human authorised which action, there is no demonstrable oversight or defensible compliance.

Treating compliance as someone else's concern. The extraterritorial scope of the EU AI Act affects more organisations than believe themselves to be outside its remit.

Further reading

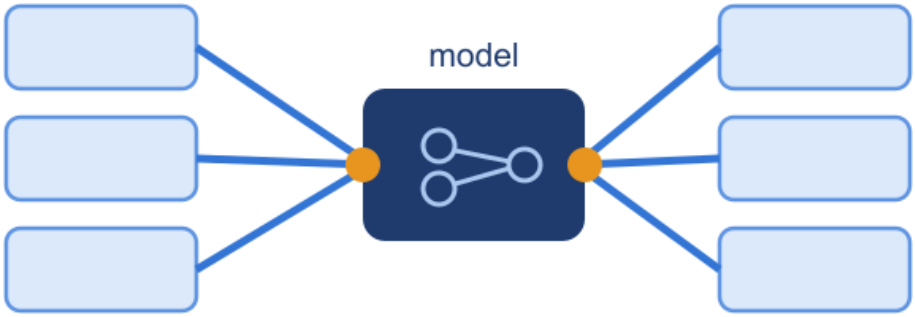
· EU AI Act — implementation timeline

· NIST — AI Risk Management Framework

BLOCK B · WORKING WITH AI AGENTS

Chapter 7. Agentic ecosystem standards (MCP, AGENTS.md) · EMERGING

Knowing the emerging standards that stabilise work with agents.



Ecosystem standards connect the model with tools and data through common interfaces.

Introduction

Work with agents is moving from isolated solutions to an ecosystem with shared standards. For the agile professional it is not necessary to implement them, but it is necessary to understand what they solve and why they are being adopted, because they mark the direction of the field. This chapter presents the two most relevant standards as of the date of the document. It is an emerging topic: what is described here may consolidate or be superseded within a few months.

7.1 The problem the standards solve

Every agent needs to connect to tools and data, and to receive instructions on how to behave. Without standards, each connection and each set of instructions is built bespoke, which fragments the ecosystem and makes it hard to reuse work. The emerging standards seek a common language: that an agent can connect to a tool or read its instructions in the same way, whatever the vendor.

7.2 Model Context Protocol (MCP)

The Model Context Protocol is an open standard for connecting models with external tools and data sources in a uniform way. It solves the connection: instead of a bespoke integration for each tool, MCP offers a common way of exposing data and capabilities to a model. Its adoption by the main vendors has established it as one of the pieces that stabilise the ecosystem.

7.3 AGENTS.md

AGENTS.md is a convention for giving agents reusable instructions through a standardised text file in a project: what to do, what to avoid, what conventions to follow. It externalises the recurring prompt out of the conversation and turns it into a versionable, shareable artefact, in line with treating prompts as team assets (chapter 2).

Adoption context. MCP and AGENTS.md have been folded into an open industry foundation backed by the main AI vendors. That institutional convergence is the signal that the ecosystem is stabilising, even though its concrete application is, today, more relevant to technical roles than to non-technical ones.

7.4 What this means for the agile professional

For a non-technical role, the value of knowing these standards lies not in implementing them but in understanding where the ecosystem is heading: towards interoperability and reuse. Knowing that they exist makes it possible to take part with judgement in team decisions about which tools to adopt and to understand why portability between vendors is beginning to be a valuable property.

What you should be able to do

Having mastered this chapter, a professional is able to:

- Explain what fragmentation problem the agentic ecosystem standards solve.
- Describe, at a high level, what the Model Context Protocol solves (the connection to tools and data).
- Describe what AGENTS.md solves (reusable, versionable instructions for agents).
- Interpret the institutional convergence around these standards as a signal of the ecosystem's stabilisation.
- Situate the value of these standards for non-technical roles in interoperability and portability.

Common mistakes and antipatterns

Believing you have to implement them to know them. The value for non-technical roles lies in understanding the direction of the field, not in implementation.

Confusing MCP and AGENTS.md. They solve different things: MCP connects to tools and data; AGENTS.md gives reusable instructions.

Taking the current state as definitive. It is an emerging topic; it is worth reviewing its evolution at each update of the map.

Ignoring portability. Adopting tools without considering interoperability can generate dependencies that are hard to reverse.

Further reading

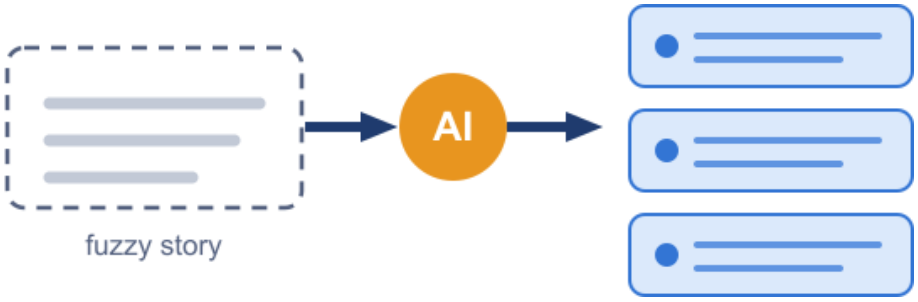
· Model Context Protocol

· AGENTS.md

BLOCK C · APPLICATION TO THE AGILE WORK CYCLE

Chapter 8. AI in the product and backlog flow · ESTABLISHED

AI support to refine the backlog, draft acceptance criteria and frame experiments.



AI reduces backlog ambiguity: from the fuzzy story to items with verifiable criteria.

Introduction

It is the most everyday and cross-cutting use of AI in agility, applicable to product owners, scrum masters and any role that works with the backlog. Its value lies in reducing ambiguity and rework without introducing bureaucracy, always keeping the decision with the team. This chapter develops the concrete applications and the limit that makes them useful.

8.1 Refinement and ambiguity detection

AI is effective at detecting ambiguity in user stories and generating refinement questions the team may not have considered. It does not replace the refinement conversation: it feeds it. The correct pattern is to use the questions it generates as a starting point for the team's dialogue, not as a questionnaire to be answered mechanically.

8.2 Slicing with demonstrable value

Splitting large stories while keeping value in each slice is a difficult skill. AI can propose splits (slicing) and alternatives the team had not considered. Human judgement decides which one keeps real, demonstrable value: the AI's proposal is material for the decision, not the decision.

8.3 Verifiable acceptance criteria

AI helps draft more complete acceptance criteria, including negative scenarios and edge cases that are easy to forget. A useful acceptance criterion covers not only the happy path but what should happen when something goes wrong. AI is good at recalling those cases; the team validates that they are the relevant ones for its context.

8.4 Hypotheses and experiments

In the discovery space, AI supports the formulation of hypotheses and the design of experiments with metrics and guardrails. It helps articulate what is to be learned, how it will be measured and what signals would indicate it is best to stop. The decision of what to experiment with and how to interpret the results remains the team's.

The limit that makes it useful. In all these uses, AI produces material for the team's conversation, not closed requirements. The moment the AI's suggestions are accepted without discussion is the moment this practice stops adding value and begins to replace the judgement that justified it.

What you should be able to do

Having mastered this chapter, a professional is able to:

- Use AI to detect ambiguity in stories and generate refinement questions that feed the team's conversation.
- Rely on AI to propose story splits, deciding with judgement which one keeps demonstrable value.
- Draft, with AI support, acceptance criteria that cover negative scenarios and edge cases.
- Formulate hypotheses and design experiments with metrics and guardrails with AI assistance.
- Keep the decision with the team, treating the AI's outputs as a starting point and not as a requirement.

Common mistakes and antipatterns

Accepting the suggestions without discussion. When the AI's proposal becomes a requirement without passing through the team's judgement, the value of the practice is lost.

Replacing the refinement conversation. The AI's questions feed the dialogue; they do not replace it.

Acceptance criteria for the happy path only. Forgetting negative scenarios and edge cases produces incomplete criteria; AI helps to recall them.

Introducing bureaucracy on the pretext of AI. If AI support adds steps without reducing ambiguity or rework, it is getting in the way, not helping.

Further reading

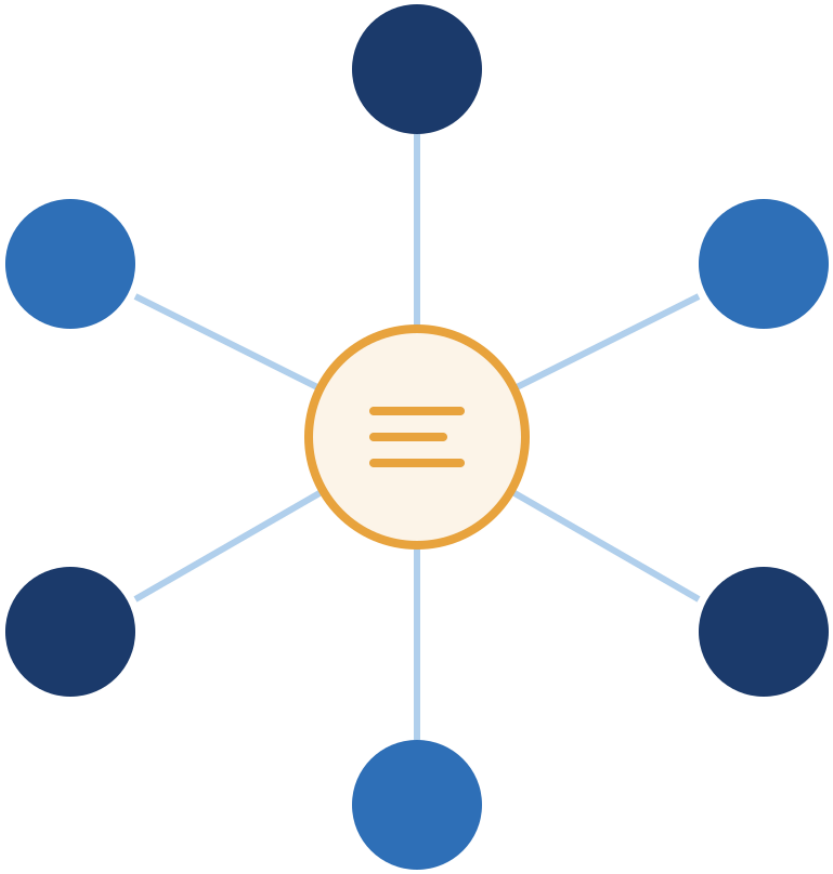
· Scrum.org — The Scrum Master and artificial intelligence

· Scrum Manager — Scrum in the age of AI

BLOCK C · APPLICATION TO THE AGILE WORK CYCLE

Chapter 9. AI in facilitation and team dynamics · CONSOLIDATING

AI support to prepare facilitation and analyse dynamics, without replacing human interaction.



AI synthesises and prepares, without replacing the human interaction that sustains facilitation.

Introduction

It is the use most missing from materials on AI and agility, and the most directly useful for scrum masters and agile coaches who do not work with code. Well framed, it frees up preparation time and enriches the analysis; badly framed, it depersonalises what only works between people. This chapter develops the legitimate uses and the boundary it is best not to cross.

9.1 Preparing facilitation

AI is a good support for preparing sessions: proposing retrospective formats tailored to a situation, generating opening questions, structuring an activity or anticipating how to approach a difficult conversation. Here the value is clear and the risk low: AI prepares, the facilitator leads.

9.2 Discovery synthesis and qualitative analysis

After a discovery session or a retrospective, AI helps synthesise large quantities of qualitative material: grouping themes, identifying patterns, highlighting recurring tensions. It is an analytical

support that saves hours of manual sorting. The interpretation of what those patterns mean for the team remains the facilitator's competency, who knows the context that AI does not see.

9.3 The boundary it is best not to cross

There is a sharp limit: AI does not replace human interaction or the reading of the team's climate. Psychological safety, trust, the perception of the unsaid, the response to emotion in the room —all of that is the facilitator's exclusive competency. Using AI to prepare and analyse is legitimate; using it to mediate the relationship between people empties facilitation of its meaning.

Guiding principle. AI touches facilitation work at its edges —the prior preparation and the subsequent analysis— but not at its centre, which is human interaction in real time. That centre is, precisely, where the facilitator's professional value resides and what AI cannot contribute.

What you should be able to do

Having mastered this chapter, a professional is able to:

- Use AI to prepare sessions: formats, questions, activity structure, approaching difficult conversations.
- Rely on AI to synthesise and qualitatively analyse discovery and retrospective material.
- Interpret the patterns AI highlights in the light of the team's context, which AI does not know.
- Recognise the boundary: AI does not mediate human interaction or read the team's climate.
- Situate the facilitator's professional value in real-time interaction, not in preparation or analysis.

Common mistakes and antipatterns

Delegating the reading of the team's climate to AI. The perception of the unsaid and of emotion in the room is an irreplaceable human competency.

Accepting the AI's synthesis without interpreting it. The patterns AI groups lack the context that only the facilitator provides.

Depersonalising facilitation. Using AI to mediate the relationship between people empties the activity of its purpose.

Confusing preparation with leading. AI prepares the session; leading it, with all it entails of presence and judgement, is the facilitator's.

Further reading

· Scrum Manager — Scrum in the age of AI

· Anthropic — Building effective agents

BLOCK C · APPLICATION TO THE AGILE WORK CYCLE

Chapter 10. AI in quality and delivery — development roles · ESTABLISHED

AI support in testing and code review, keeping the professional's technical judgement.

AI extends test coverage and review, including edge cases beyond the happy path.

Introduction

This chapter covers a high-value use for technical teams. It is explicitly marked as applicable to development roles because its direct application requires working with code, unlike the cross-cutting uses of the previous chapters. For those who do not program, the chapter provides an understanding of the scope; for those who do, concrete practices.

10.1 Generating and reviewing tests

AI extends the scope of testing: it generates cases aligned with acceptance criteria and with the identified risks, and it is especially useful for ensuring coverage of negative scenarios and edge cases beyond the “happy path”. The professional validates that the generated cases are the relevant ones and that they cover what really matters; the number of tests is no sign of quality if they do not attack the real risks.

10.2 Support in code review

As a peer in code review, AI helps detect issues of readability, maintainability, documentation and, above all, security risks: input validation, error handling, permission management, side effects. It is a support that extends the human reviewer's coverage, not a substitute for their judgement. The final decision on a change remains the technical professional's.

Technical judgement is not delegated. AI in quality extends what a reviewer can cover, but responsibility for the code and the decision to accept it remain with the professional. An AI finding is a signal to examine, not a verdict to apply without judgement.

10.3 The specific risk of generated code

It is worth recalling the context that lends urgency to this competency: a significant proportion of AI-generated code introduces vulnerabilities, and the sense of speed does not always correspond to a real improvement in productivity. That is why verification (chapter 3) and the trust boundary

(chapter 4) apply with particular force to code: what is generated is reviewed with the same or greater rigour than what is written by hand.

What you should be able to do

Having mastered this chapter, a professional is able to:

- Generate, with AI, test cases aligned with acceptance criteria and risks, covering negative scenarios and edge cases.
- Validate that the generated tests attack the real risks, without confusing quantity with quality.
- Use AI as support in code review with a focus on security: inputs, errors, permissions, side effects.
- Keep technical judgement and the final decision on the code with the professional, not with the AI.
- Apply verification and the trust boundary with particular rigour to AI-generated code.

Common mistakes and antipatterns

Confusing the number of tests with quality. Many tests that do not attack the real risks give false confidence.

Accepting generated code without rigorous review. A relevant proportion introduces vulnerabilities; the review must be as demanding as, or more than, for manual code.

Treating the AI's findings as verdicts. They are signals to examine with technical judgement, not automatic decisions.

Being carried away by the sense of speed. The perception of going faster does not equate to a real improvement; it is best to measure, not assume.

Further reading

· OWASP Top 10 for LLM Applications

· Anthropic — Building effective agents