

Empiricism at Machine Speed

Part 2 - Scrum Was Built for This: The Theoretical Case for Empirical AI Governance

Matthew Hodgson, CEO, Zen Ex Machina & PST
Dave West, CEO, Scrum.org

part 2 of 5

Executive Summary

Scrum was designed for complex problems: work where the right answer cannot be specified in advance and where quality is a judgment rather than a test result. Agentic AI knowledge work is that kind of work. The IT governance frameworks most enterprises rely on were built for a different job, and Paper 1 named three questions they leave open: execution governance, value governance, and diagnostic governance.

This paper makes the theoretical case that Scrum answers all three, and says where the case stops.

1. Enterprise IT governance frameworks tell a board what to oversee. They do not tell a team how to govern work whose right answer only emerges as the work is done.
2. A complex product still contains simple tasks. Cynefin, a sense-making framework that sorts problems by how far cause and effect can be known in advance, lets a team classify each backlog item before any agent acts.
3. Transparency, inspection, and adaptation work as one chain, each making the next possible. Transparency answers the execution question, inspection the value question, and adaptation the diagnostic question.
4. Adaptation happens at three levels. The third, triple-loop learning, questions the governance framework itself, and it needs an accountable human to make the call.

The argument is theoretical. Paper 3 tests it against a single-case experimental design, tracked Sprint by Sprint.

The Problem Scrum Was Built For

Scrum's authors define it as "a lightweight framework that helps people, teams and organizations generate value through adaptive solutions for complex problems" (Schwaber & Sutherland, 2020). Snowden and Boone (2007) describe complex problems as those where cause and effect can be understood only in retrospect, so the right answer emerges from doing the work and cannot be planned before it. Scrum meets that condition with empiricism: knowledge comes from experience, and decisions rest on what has been observed. The Scrum Guide puts the premise plainly: "In complex environments, what will happen is unknown" (Schwaber & Sutherland, 2020).

Agentic AI knowledge work sits squarely in that condition. When an agent drafts a regulatory briefing or a strategic recommendation, quality depends on judgment, and the right version often becomes clear only once there is a draft to react to. No specification written beforehand fully describes it.

The governance frameworks most enterprises already use were built for a different job. ISO/IEC 38500 gives directors principles for evaluating, directing, and monitoring an organization's use of IT (International Organization for Standardization, 2015). COBIT 2019 sets out governance and management objectives for enterprise information and technology (ISACA, 2019). Both operate at the level of the board and the enterprise. Neither describes how a team should govern work whose right answer is unknown when it starts, and neither was written with autonomous agents inside the team.

That leaves three questions open, which Paper 1 set out in full. Execution governance asks who is accountable for the work an AI system produces, and whether each output can be traced to the instructions that produced it. Value governance asks who decides whether the AI team is working on the right problem at all. Diagnostic governance asks how a well-governed AI team exposes problems in the organization it works within. Each of Scrum's empirical pillars answers one of these questions. Before any of them applies, though, a team has to know what kind of problem each piece of work is.

Classify the Work Before Agents Act

Calling agentic knowledge work complex describes the product, not every item on its backlog. A complex product still contains routine tasks, such as formatting a reference list or compiling a status summary. It also contains analytical ones, where the answer is knowable but takes expert work to find. Snowden and Boone (2007) argue that leaders need to recognize which kind of situation they are in and change how they respond, because an approach that works in one domain fails in another. Agentic teams face the same demand, item by item.

Skip the classification and agents treat every item as straightforward. They go directly to execution, and the cost shows up as wasted effort, misdirected spend, and outputs that miss in ways a short piece of upfront analysis would have caught. The reverse mistake is just as expensive. Treat every item as complex, demand options analysis for routine tasks, and the team burns capacity the Sprint cannot spare.

Scrum already has a place for this decision. In Sprint Planning, the Developers plan how each selected item will become an Increment, and "how this is done is at the sole discretion of the Developers" (Schwaber & Sutherland, 2020). Cynefin gives that decision a vocabulary: classify the item first, then choose the approach that fits its domain. The domain also shapes how agents should coordinate. Table 1 shows the formation that suits each domain in the operational case Paper 3 describes.

Table 1. Problem domain and the AI team formation that suits it

Domain	What it looks like	AI Formation
Clear	Cause and effect are known; established practice applies	Sequential handoff - defined steps run in order
Complicated	Cause and effect are knowable, but need analysis	Parallel breakout -several options analyzed at once
Complex	The right answer emerges from the work and cannot be specified first	Collaborative convergence: agents work the problem together until understanding emerges
Chaotic	No stable cause-and-effect relationship; act first, then sense	Human judgment; the empirical loop has nothing stable to iterate toward

In a 25,000-task experiment on multi-agent systems, Dochkina (2026) found that, among strong models, the quality difference between the best and worst coordination protocol run on the same model was 44%. The difference between 2 strong models run on the same protocol was about 14%. In that study, how the agents were organized mattered more than which model they ran on. Its limits are set out at the end of this paper.

The chaotic row is a design position, not an observation: the operational case has not yet met a failure severe enough to call for a chaotic-domain response, so the guidance in that row comes from reasoning alone.

Once the team knows what kind of work it holds, the next problem is seeing that work being done.

Transparency Makes Execution Governable

Scrum requires that “the emergent process and work must be visible to those performing the work as well as those receiving the work” (Schwaber & Sutherland, 2020). Transparency belongs to the artifacts, and each artifact carries a commitment that gives its quality something to be measured against: the Product Goal for the Product Backlog, the Sprint Goal for the Sprint Backlog, and the Definition of Done for the Increment.

In a human team, informal visibility makes up for weak artifacts. A Product Owner can ask a question, sit in on a working session, or read a draft. The channel is patchy, but it exists. An agentic team takes it away. When an agent carries a backlog item from start to finish in a single session, there is no working session to sit in on and no draft in progress to read. Execution stays hidden unless the system is built to expose it.

Execution governance therefore needs two kinds of transparency. Output transparency shows what the agents produced and how they got there; in the operational case, workflow state and token logs record every execution step and what it cost. System transparency shows which instructions governed the work. Each agent works from versioned instruction files, so the logic behind any output can be inspected at any stage.

Output transparency alone falls short. It produces visible artifacts with no traceable logic behind them, and in knowledge work “it looks right” is not a quality standard. The Definition of Done is what makes an Increment interpretable as well as visible, because it describes the quality measures the Increment has met (Schwaber & Sutherland, 2020).

Together, the two answer the execution governance question: every output traces to an instruction, and every instruction is a human-authored document under version control. Visibility is only the precondition, though. Someone still has to look, and at agentic volume nobody can look at everything.

Inspection Works at Two Levels

A Scrum Team of typically 10 or fewer people (Schwaber & Sutherland, 2020) produces Increments at human pace across a Sprint. An agentic team with clear instructions and a clear Sprint Goal can produce the same volume in hours. The Product Owner cannot read all of it, and Scrum does not ask them to. What the Product Owner needs is enough transparency to see how value is developing against the Product Goal. Inspection therefore splits into two levels: the quality of individual outputs, which can be delegated, and the direction of the product, which cannot.

Output quality: delegation calibrated to risk

For lower-stakes artifacts such as internal briefings, draft content, and research summaries, a Product Owner can delegate quality decisions to the Developers, provided the outputs meet the acceptance criteria and the Definition of Done. For higher-stakes outputs such as client-facing content, regulatory submissions, and strategic recommendations, the Product Owner makes the release call, and the governance design supplies the evidence it rests on.

Delegation grows as the team earns it. Appelo (2010) treats delegation as a matter of degree, with authority extended step by step as competence is demonstrated. The same holds here: authority moves to the team on operational evidence, not on assumption. Takeuchi and Nonaka (1986) saw a similar pattern in the strongest product development teams they studied and called it subtle control. Management set demanding goals and broad constraints, then left the method to the team.

Delegation is only as safe as the checks behind it, and recent research helps here. Kwok et al. (2026) treat verification, the ability to judge whether an output is correct, as a way of scaling AI performance in its own right, separate from pre-training, post-training, and test-time compute. Across coding, robotics, and medical decision-making, they found three design choices that improve verification accuracy. Splitting one assessment into several domain-specific criteria separates good outputs from bad ones better than a single overall judgment. Running several assessment passes and combining the results smooths out random variation. Graded scores distinguish between outputs of different quality more finely than pass-or-fail verdicts.

All three concern how verification is designed. None depends on which model performs it, and none is new to a well-run Scrum Team. A Definition of Done that specifies quality across several criteria is criteria decomposition. A layered review pipeline, of the kind the operational case uses, is repeated evaluation. A findings model that returns severity ratings and a verdict is graded assessment.

Direction: the Sprint Review

Output checks cannot tell anyone whether the team is working on the right thing. That is the value governance question, and it needs a different kind of inspection.

The Agile Manifesto's principle of sustainable pace was written for human teams at human tempo (Beck et al., 2001). In an agentic system the risk runs the other way. The agents will not burn out, but their output will outrun the human's capacity to take stock and redirect, because continuous production has no natural pause.

The Sprint Review provides one. Its purpose is to inspect the outcome of the Sprint and determine future adaptations, and the Product Backlog may be adjusted as a result (Schwaber & Sutherland, 2020). Once per Sprint, the Product Owner and stakeholders step back from the output and ask whether the Sprint moved the product toward its Product Goal, whether the product still addresses the right problem, and what the environment now requires that it did not a Sprint ago. Those questions need human judgment. Without a fixed point at which to ask them, the Product Owner loses the ability to steer, because the work never stops long enough to be steered.

That answers the value governance question: direction is inspected on a fixed cadence, against the Product Goal, by the person accountable for value. What inspection cannot do by itself is change the standard it inspects against.

Adaptation Runs at Three Levels

Kwok et al. (2026) show that well-designed criteria improve verification, but they hold those criteria fixed for each benchmark. Their work does not address which criteria suit a given domain, when those criteria should change, or how to revise them from what inspection reveals. An organization in a moving regulatory and competitive environment cannot hold its criteria constant. Something has to revise them, and who holds that authority decides whether adaptation happens at all.

Argyris and Schön (1978) distinguished two levels of organizational learning. Single-loop learning corrects an output within the existing rules. Double-loop learning changes the rules themselves. Flood and Romm (1996) added a third level, triple-loop learning, which questions the assumptions that produced the rules. Each level sits at a different point in Scrum and needs a different decision-maker in an agentic team (Table 2).

Table 2. What changes at each learning level, and where it happens in Scrum

Level	Trigger	What changes & where	Who decides
Single loop	An output fails a quality check	The output, during the Sprint	The agent, inside its instructions
Double loop	The same failure appears across several Increments	The governing instruction, at the Sprint Retrospective	The team, with the Product Owner approving
Triple loop	Neither output correction nor instruction revision resolves it	The governing framework itself, questioned at the Sprint Retrospective	The Product Owner alone

Revising the instructions

Human Developers adapt their plan through conversation. Agents cannot; their range of action is bounded by their instruction files. Single-loop correction happens inside those bounds: an output fails a check and the agent revises it. Double-loop learning happens at the Sprint Retrospective, where the Scrum Team inspects how the last Sprint went “with regards to individuals, interactions, processes, tools, and their Definition of Done” (Schwaber & Sutherland, 2020). In an

agentic team, the processes under inspection include the instruction files. When a run of failures exposes a flaw in the governing instructions, the file is rewritten and the change is dated.

The Scrum Guide notes that “adaptation becomes more difficult when the people involved are not empowered or self-managing” (Schwaber & Sutherland, 2020). For an agentic team, authority over the instruction files is what empowerment means in practice. A team that cannot revise its agents’ instructions can see the difference between what an instruction produces and what is needed, and has no way to act on it. Instruction files therefore have to be writable by the team, held under version control, and treated as governance artifacts in their own right. A team that holds Retrospectives without that authority has inspection without adaptation, and the Retrospective turns into a compliance exercise.

The same record answers the diagnostic governance question, because every revision states what needed to change and why. When the cause sits inside the team’s instructions, the team fixes it. When it sits outside them, in an approval path, a data restriction, or a policy the team cannot touch, the dated record becomes evidence about the organization rather than the AI system. That is the diagnostic signal Paper 1 described.

When the framework is the problem

Double-loop learning assumes the framework is sound and only the rules inside it need to change. Sometimes the framework is the problem. If the nature of the work shifts, a team can classify each item correctly and revise its instructions faithfully and still get the wrong results, because the framework those rules sit inside no longer fits. That failure needs the third loop.

Triple-loop learning is harder to pin down than the other two. Reviewing the literature, Tosey et al. (2012) found that the term does not appear in Argyris and Schön’s published work, that its conceptualizations vary widely and often have little theoretical grounding, and that empirical support for it is lacking. They also caution against assuming that higher levels of learning are always better. Triple-loop questions should therefore be rare, reserved for failures the first two loops cannot resolve, and they need a concrete place to be asked.

Scrum supplies that place. At each Retrospective, the Scrum Guide asks the team to identify the “assumptions that led them astray” and explore their origins (Schwaber & Sutherland, 2020). An agentic team is well placed to do this, because its governing assumptions are written down in versioned instruction files, where they can be read, dated, and challenged. Cynefin gives the question a starting point: is this problem the kind it is being treated as, and is the framework the right response to it? That is the classification question from earlier in this paper, asked of the framework instead of a backlog item.

The team can raise the question. Only the accountable human can answer it, because changing the framework changes what everyone else is accountable to. Triple-loop learning without human judgment produces change without accountability.

Where the Theory Runs Out

A theoretical case that does not name its limits is advocacy: it argues for a position instead of testing one. Four limits are structural to this argument, and a fifth concerns the external evidence it cites.

The alignment is untested where the framework is set externally. The operational case behind this series runs without external regulatory mandates. Regulated deployments face constraints the theory has not yet met, particularly where a compliance function cannot be embedded in the flow of work.

The triple-loop mechanism stalls without an empowered human. It depends on someone who can answer the governing-assumption question. In organizations bound by statutory process, the accountable person often cannot act on the answer, and the mechanism then produces a question with no change behind it.

Cynefin is a practical sense-making tool, not a complete account of problem types. Beyond some level of complexity its four domains stop discriminating usefully, and where that level sits matters for any argument built on them, this one included.

The Scrum Guide defines the Scrum Team as a small team of people with no sub-teams or hierarchies (Schwaber & Sutherland, 2020). Placing AI agents inside the Developer accountability is an extension of the Guide; nothing in its text already provides for agents. This series proposes that extension and tests it in a single-case experimental design, tracked Sprint by Sprint. Whether it holds more widely remains an open question.

Dochkina's (2026) 44% figure compares the best and worst of the 4 protocols the study reports, run on the same model; the 14% figure compares 2 models run on the same protocol. The experiment used synthetically generated tasks scored by AI judges rather than human evaluators, reports 4 of the 8 protocols tested, and is a single-author preprint that has not been peer-reviewed. The study itself says its findings need validation against real business workflows, such as regulatory analysis and incident response, before they can be assumed to hold there. Kwok et al. (2026) is also a preprint, although this paper uses only its design findings.

These are the conditions under which the theory would need revision rather than extension. Testing them requires evidence from regulated environments, from organizations bound by statutory process, and from more than one team.

Conclusion

Scrum fits agentic AI governance for a reason that has nothing to do with AI. It describes how to work when the right answer cannot be known before the work starts, and agentic knowledge work is that kind of problem. Enterprise governance frameworks tell a board what to oversee. Scrum tells a team how to govern the work as it unfolds.

Each backlog item has to be classified before agents act on it. Execution has to be exposed deliberately, because agents leave no informal trail. Instruction files have to be governance artifacts the team is authorized to revise. And an accountable human has to be able to question the framework itself. Papers 3 and 4 document these adaptations, and the framework transfers only with them in place.

The evidence for that claim comes from a single-case experimental design, tracked Sprint by Sprint. Paper 3 tests the theory against twelve months of that operation and sets out what the evidence can and cannot show.

About the Authors

Matthew Hodgson is co-founder and CEO of Zen Ex Machina, a diagnostic-first executive strategy and advisory firm operating across government and corporate Australia and the Asia-Pacific region. He is a Scrum.org Professional Scrum Trainer and has led complex organizational transformations across Australian federal agencies and corporate environments for over 20 years. He is the author of *Evolve: The Operating Model AI Demands*.

Dave West is CEO at Scrum.org, The Home of Scrum. A frequent keynote speaker and widely published author, he co-authored *The Nexus Framework for Scaling Scrum* and *Head First Object-Oriented Analysis and Design*. He led development of the Rational Unified Process at IBM/Rational, managed Ivar Jacobson Consulting for North America, and served as VP research director at Forrester Research. Before Scrum.org, he was Chief Product Officer at Tasktop, helping grow it into a VC-backed company of nearly 100.

About Scrum.org

Scrum.org, the Home of Scrum, was founded by Scrum co-creator Ken Schwaber as a mission-based organization to help people and teams solve complex problems. We do this by enabling people to apply Professional Scrum through training courses, certifications and ongoing learning all based on a common competency model.

References

- Appelo, J. (2010). *Management 3.0: Leading agile developers, developing agile leaders*. Addison-Wesley.
- Argyris, C., & Schön, D. A. (1978). *Organizational learning: A theory of action perspective*. Addison-Wesley.
- Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). *Manifesto for agile software development*. Agile Alliance. <https://agilemanifesto.org>.
- Dochkina, V. (2026). *Drop the hierarchy and roles: How self-organizing LLM agents outperform designed structures* [Preprint]. arXiv:2603.28990. <https://arxiv.org/abs/2603.28990>
- Flood, R. L., & Romm, N. R. A. (1996). *Diversity management: Triple loop learning*. Wiley.
- International Organization for Standardization. (2015). ISO/IEC 38500:2015: *Information technology: Governance of IT for the organization*. ISO. <https://www.iso.org/standard/62816.html>
- ISACA. (2019). *COBIT 2019 framework: Introduction and methodology*. ISACA. <https://www.isaca.org/resources/cobit>
- Kwok, J., Li, S., Atreya, P., Liu, Y., Jiang, Y., Finn, C., Pavone, M., Stoica, I., & Mirhoseini, A. (2026). *LLM-as-a-Verifier: A general-purpose verification framework* [Preprint]. arXiv:2607.05391. <https://arxiv.org/abs/2607.05391>
- Schwaber, K., & Sutherland, J. (2020). *The Scrum guide: The definitive guide to Scrum: The rules of the game*. Scrum.org. <https://scrumguides.org>
- Snowden, D. J., & Boone, M. E. (2007). *A leader's framework for decision making*. Harvard Business Review, 85(11), 68–76.
- Takeuchi, H., & Nonaka, I. (1986). *The new product development game*. Harvard Business Review, 64(1), 137–146.
- Tosey, P., Visser, M., & Saunders, M. N. K. (2012). *The origins and conceptualizations of 'triple-loop' learning: A critical review*. Management Learning, 43(3), 291–307. <https://doi.org/10.1177/1350507611426239>