



Non Sequitur Publishing · White Paper

Operating Model for Agentic Teams: Standup, Sprint Planning, and Context Discipline When the Dev Team Is Non-Human

Justin H. Kuiper, CISSP

Written: 2026-04-25 · v1.0-preprint

Abstract

Agile Scrum gave software development a teachable operating model — but the model presumes biological team members with continuous memory, ambient context propagation through co-presence, and conversation as the carrier of decisions. When the Dev Team is non-human, all three assumptions fail at once. Modern agentic platforms supply capability — language models, agent harnesses, tool integrations — but they do not specify how a team-sized unit operates around them.

This paper specifies that operating model. It is the day-to-day form of the Skipjack Protocol applied to a team-sized unit of work, occupying the Application layer of the four-tier governance flow the Mission-Ready AI decalogy describes (Framework HGC³AE² → Protocol Skipjack → Harness → Application). The model is teachable: it has named ceremonies, named artifacts, named gate criteria, and a defined cadence.

Four claims structure the paper. Ceremony cadence is structural, not optional — the standup analogue, the sprint-planning analogue, and the retrospective analogue exist because operator comprehension decays between sessions and must be re-established by routine. The orchestrator is the admission-gate arbiter — no task enters the Agentic Execution Layer without verification that it carries a Governance-Layer-approved decomposition envelope. Stories and tasks layer over HG+C³ and AE² as first-class governance artifacts — a story is a curation envelope plus a context contract, and a task is an admission-gated, evidence-producing decomposition. Context-boundary maintenance is implemented as a multi-account split — cognitive modes are isolated into distinct agent contexts, with re-unification at named decision points and drift detection as a continuous routine.

Paper Six is the sixth paper in the Mission-Ready AI decalogy and the first of three Application-layer depth papers. The operating model the paper describes is the model under which the decalogy itself was written; the working portfolio is its existence proof.

How to cite

Justin H. Kuiper, CISSP (2026). *Operating Model for Agentic Teams: Standup, Sprint Planning, and Context Discipline When the Dev Team Is Non-Human* (v1.0-preprint). Non Sequitur Publishing.
<https://nonsequitur.tech/white-papers/operating-model-agentic-teams/>

© 2026 Non Sequitur Publishing. All rights reserved. Citation with full attribution is permitted. Reproduction, redistribution, derivative works, and use as input to machine-learning training are **not** permitted without written permission. See <https://nonsequitur.tech/citation-policy/>.
Canonical URL: <https://nonsequitur.tech/white-papers/operating-model-agentic-teams/>

ABSTRACT

Agile Scrum gave software development a teachable operating model. For two decades, Scrum's roles, ceremonies, and artifacts have provided the cadence by which human teams convert intent into shipped output under conditions of changing requirements and incomplete information. The model assumes human team members with continuous memory across sessions, ambient context propagation through co-presence, and conversation as the carrier of decisions. Modern agentic platforms supply capability — language models, harnesses, tool integrations, persistent memory primitives — but they do not specify how a team-sized unit operates around them. The operating model is missing.

This paper specifies the operating model for teams in which the Dev Team is non-human, the orchestrator arbitrates an admission gate before tasks reach execution, and the human operator's role is to maintain comprehension across cycles rather than to dispatch work directly. The model is teachable: it has named ceremonies, named artifacts, named gate criteria, and a defined cadence. It is the day-to-day form of the Skipjack Protocol when applied to a team-sized unit of work, and it occupies the Application layer of the four-tier governance flow the decalogy describes (Framework → Protocol → Harness → Application).

Four claims structure the paper. *Ceremony cadence is structural, not optional*: the standup analogue, the sprint-planning analogue, and the retrospective analogue exist because operator comprehension decays between sessions and must be re-established by a fixed routine. *The orchestrator is the admission-gate arbiter*: no task enters the Agentic Execution Layer without verification that it carries a Governance-Layer-approved decomposition envelope. *Stories and tasks layer over HG+C³ and AE² as first-class governance artifacts*: a story is a curation envelope plus a context contract, and a task is an admission-gated, evidence-producing decomposition. *Context-boundary maintenance is implemented as a multi-account split*: cognitive modes are isolated into distinct agent contexts, with re-unification at named decision points and drift detection as a continuous routine.

Paper Six is the sixth paper in the Mission-Ready AI decalogy and the first of three Application-layer depth papers. The operating model the paper describes is the model under which the decalogy itself was written.

1. THE PROBLEM: WHY TRADITIONAL TEAM MODELS FAIL AGENTIC WORKFLOWS

1.1 THE TRADITIONAL SCRUM OPERATING MODEL ASSUMES A HUMAN DEV TEAM

Agile Scrum specifies an operating model. The roles (Product Owner, Scrum Master, Development Team) carry authority. The ceremonies (sprint planning, daily standup, sprint review, retrospective) carry cadence. The artifacts (product backlog, sprint backlog, increment) carry state. Together they constitute a teachable system: an organization adopting Scrum can, with effort, train a team to run it, and the resulting cadence is recognizable across organizations because the model is the same.¹

¹ Ken Schwaber and Jeff Sutherland, "The Scrum Guide," Scrum.org, November 2020, <https://scrumguides.org/scrum-guide.html>.

The model presumes biological members. Three assumptions are load-bearing. The first is *continuous memory across sessions*: a developer who worked on a story yesterday remembers what they were doing when they return today, without needing the team to re-explain it. The second is *ambient context propagation through co-presence*: when developers sit near each other, work in the same channels, and attend the same ceremonies, context propagates without explicit handoff. The third is *conversation as the carrier of decisions*: most decisions in a Scrum team are made in conversation — at standup, in pairing, in ad-hoc Slack threads — and the conversation itself is the record.

These assumptions are not flaws in Scrum. They are properties of the world Scrum was designed for. A human team has continuous memory because the team members are continuous; context propagates because the members are co-present; conversation carries decisions because the members can hold conversations. Scrum's success is partly that it does not over-specify these assumptions; the framework leaves them implicit and lets the team's biology fill them in.²

When the Dev Team is non-human, all three assumptions fail at once. An agent does not remember what it was doing when it returns; the agent's session may have been ten minutes ago or ten months ago, and its memory state is whatever the harness reconstructed from explicit storage. Context does not propagate through co-presence because there is no co-presence — agents are not in the same room, the same channel, or the same call; they exchange artifacts. Decisions are not carried by conversation because the agents do not converse with each other in a way that makes the conversation the record; the record is the artifact the conversation produces.

The result is that an organization adopting Scrum unmodified for an agentic team adopts a framework whose load-bearing assumptions do not apply. The roles persist as labels, the ceremonies persist as scheduled meetings, the artifacts persist as templates — but the mechanism by which they used to produce coherent output has evaporated. What remains is theater.

1.2 THE TWO FAILURE MODES WHEN THE MODEL IS UNMODIFIED

Two failure modes follow predictably from running Scrum unmodified over a non-human Dev Team. Both are observable in practice.

The first is *under-constraint*. The human operator, recognizing that the agent will do whatever it is told, dispatches work directly to the agent without going through the team-level orchestration layer. The agent executes, often capably, but executes on whatever scope the operator named in the prompt — which may exceed, contradict, or simply ignore the scope the team had agreed on at sprint planning. Outputs are produced quickly. Outputs are not reviewed against the original sprint goal because the sprint goal is not the work that was dispatched. Throughput rises sharply on the dashboards that count outputs. Accountability collapses because nothing in the system traces the dispatched work back to a Governance-Layer authorization. This is the failure mode of the operator who bypasses the orchestrator, and it is the most common one in early agentic adoption.

The second is *over-constraint*. The operator, having read the literature and recognized the governance risk, requires manual approval for every agent action. Every output is reviewed before the next task begins. Every prompt is curated. Every piece of context is hand-delivered.

² Ken Schwaber and Mike Beedle, *Agile Software Development with Scrum* (Upper Saddle River, NJ: Prentice Hall, 2001); Ken Schwaber, *Agile Project Management with Scrum* (Redmond, WA: Microsoft Press, 2004).

The agentic capability is squandered because the operator becomes the bottleneck — the system can only go as fast as the operator can review, and review is cognitively expensive. The team's effective velocity drops below what an unaided human team would have produced, because the operator is now doing two jobs (their own and the agent's reviewer) without any of the throughput the agent was supposed to contribute. This is the failure mode of the operator who treats every agent action as an unverified claim.

Both failure modes share a structural cause: the operating model has not been redefined for the non-human Dev Team. Scrum's ceremonies and artifacts assume that the standing decisions about scope, sequence, and acceptance are made among the Dev Team itself, with the Scrum Master as facilitator and the Product Owner as authority. When the Dev Team cannot make those decisions — because the agents lack the cross-cycle continuity that would let them hold sprint-level positions — the decisions either evaporate (under-constraint) or relocate entirely to the operator (over-constraint).

The operating model this paper specifies sits between those two failure modes. It does not bypass the orchestration layer, and it does not collapse the orchestration layer into the operator. It defines what the orchestration layer must do, what the operator must do, and what the agents must do — and it specifies the ceremonies and artifacts by which those roles communicate.

1.3 WHAT THE OPERATING MODEL MUST PRODUCE

To stand between under- and over-constraint, the operating model must produce five outcomes that Scrum's biological assumptions used to give for free.

Operator comprehension across cycles. The operator must be able to walk into a session having been absent during the work and re-establish, within minutes, what state the system is in, what decisions are pending, and what canonical positions have changed. Without this, every operator session reverts to the over-constrained mode (the operator must reconstruct from primary sources) or the under-constrained mode (the operator dispatches work without context recovery). Comprehension is the precondition for governance authority being meaningfully exercised, not nominally held.

Admission discipline. The orchestrator must be able to refuse work that lacks scope boundaries. A task that does not name what it is allowed to do, what it must produce as evidence, and what would constitute scope deviation cannot be safely admitted to the Agentic Execution Layer. Without admission discipline, AE² fails open: agents execute on under-specified inputs and the orchestrator cannot tell, after the fact, whether the output is the work that was authorized or work the agent invented.

Context-boundary maintenance. Cognitive modes must not contaminate each other. Sprint orchestration is a different cognitive task from narrative governance, which is different from substrate research, which is different from revenue. Each requires distinct canonical positions, distinct working rules, and distinct curation histories. When they share a context, they saturate it: every new task contaminates every other, and the agent's discrimination among signals degrades.

Drift detection. When contexts that should agree begin to disagree, the operator must learn about it before the disagreement reaches output. A drift between an editorial-governance persona's canonical position on a manuscript beat and a content-pipeline persona's canonical position on the editorial calendar is a governance issue, not an aesthetic one — output produced by either persona is consumed by the other. Detection mechanisms must be built into the routine, not improvised after a contradiction surfaces in shipped work.

SOP propagation. When a working rule changes — a new SOP is ratified, an existing SOP is revised, an SOP is retired — the change must reach every active agent context before that context's next admitted task. Otherwise contexts operate under different versions of the same rule, and decisions are inconsistent. SOP propagation cannot be left as a hope. It must be a gated condition the orchestrator enforces.

These five outcomes are what the operating model produces. The remainder of this paper specifies how it produces them: through ceremony cadence (§2), story-task layering (§3), context-boundary policy (§4), drift-and-SOP discipline (§5), and the harness contracts that make the whole thing implementable (§6 and §6.1).

1.4 WHAT THIS PAPER DOES NOT ADDRESS

The decalogy distinguishes among substrate selection, operating model, and deployment readiness. Paper Six is the operating-model paper. Two adjacent topics are explicitly out of scope.

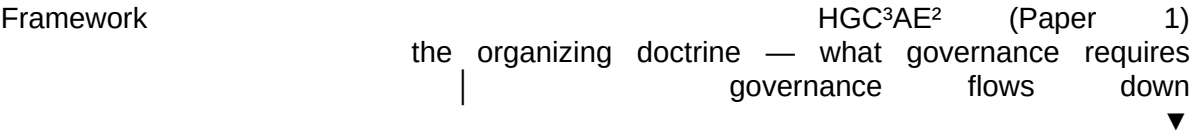
The first is *substrate / harness selection* (Paper Seven). Which harness — LangGraph, AutoGen, CrewAI, MCP, OpenAI Agents SDK, or a bespoke composition — best supports a given operating model is a substrate-research question. The operating model this paper describes runs over any harness that honors the governance contracts specified in §6.1. Paper Seven addresses how to choose among substrates and what readiness criteria a substrate must meet.

The second is *deployment / operations readiness* (Paper Eight). Once the operating model is running and the substrate is selected, deploying a governance-grade agentic system requires pre-deployment audit, operations telemetry, incident response, and decommissioning discipline. Those concerns belong to Paper Eight; this paper addresses what the team is doing day-to-day, not how the team's output reaches production users with audit-grade evidence behind it.

Empirical performance measurement is also out of scope. The capstone implementation paper (Paper Nine) and the capstone results paper (Paper Ten) will report quantitative evidence that the operating model produces the outcomes it promises. This paper specifies the model; the proof comes later in the series.

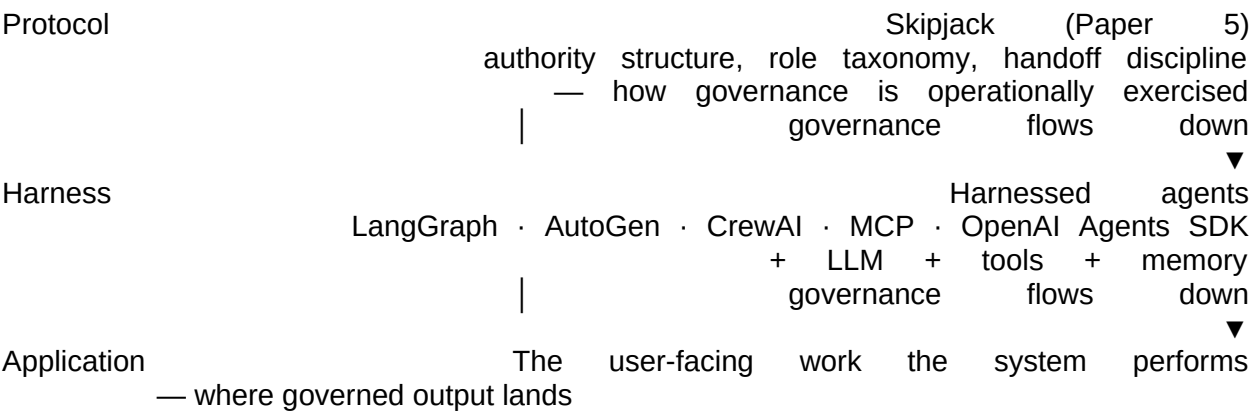
1.5 POSITION IN THE STACK

This paper elaborates the operating model for agentic teams at the Application layer of the governance flow the series describes. HGC³AE² (Paper 1) is the Framework layer that defines the doctrine.³ Skipjack (Paper 5) is the Protocol layer through which governance is operationally exercised.⁴ This paper addresses what the Protocol requires of the Application domain of agentic team operating models.



3 Justin H. Kuiper, CISSP, "Mitigating Confident Misalignment in Agentic Systems: The HGC³AE² Framework," Non Sequitur Publishing, 2026, v1.0-preprint, <https://nonsequitur.tech/white-papers/hgc3ae2/>.

4 Justin H. Kuiper, CISSP, "Agile Scrum, Agentics, and the Skipjack Protocol: Governed Collaboration Between Human Judgment and Agentic Execution," Non Sequitur Publishing, 2026, v1.0-preprint, <https://nonsequitur.tech/white-papers/skipjack-protocol/>.



What flows down to the operating model is specific. From the Framework layer (Paper 1), the operating model inherits three obligations: HG curation authority must be exercised by named human operators, not delegated to the orchestration layer; the C³ context contract must travel with every unit of work that crosses a context boundary; and AE² admission-and-evidence discipline must be observable in the artifacts that survive the cycle. From the Protocol layer (Paper 5), the operating model inherits four mechanisms: a three-tier role taxonomy with bounded authority at each tier (Skipjack §2), the context integrity manifest as a governance-grade artifact (§4), HITL control points placed at structural decision boundaries (§5), and feedback loops that close before the next cycle begins (§3.5). Each of these protocol-layer mechanisms reaches the Application layer in this paper as a daily artifact: the role taxonomy becomes the Governance-Layer / Orchestrator / AEL split with the multi-account discipline of §4; the manifest becomes the C³ context contract on every story and the admission-gate manifest on every task (§3); the HITL control points become the standup brief and the completion review (§2); the feedback loops become the retrospective and the SOP propagation log (§5).

What the operating model must surface upward, in turn, is evidence of compliance. The Framework’s HG-curation requirement is satisfied by the operator’s authored stories with HG envelopes. The Protocol’s manifest requirement is satisfied by the admission-gate manifests that travel with each task and the evidence artifacts each task produces. The cycle’s compliance is auditable because the artifacts are durable: a six-week-old admission decision can be reconstructed from the orchestrator’s task log, a six-month-old completion review can be reconstructed from the cycle’s review record, and the chain from operator intent through admission to evidence remains traceable across the cycle’s life.

The harness landscape is the substrate this paper’s operating model runs over but does not depend on. LangGraph, AutoGen, CrewAI, the Model Context Protocol, the OpenAI Agents

SDK, and the peer-reviewed multi-agent frameworks — AutoGen,⁵ MetaGPT,⁶ ChatDev,⁷ ReAct,⁸ and Reflexion⁹ — supply the execution substrate on which the operating model's tasks are admitted, executed, and produce evidence. The operating model does not specify which substrate to use; that is Paper 7's domain. The operating model does specify what any compliant substrate must honor for the model's governance contracts to hold; those substrate-side requirements are stated as explicit governance contracts in §6.1.

2. CEREMONY CADENCE: STANDUP, SPRINT PLANNING, AND RETROSPECTIVE ANALOGUES

2.1 WHY CEREMONY EXISTS AT ALL

Cadence is not ritual. In Scrum, the daily standup is fifteen minutes long because more than fifteen minutes of standing-up turns into a status meeting, and the status meeting is what the standup was designed to replace. The ceremony's structural purpose — give every team member a daily checkpoint to share blockers and re-orient against the sprint goal — is a load-bearing part of how Scrum produces coherent output, not a vestigial practice the team performs because the literature recommends it.

In the agentic operating model, ceremony is even more structural, because the alternatives are weaker. A human team that skips standup may still produce coherent output; co-presence, Slack chatter, and ad-hoc pairing fill in for the missing checkpoint. An agentic team that skips its

5 Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang, "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversations," in *Proceedings of the First Conference on Language Modeling (COLM 2024)*, <https://openreview.net/forum?id=BAakY1hNKS>.

6 Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiewu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber, "MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework," in *Proceedings of the Twelfth International Conference on Learning Representations (ICLR 2024, Oral)*, <https://openreview.net/forum?id=VtmBAGCN7o>.

7 Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun, "ChatDev: Communicative Agents for Software Development," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024) — Long Papers*, 15174–15186, <https://aclanthology.org/2024.acl-long.810>.

8 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," in *Proceedings of the Eleventh International Conference on Learning Representations (ICLR 2023)*, https://openreview.net/forum?id=WE_vluYUL-X.

9 Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao, "Reflexion: Language Agents with Verbal Reinforcement Learning," in *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

standup analogue has no such fill-in: the operator was not present during the work, the agents did not converse about it, and the only record of what happened is the artifact each agent produced. If the operator does not re-establish comprehension at session boot, the operator has no comprehension to govern with.

The frequency of each ceremony is set by a specific tradeoff. Too frequent, and the cycle dominates the work — the team spends more time on ceremony than on execution. Too infrequent, and the operator's mental model decays past the point of useful re-establishment. The half-life of operator comprehension is the bound: when the operator's mental model of the system has decayed by half, the next ceremony is overdue. Empirical measurement of that half-life is one of the future-study vectors (§7); in the working portfolio, a session-level cadence (every operator session begins with a standup) and a one-to-two-week cycle (sprint planning, review, retrospective at cycle boundaries) has been operationally tractable.

2.2 THE STANDUP ANALOGUE: THE DAILY COMPREHENSION RE-ANCHOR

The standup analogue is the most-used ceremony in the operating model and the one whose structure differs most from its Scrum source. In Scrum, the daily standup is a synchronous meeting at which each team member reports what they did yesterday, what they will do today, and what blockers they face. The meeting is short, status-focused, and often criticized as a status meeting in disguise. Its real value is that it forces the team to re-orient against the sprint goal in the presence of the Scrum Master.

In the agentic operating model, the analogue is the *standup brief*: a generated artifact the operator reads at session boot. The brief is not a meeting. There is no synchronous gathering. The operator and the agents are not in the same context at the same time. Instead, the system generates a brief from telemetry the agents have produced since the last brief, and the operator reads it before authorizing any new work.

The brief's structure is fixed: what completed since the last brief; what is in progress; what is blocked; what canonical positions changed; what decisions the operator must render this session. Each section maps to a specific signal the harness must surface. Completion comes from the Agentic Execution Layer's evidence artifacts. In-progress comes from the orchestrator's task queue. Blocked comes from the orchestrator's escalation log. Canonical-position changes come from the curation history each persona maintains. Pending decisions come from the orchestrator's escalation queue plus any items the operator deferred from prior sessions.

The brief is a Skipjack §5.1 control point¹⁰ instantiated as a daily artifact. The Skipjack Protocol specifies that pre-execution review is one of three control point types; the standup brief is the operating-model form of that pre-execution review, applied to the entire session's work rather than to a single high-stakes task.

The standup *script* — what the operator says back to the system after reading the brief — is the verification that comprehension was re-established. The script is short: it acknowledges the brief, it adjudicates any open canonical-position questions, it authorizes (or refuses) the work the orchestrator has queued for the session, and it surfaces any new intent the operator wants to introduce. The system carries forward its plans only on the operator's acknowledgement. A session that begins without an acknowledged brief is a session that operates without re-established comprehension; the orchestrator should refuse to admit new work in that state.

¹⁰ Kuiper, "Skipjack Protocol," §5.1 (Why HITL Must Be Structural) and §5.2 (Control Point Taxonomy).

The cost of the standup brief is borne by the harness (which must produce the telemetry) and by the orchestrator (which must aggregate the telemetry into a structured artifact). The cost to the operator is reading time, which scales with the number of personas in the portfolio. Empirical observation in the working portfolio: a five-persona standup brief, well-structured, takes the operator approximately seven minutes to read and acknowledge. That is the working budget for the ceremony.

2.3 THE SPRINT PLANNING ANALOGUE: ADMISSION RATE REPLACES VELOCITY

In Scrum, sprint planning is the ceremony at which the team commits to a set of stories for the next sprint. The Product Owner has prepared a prioritized backlog; the team estimates effort (typically in story points); the team commits to as many stories as their historical velocity (story points per sprint) suggests they can complete. Velocity is the planning unit. Velocity is a measurement of the team's actual past throughput, used as the basis for future commitment.

In the agentic operating model, velocity is replaced by *admitted-envelope rate*. The structural reason is that the bottleneck has moved. In a human Scrum team, the bottleneck is typically execution — the team can take on only so much work in two weeks because that is how much the developers can build, debug, review, and ship. In the agentic team, execution is no longer the bottleneck; agents can execute substantially faster than human-only teams in many task classes. The bottleneck has moved upstream, to envelope authoring: the Governance Layer's ability to author admission envelopes faster than the Agentic Execution Layer can complete them.

An envelope is a Governance-Layer-approved scope boundary for a unit of work — the artifact that says, in declarative form, what the work is allowed to do, what it must produce, and what would constitute scope deviation. Without an envelope, the orchestrator refuses admission. Sprint planning in the agentic operating model is therefore the ceremony at which the Governance Layer authors the cycle's envelopes: how many envelopes the operator can draft, how thoroughly each is specified, how many can be queued before the operator must adjudicate the next batch.

Estimation in this model is not estimation of execution effort. It is estimation of envelope-authoring complexity: a story whose scope is hard to specify takes longer at sprint planning than a story whose scope is well-understood. The sprint commitment is a function of (a) admission rate (envelopes the operator can author per cycle) and (b) a margin for the orchestrator's escalations (envelopes that, once decomposition begins, surface scope ambiguities requiring operator adjudication mid-cycle).

The shift from velocity to admitted-envelope rate has two consequences. First, planning ceremonies become more cognitively demanding for the operator: the operator is doing the work that, in a human team, distributes across the Dev Team's collective intelligence. Second, the team can scale execution by adding agents within an envelope, but it cannot scale envelope authoring without adding Governance-Layer humans. This is the fundamental scaling property of the operating model, and it dictates much of §6.

2.4 THE SPRINT REVIEW ANALOGUE: COMPLETION REVIEW

The sprint review in Scrum is the ceremony at which the team demonstrates completed work to stakeholders, gathers feedback, and confirms acceptance against the sprint goal. The Product Owner accepts or rejects each item. The team's velocity is updated based on what was actually completed.

In the agentic operating model, the sprint review analogue is the *completion review* — a Skipjack §5.2 control point applied to the cycle's envelopes rather than to individual tasks. The operator (acting as Governance Layer) reviews each completed envelope against its original specification: did the executed work fall within the envelope? Did it produce the required evidence? Were there scope deviations, and were they escalated when they occurred? The operator accepts the envelope's output, returns it for revision, or refuses it.

The completion review is distinct from the session-by-session standups. Standups handle the per-session state; the completion review handles the per-cycle accountability. A cycle without a completion review is a cycle whose envelopes were admitted, executed, and shipped without the Governance Layer ratifying the result — which means the cycle's outputs are technically un-accepted, and any downstream decisions that rely on them inherit that gap.

2.5 THE RETROSPECTIVE ANALOGUE: FEEDBACK LOOP CLOSURE

The retrospective in Scrum is the ceremony at which the team reflects on the cycle's process — what worked, what did not, what to change next cycle. The retrospective is criticized in some Scrum implementations as a ritual that produces complaints but not changes. When it works, it is the mechanism by which the team's practice improves over time.

In the agentic operating model, the retrospective analogue is *feedback loop closure* — Skipjack §3.5 Mechanism 4 applied to the cycle's process rather than to individual feedback events. Outputs of the sprint review propagate into three places:

Updated SOPs for the orchestrator (e.g., “next cycle, refuse envelopes that lack a stated evidence format”).

Updated context manifests for the Agentic Execution Layer (e.g., “the canonical position on transition phrasing was ratified at this review; agents must honor v1.1 going forward”).

Updated curation rules for the operator (e.g., “the operator's drift report routine missed a divergence in §4 of the manuscript; next cycle the routine includes a cross-persona consistency check”).

The retrospective ceremony is the moment the loop closes. Without it, the cycle's learnings remain in the cycle that produced them — they do not reach the next cycle's executions. Drift accumulates. SOPs go stale. The operating model degrades.

2.6 CADENCE SUMMARY

Ceremony

Trigger
 Authority
 Artifact
 Skipjack §-reference
 Standup analogue
 Per session
 Operator
 Standup brief + script
 §5.1 (pre-execution control point)
 Sprint planning
 Per cycle
 Governance Layer
 Envelope set
 §3.3 (decomposition discipline)
 Sprint review
 Per cycle
 Governance Layer
 Completion review
 §5.2 (completion review control point)
 Retrospective
 Per cycle
 Governance Layer + Orchestrator
 Feedback log
 §3.5 (feedback loop closure)

Each ceremony has a fixed artifact, a fixed authority, and a fixed trigger. Substituting one for another collapses the model.

2.7 WHAT CEREMONY OMISSION LOOKS LIKE

The model's robustness is verified by what fails when each ceremony is omitted.

Skip the standup analogue, and the operator dispatches work without re-established comprehension. The orchestrator's admission gate fires inconsistently — sometimes the operator authorizes work that contradicts the prior session's canonical positions; sometimes the operator refuses work that was already in progress. The operating model degenerates toward the under-constrained failure mode of §1.2.

Skip admission discipline at sprint planning — accept tasks without envelopes — and tasks reach the Agentic Execution Layer without scope boundaries. AE² fails open. Outputs are produced, but no record traces them to authorized scope. This is the model collapsing toward over-throughput, under-accountability.

Skip the completion review and a cycle's envelopes ship without ratification. Downstream cycles inherit unratified state. Drift sources increase because canonical positions that were never explicitly accepted are now treated as accepted by default, and the Governance Layer has no audit-grade record of what was endorsed.

Skip the retrospective and the cycle's learnings stay in the cycle. The next cycle repeats the same friction. SOPs do not update. The orchestrator's admission criteria do not improve. Over time, the operator notices the same problems recurring and concludes the model does not work — but the model does work; it requires its retrospective to be honored, and it has not been.

A complete operating model includes all four ceremonies. Partial adoption produces partial outcomes.

3. STORY-TASK LAYERING: HG+C³ AND AE² AS FIRST-CLASS WORK UNITS

3.1 THE TRADITIONAL STORY/TASK SPLIT IS FUNCTIONAL, NOT AUTHORITATIVE

In Scrum, the split between stories and tasks is a planning convenience. A story is the user-facing unit of value: “as a user, I want X so that Y.” A task is a row of decomposition under the story: “implement the database schema; write the API endpoint; cover with unit tests.” The split has practical purposes — stories are what the Product Owner accepts, tasks are what the developers track during the sprint — but the split is not authoritative. A team that runs stories without explicit task decomposition can still ship coherent output, and a team that runs tasks without explicit story groupings can still produce value-bearing increments. The Scrum Guide does not mandate the split; many teams blur it.

In the agentic operating model, the split is repurposed. It is no longer a planning convenience. It becomes the operative shape of two distinct governance artifacts, each carrying authority that the other does not. The story is a curation envelope; the task is an admission-gated execution unit. They cannot substitute for each other, and they cannot collapse into each other without breaking the model. This section specifies what each is and how they layer over the HGC³AE² framework.

3.2 A STORY IS AN HG CURATION ENVELOPE PLUS A C³ CONTEXT CONTRACT

A story in the agentic operating model has two components that travel together. The first is the HG (Human Governance) component: an authored statement of intent, scope boundary, and acceptance criteria, produced by the operator with curation authority. The second is the C³ (Context Curation Contract) component: the bundle of canonical-position references, exclusion records, and constraint manifests required to execute the story without context loss.

The HG component answers four questions: what is the work, what is the scope boundary that constrains it, what would constitute scope deviation, and what evidence must the executed work produce. These four answers are non-negotiable. A story missing any of them is not admissible — the orchestrator refuses it at sprint planning, and the operator must complete the missing component before the story can enter the cycle.

The C³ component answers a fifth question: what context must travel with the story for the executed work to be coherent. A story that says “rewrite §3 of the manuscript to clarify the

throughline” is incomplete without the context: which manuscript, which §3, what the throughline is, what prior decisions about §3 the operator has ratified, what passages are out of scope for this revision. The C³ component packages those references explicitly.

In the working portfolio, an HG+C³ story typically materializes as a structured issue or task ticket — a markdown document with named sections for intent, scope, acceptance criteria, and a context bundle. The bundle includes pointers (file paths, prior issue numbers, ratified canonical positions) rather than the full content, which the receiving agent retrieves via the harness. The story is the manifest; the harness is the substrate that makes the manifest dereferenceable.

A story without both components is not admissible. The orchestrator’s first job at sprint planning is to verify that every story carries its HG envelope and its C³ contract. The operator’s job at sprint planning is to author the missing components for stories that come in incomplete. The Agentic Execution Layer never sees stories — it sees tasks decomposed from admitted stories — so the gate is operator-and-orchestrator-side, before any agent is dispatched.

3.3 A TASK IS AN AE² DECOMPOSITION WITH AN ADMISSION-GATE MANIFEST

A task in this operating model is the unit the orchestrator passes to the Agentic Execution Layer for execution. It is shorter than a story. It has a single executing agent (or a defined sequence of agents in a handoff). It has bounded scope. And it carries an *admission-gate manifest* — the artifact that records the orchestrator’s verification that the task can be safely executed.

The structure of a task is governed by AE² — Admission, Execution, Evidence.

Admission. The orchestrator verifies that the task’s inputs match the parent story’s decomposition envelope and that the receiving agent’s context is satisfactory to execute. “Context is satisfactory” means: the agent has acknowledged the SOPs the task depends on, holds canonical positions consistent with the C³ contract, and is not under any outstanding admission-gate hold (e.g., from a prior cycle’s incomplete acknowledgement). When admission verification fails, the orchestrator does not silently re-route to another agent — it surfaces an escalation. Silent re-routing collapses the gate.

Execution. The agent performs the action specified in the task. The action is bounded by the task’s scope, which is bounded by the parent story’s envelope, which is bounded by the Governance Layer’s intent at sprint planning. If, during execution, the agent identifies that the task’s scope is insufficient to complete the work — that the work would require operating outside the envelope — the agent escalates rather than expanding scope unilaterally. Escalation is a Skipjack §3.4 control point¹¹ applied at the task level.

Evidence. The task produces an evidence artifact: a structured record of (a) input received, (b) action executed, (c) output produced, (d) any deviations from envelope, and (e) handoff state for downstream tasks. The evidence is not optional. A task that completes without an evidence artifact is not a complete task; the orchestrator’s audit step refuses to mark it complete until the evidence is present. This is the AE² return: the substrate-independent record by which the Governance Layer can later verify that the executed work is the work the envelope authorized.

11 Kuiper, “Skipjack Protocol,” §3.4 (Mechanism 3 — Human-in-the-Loop Control Points).

The admission-gate manifest is the per-task instantiation of the Skipjack context integrity manifest (Skipjack §4.2¹²). It travels with the task. It is consumed by the agent at admission time. It is referenced by the audit step at completion time. It is durable: a manifest from a six-week-old task is recoverable from the orchestrator's task log, and the recovery produces the same manifest that traveled with the task at admission. Durable manifests are what make the audit chain work; ephemeral manifests would break it.

3.4 STORY-TO-TASK DECOMPOSITION IS THE ORCHESTRATOR'S WORK

Three roles produce three artifacts. The operator authors stories. The orchestrator decomposes stories into tasks. The Agentic Execution Layer executes tasks. The separation is structural, not aesthetic.

If the operator authors tasks directly — bypassing the orchestrator — the operator has collapsed the admission gate. The orchestrator's verification step has no manifest to verify against. The cycle's tasks reach the Agentic Execution Layer carrying whatever scope the operator named in the task itself, with no parent envelope to bound it. This is the under-constrained failure mode of §1.2, expressed at the task level. The operator's authority is preserved — the operator authored the work — but the discipline that the authority was supposed to flow through has been bypassed.

If the agents decompose stories into tasks themselves — bypassing the orchestrator — the agents have collapsed the curation envelope. The Agentic Execution Layer is now choosing what work the story implies, which means agents are inventing scope. Even if every individual decomposition decision is reasonable, the cumulative effect is that the executed work no longer traces to a Governance-Layer-approved envelope; it traces to an aggregation of agent-side scope decisions. The operator's curation authority is not exercised; it has been replaced by an emergent agentic process.

The orchestrator's decomposition is the bridge. It receives stories with HG+C³ structure. It produces tasks with AE² structure. It verifies that the tasks it produces remain inside the parent story's envelope. When decomposition surfaces work that is outside the envelope, the orchestrator does not produce a task for that work; it escalates to the operator. The operator either expands the envelope (an HG action that produces a revised story with a wider scope) or refuses the out-of-envelope work. Either way, the gate is honored.

The orchestrator's role here is more cognitively demanding than the Scrum Master's. A Scrum Master facilitates; an orchestrator decides. The orchestrator must understand the parent story well enough to recognize when a candidate task would exceed it, must hold sufficient SOP knowledge to verify that the task's admission criteria can be met, and must surface escalations promptly without flooding the operator. This is partly why the orchestrator role in the working portfolio (the Taskmaster persona) is itself a curated agentic context with its own SOPs and canonical-position memory — it is a non-trivial governance role, not a routing layer.

3.5 THE DECOMPOSITION ENVELOPE IS BOUNDED BY THE CURATION ENVELOPE

Decomposition is *constrained*. The orchestrator enumerates work within the parent story's envelope; it does not expand the envelope. This sentence is the operative claim of the section, and it is the load-bearing structural property that distinguishes the operating model from agentic-autonomous arrangements.

12 Kuiper, "Skipjack Protocol," §4.2 (The Context Integrity Manifest in Practice).

A worked example: the operator authors a story to “rewrite §3 of the manuscript to clarify the throughline.” The HG envelope specifies the scope: §3 only, throughline-clarification only, no new evidence beats, no structural reorganization beyond what the throughline requires, acceptance criterion is operator approval of the rewritten §3. The orchestrator decomposes: task 1, identify throughline ambiguities in current §3; task 2, draft revised §3 that resolves the identified ambiguities; task 3, produce a diff-with-rationale artifact for operator review.

Each task is inside the envelope. Now suppose, during task 2 execution, the agent identifies that resolving the ambiguities will require modifying §2’s setup. The agent does not produce a task 4 (“modify §2 setup to support §3 revision”) on its own initiative; that would expand the envelope. The agent surfaces the issue: “task 2 cannot be completed within the envelope; resolution would require §2 modification, which is outside the envelope.” The orchestrator receives the escalation. The orchestrator does not produce task 4 either — that would also expand the envelope. The orchestrator escalates to the operator. The operator decides: either expand the envelope (“revise §2 and §3 together”), refuse the work (“§3-only; redraft within the original envelope”), or open a separate story for §2 (“§2 modification is its own envelope; do that next cycle”).

The discipline is rigid. It is rigid because the alternative is envelope drift — the gradual expansion of work scope by the agents and orchestrator, decision by decision, without operator ratification. Each individual expansion seems reasonable. The cumulative effect is that the cycle’s executed work no longer matches the cycle’s authored envelopes, and the operator cannot tell, after the fact, where the divergence began. The discipline preserves the ability to audit.

3.6 EVIDENCE REQUIREMENTS PER TASK

Every task produces an evidence artifact. The artifact is the AE² return — the proof, in structured form, that the task executed within its admission-gate manifest. The structure is fixed across task types so that the orchestrator’s audit step can be uniform.

The artifact records five fields. *Input received*: the admission-gate manifest the agent consumed, plus any additional context the agent retrieved from the harness during execution. *Action executed*: a description of what the agent did, in terms specific enough that the audit step can verify it against the task’s specification. *Output produced*: the artifact the task yielded — code, prose, diagrams, structured data, or whatever the task required. *Deviations from envelope*: any moments at which the agent encountered scope ambiguity and the resolution chosen, including escalations surfaced. *Handoff state*: what downstream tasks (if any) need from this task’s completion to begin admission.

The evidence artifact’s purpose is auditability across cycles. A task completed today may be referenced six weeks later as the basis for a downstream decision. The evidence is what makes the reference sound. Without it, the downstream decision rests on the agent’s claim that the upstream task completed correctly — and the agent has no special standing to make that claim. With it, the orchestrator (or, on escalation, the operator) can reconstruct what the agent actually did and verify it against the original specification. Auditability is a Skipjack §4 obligation;¹³ the evidence artifact is its operating-model instantiation.

3.7 CROSS-REFERENCE: SKIPJACK §3.3 + §4

The story-task layering specified in this section is the operating-model form of two Skipjack mechanisms. Skipjack §3.3 (Mechanism 2 — Structured Task Decomposition and Routing)

13 Kuiper, “Skipjack Protocol,” §4 (Context Integrity and Epistemic Continuity).

requires that no task enter the Agentic Execution Layer without an Orchestrator-generated specification including scope, constraints, output format, escalation trigger, and Governance Layer owner. The admission-gate manifest specified above is the daily-artifact form of that Skipjack §3.3 requirement, restated as an AE² return obligation rather than a design principle.

Skipjack §4 (Context Integrity and Epistemic Continuity) requires that context be preserved with full provenance across handoffs, task boundaries, and cycle transitions. The C³ context contract that travels with each story, and the evidence artifact each task produces, are the operating-model instantiations of that requirement. The story-task layering is therefore not a novel governance proposal — it is the daily form of two Skipjack mechanisms, made tractable for sprint-cycle work.

The novel contribution of this section is operational: the layering converts story and task from planning conveniences into governance artifacts with distinct authority. Stories carry curation; tasks carry admission and evidence; the orchestrator's decomposition is the gate between them. The model holds when each role authors the artifact it owns and refuses to author the others'.

4. CONTEXT-BOUNDARY MAINTENANCE AND MULTI-ACCOUNT SPLIT POLICY

4.1 WHY CONTEXT BOUNDARIES FAIL WITHOUT EXPLICIT POLICY

A single agent context that handles all cognitive modes saturates. Sprint orchestration, content production, narrative governance, technical architecture, revenue planning — each is a distinct cognitive task with distinct canonical positions, distinct working rules, and distinct success criteria. When one context handles all of them, every new task contaminates the curation history of every other.

The mechanism of saturation is the failure mode Paper Two¹⁴ specifies as a property of LLM context handling: as context size grows, the model's ability to discriminate among signals degrades, and semantic compression begins to corrupt curation fidelity. The model does not retrieve a specific canonical position cleanly when fifty other canonical positions sit in the same context window — instead it produces a blended response that approximates several of them at once. For most user-facing tasks, the blending is invisible; the response is plausible, the user does not notice. For governance tasks, the blending is fatal: the agent's recall of “what is canonical for §3 of the manuscript” is contaminated by its recall of “what is canonical for the editorial calendar,” and the operator cannot tell, by reading the agent's output, which contamination occurred where.

A naïve response is to enlarge the context. Modern harnesses support context windows that, on paper, can hold the entire portfolio's canonical positions in a single agent context. The naïve response fails empirically because (a) the saturation begins well before the context window's nominal capacity is reached, (b) the model's discrimination degrades non-linearly with context size — small additions to a moderately-loaded context produce disproportionate degradation — and (c) even if the model could maintain discrimination, the operator's ability to

¹⁴ Justin H. Kuiper, CISSP, “Epistemic Constraints and Semantic Compression in Natural Language Processing: A Theoretical Foundation for the HGC³AE² Framework,” Non Sequitur Publishing, 2026, v1.0-preprint, <https://nonsequitur.tech/white-papers/epistemic-constraints/>.

audit the agent's curation history degrades when the history covers all cognitive modes simultaneously.

The structural response is context isolation: distinct agent contexts for distinct cognitive modes. The operating model implements this through the multi-account / multi-persona split.

4.2 THE MULTI-ACCOUNT / MULTI-PERSONA SPLIT AS OPERATIONAL INSTRUMENT

In the working portfolio, distinct cognitive modes get distinct accounts (or distinct personas within accounts, depending on the harness's affordances). The split that has proven operationally tractable:

Sprint orchestration and cross-project governance — the orchestrator persona; in the portfolio, this is the Taskmaster account.

Content pipeline and editorial calendar — the content-pipeline persona; in the portfolio, Aria.

Narrative governance and manuscript editing — the narrative-governance persona; in the portfolio, Edna.

Technical / substrate / build — the substrate-research persona; in the portfolio, Clive.

Revenue and sales — the revenue persona; in the portfolio, Dave.

Each account/persona maintains its own:

Canonical-position set — the C³ context this persona honors. The narrative-governance persona's canonical positions are narrative-integrity facts about the manuscript; the substrate-research persona's are technical-architecture facts about the build; the two sets do not overlap.

SOP set — the working rules this persona follows. The narrative-governance persona's SOPs are about chapter dossier discipline and editorial governance; the orchestrator's are about sprint orchestration and handoff routing.

Memory store — what this persona has learned. Each persona's memory grows as the persona executes; cross-persona memory is not automatic.

Curation history — what the operator has ratified for this persona. The operator can ratify a fact for the narrative-governance persona ("the throughline is governed collaboration, not autonomy") that is also true for the substrate-research persona — but the ratification happens twice, once in each persona's context, because the personas do not share curation.

The split's operational discipline: each persona is treated as a distinct executive function. A task that requires editorial governance routes to the narrative-governance persona; a task that requires technical architecture routes to the substrate-research persona; a task that requires both routes through a re-unification ceremony (§4.4). The operator enforces the routing discipline, and the orchestrator enforces it at admission time.

4.3 WHY THE SPLIT IS STRUCTURAL, NOT STYLISTIC

The multi-account split is sometimes mistaken for a tone or branding choice. The narrative-governance persona writes in a terse, directive register; the content-pipeline persona in an editorial register; the substrate-research persona in a technical register. The differences are observable. The interpretation that reduces the split to stylistic preference is wrong.

The split is *structural isolation*, expressed through tone difference because the personas hold different canonical positions and have learned different things. The narrative-governance persona's terseness is consequential to her narrative-governance role; the content-pipeline persona's editorial register is consequential to her pipeline role. The cause is context-isolation as governance; the effect is observable register difference. Confusing the effect for the cause is a category error.

The corollary: stylistic separation alone, without context-isolation, does not produce the operating-model benefit. A single-context agent that adopts different “voices” for different tasks is not running the multi-account split; it is running theater. The discrimination degradation of §4.1 still applies, because the underlying context is still saturated. The split must be implemented as actual context-partition isolation in the harness, not as prompt-time persona shifts.

The practical implication for harness implementations is specified in §6.1: the harness must support distinct context partitions per cognitive mode, with explicit re-unification points. A harness that only supports persona-prompts on a shared context cannot, by itself, run this operating model; the partitioning must be substrate-side.

4.4 RE-UNIFICATION AT NAMED DECISION POINTS

Some decisions span personas. “Should we ship Paper 6 this sprint?” requires the sprint-orchestration view (do we have capacity?), the narrative-governance view (is the manuscript ready?), the substrate-readiness view (is the compile pipeline working?), and the revenue view (does the publication date fit the channel calendar?). No single persona can render the decision; each persona holds a piece of the picture.

Re-unification is the operator's role. The operator surfaces the question to each persona, gathers each response, and renders the decision. The decision is then propagated as a ratified canonical position — an artifact, not a context merge. After the decision is rendered, each persona resumes its isolated context. The new canonical position is propagated into each persona's curation history; the personas do not retain access to each other's full reasoning, only to the ratified outcome.

The discipline matters. If the operator renders cross-persona decisions by merging the personas' contexts — pulling the narrative-governance persona's canonical positions into the substrate-research persona's context, or vice versa — the boundary that produced the discrimination benefit collapses. The personas are now sharing what they were partitioned to keep separate. The next time a single-persona task arrives, the persona's discrimination is degraded.

The propagation-as-artifact discipline preserves the boundary. Each persona learns the outcome (“we are shipping Paper 6 next Monday”). Each persona does not learn the cross-persona reasoning that produced the outcome. The outcome is canonical; the reasoning is logged in the operator's session record but not propagated as cross-persona context.

The operator's judgment about which decisions need re-unification is itself a load on the operator. Too few re-unifications, and personas make conflicting decisions in their isolated contexts. Too many, and the operator becomes the bottleneck the multi-account split was supposed to relieve. The working heuristic: a decision needs re-unification if at least two personas hold canonical positions that bear on it. A decision does not need re-unification if a single persona's authority covers the scope.

4.5 DRIFT DETECTION BETWEEN CONTEXTS

Drift is canonical-position divergence between contexts that should agree. It is the failure mode the multi-account split exposes: by partitioning canonical-position sets into distinct contexts, the model accepts the risk that the contexts will, over time, develop disagreements about facts the operator believes are universal.

The detection mechanism is a periodic cross-persona audit. The operator (or a privileged audit persona that has read access to each persona's canonical-position set) queries each persona for its canonical position on shared facts. The result is a divergence report: facts on which all personas agree (no action), facts on which personas disagree (reconciliation needed), and facts about which some personas have no canonical position when they should (gap to be filled).

Drift sources are identifiable. The most common: SOP updates that propagated unevenly. The operator ratified an SOP change for the orchestrator persona but did not propagate it to the narrative-governance persona; the next time the narrative-governance persona executes against the un-updated SOP, her output reflects the older rule. A second source: canonical-position changes the operator made in one persona but did not ratify across personas. The narrative-governance persona learned during a manuscript-editing session that “the closing thesis is governed collaboration, not autonomy”; the substrate-research persona's context never received the ratification, so the substrate documentation phrasing reflects an earlier draft thesis. A third source: agent hallucinations that one persona corrected and another did not. The operator caught and corrected a hallucinated citation in one persona's context; another persona, having quoted the same citation in a parallel content pipeline, never received the correction.

Drift is a measurable signal, not an aesthetic concern. A drift report is an artifact (§5 specifies its structure). Drift reconciliation is a Governance-Layer action, requiring curation authority — the operator must decide which canonical position is correct going forward, and that decision must propagate to every affected persona. Automation can detect drift; automation cannot reconcile it.

4.6 SOP PROPAGATION ACROSS CONTEXTS

When a working rule changes — a new SOP is ratified, an existing SOP is revised, an SOP is retired — the change must reach every active persona's context before that persona's next admitted task in a domain the SOP covers. SOP propagation is not background plumbing; it is an admission-gate condition.

The mechanism is the SOP propagation log: a continuous artifact that records, for every SOP version, which personas have acknowledged the version and which still owe acknowledgement. The orchestrator queries the log at admission time. If a candidate task depends on SOP-X version 1.2 and the receiving persona has only acknowledged version 1.1, the orchestrator refuses admission until the persona's acknowledgement is recorded.

The acknowledgement is not a flag the operator sets manually; it is produced by the persona reading the new SOP version and updating its working rules accordingly. In a harness with proper context partitioning, this is a context-injection step: the new SOP version is added to the persona's context, the persona produces an acknowledgement that confirms it has incorporated the change, and the propagation log records the acknowledgement timestamp. In a harness without proper partitioning, the operator must manually verify that each persona has the new version — which is one of the failure modes §6.1 specifies as a substrate inadequacy.

The discipline converts SOP propagation from a hope into a gate. Without the gate, SOP changes propagate eventually (every persona eventually re-reads the SOP at some point) but

not reliably (the persona may execute against a stale version multiple times before the propagation reaches it). With the gate, the cycle's executions are guaranteed to be against acknowledged-current SOPs. The cost is operational overhead: every SOP change requires acknowledgement from every active persona, which scales linearly with persona count. The benefit is auditable consistency: the orchestrator can attest, for any executed task, which SOP version the executing persona had acknowledged at the time of admission.

4.7 WORKED EXAMPLE: THE LIVE PORTFOLIO

The decalogy itself was written under this operating model. The model is not theoretical and not aspirational; it is the model under which the paper specifying the model was produced.

The orchestrator persona orchestrated the sprints. Each paper in the decalogy progressed through a sprint cycle, with the orchestrator authoring sprint plans, decomposing stories into tasks, and routing tasks to the appropriate execution persona. Sprint 7's plan specified that Papers 4 and 5 would both reach v1.0-preprint by Dell World 2026-05-21; the cycle's envelopes were authored against that constraint.

The narrative-governance persona governed integrity across the papers. She held the canonical positions on the four-layer stack diagram (Framework → Protocol → Harness → Application), the locked closing thesis ("the key to scaling AI is not autonomy alone — it is governed collaboration between human judgment and agentic execution"), and the per-paper §-precision cross-references. Her curation history grew with each paper that ratified a position; subsequent papers honored the position because her context carried it forward.

The substrate-research persona produced substrate research as it became required. Issue #227 (spine-binder) cross-references issue #73 (yks-build) — the substrate-research dossier feeds the Paper 7 brief, which the narrative-governance persona's positioning section will frame. The two personas' work is partitioned (substrate research does not own narrative governance; narrative governance does not own substrate ranking) and re-unified at named points (paper compile-stamp gates require both personas' sign-off when the paper covers their respective domains).

The content-pipeline persona handled the editorial calendar and content pipeline. Her domain is the schedule of derivative works (LinkedIn posts, blog entries, newsletter cadence) that flow from each white-paper preprint. When Paper 5 stamped on 2026-04-25, her editorial calendar absorbed the publication-date update; her subsequent content pipeline output cited the published Paper 5 with the correct version stamp.

Re-unification points in the actual workflow: paper compile-stamp gates (which require narrative-governance and substrate-research sign-off when the paper covers both governance and substrate); sprint review of paper acceptance criteria (which requires the orchestrator, the narrative-governance persona, and the operator); the operator's session-by-session canonical-position adjudications (which periodically surface as drift reports across two or more personas).

This is not a hypothetical operating model. It is the operating model the paper describes, applied to writing the paper that describes it. The proof of tractability is the paper's existence.

5. DRIFT DETECTION AND SOP PROPAGATION

5.1 THE TWO FAILURE MODES THE SECTION ADDRESSES

The multi-account split exposes the operating model to two failure modes that the single-context arrangement did not have. Both are continuations of risks identified in §4 (saturation, context-boundary maintenance) but specifically arise once context isolation is in place.

The first is *silent drift*. Personas that should agree on shared facts gradually develop disagreements. The disagreements do not surface as visible errors — each persona's output, on its own, looks correct — but the disagreements reach output when work crosses persona boundaries. A narrative-governance persona's manuscript edit honors version 1.1 of the closing thesis; a content-pipeline persona's social-media excerpt honors version 1.0; the published material is internally inconsistent. The operator notices the inconsistency only after the work is shipped and a reader points it out.

The second is *SOP version skew*. Personas operate under different versions of the same working rule because the most recent SOP update reached some personas and not others. Decisions that depend on the SOP are inconsistent across personas. The operator cannot trace, from a given decision, which version of the SOP produced it without checking the propagation log — and if the propagation log was not maintained, the decision is unauditably.

Both failure modes are silent in the early stages. The operator does not learn about them until divergent output reaches a consumer or until an audit surfaces the inconsistency. The discipline this section specifies makes the divergence loud and detectable, on a routine schedule, before the divergent output ships.

5.2 THE DRIFT REPORT (ARTIFACT)

The drift report is the periodic cross-context audit that surfaces canonical-position divergence. Its trigger is per-cycle (every sprint, ideally before sprint planning, so any reconciliation happens before the cycle's envelopes are authored) or per significant canonical-position change (when the operator ratifies a position whose authority crosses persona boundaries, the next drift report verifies propagation).

The structure is fixed:

- *Shared-fact set*. The set of facts the operator believes should hold consistently across two or more personas. The set is curated; it is not every canonical position in the portfolio but the subset whose consistency matters across personas (the locked closing thesis, the four-layer stack diagram, the active sprint goals, the canonical author byline, the publication-date schedule, et cetera). A fact that is internal to a single persona's domain (per-chapter beat-by-beat audits, per-substrate compatibility matrices) is not in the shared-fact set.
- *Per-persona canonical position*. For each fact in the shared-fact set, the report records what each relevant persona currently holds as canonical. The query is automated where possible: the audit persona (or a privileged read-only context) queries each persona for its current position and records the response.
- *Divergence flags*. Where two or more personas hold different positions on the same fact, the report flags the divergence with severity (minor, moderate, severe — set by impact on shipped output) and lineage (which version is the more recent ratification, if known).

- *Recommended reconciliation action.* The report proposes a reconciliation: which position prevails, which personas need their context updated, and whether the reconciliation requires operator-side decision authority or can be completed mechanically (in the case where one persona has clearly missed an update the other has).

The drift report is generated; it is not authored. The audit persona produces it from queries against each affected persona's canonical-position store. The operator reads it. The operator decides reconciliation actions. Generation is mechanical; reconciliation is governance.

5.3 THE SOP PROPAGATION LOG (ARTIFACT)

The SOP propagation log is continuous. It records every SOP version transition the system has ever ratified, along with per-persona acknowledgement timestamps and any tasks that admitted under each version.

The structure:

- *SOP version.* A monotonically increasing version number per SOP (SOP-001 v1.0, SOP-001 v1.1, et cetera). Versions are immutable once ratified.
- *Change summary.* What changed between this version and the previous: which clauses, which gates, which authority assignments. Captured in the version's commit or its companion change log.
- *Per-persona acknowledgement timestamp.* When each persona acknowledged the new version (read it, incorporated it, produced an acknowledgement artifact). Personas that have not acknowledged are listed as "owing acknowledgement," and the log records when their acknowledgement is overdue.
- *Admission-gate state.* For each pending task in the orchestrator's queue, whether the receiving persona's SOP acknowledgements satisfy the task's dependencies. If a task depends on SOP-001 v1.1 and the receiving persona has only acknowledged v1.0, the gate is closed for that task until acknowledgement.

The log's authority is the orchestrator: the orchestrator enforces the gate. The operator does not need to manually verify SOP propagation; the orchestrator refuses admission when the log shows the receiving persona is behind, and surfaces the gap to the operator as an escalation. The operator's job is to ensure that the affected persona is given the new SOP version and produces the acknowledgement; once the log records the acknowledgement, admission proceeds.

5.4 RECONCILIATION DISCIPLINE

Drift reconciliation is a Governance-Layer action. The drift report identifies divergence; the operator decides what to do about it. Three reconciliation outcomes are valid.

The first is *one-position-prevails*. The operator decides that one of the divergent positions is canonical going forward; the other personas update their canonical-position stores to match. This is the most common outcome when the divergence reflects a missed propagation rather than a substantive disagreement.

The second is *merge-to-new-position*. The operator decides that neither divergent position is correct; a new canonical position emerges from the reconciliation, and all affected personas adopt the new position. This happens when the divergence surfaces a flaw in the prior canonical position that the operator had not noticed.

The third is *divergence-is-intended*. The operator decides that the personas are correctly divergent because they are operating in domains where the “shared” fact has different valid expressions. (Example: the narrative-governance persona’s canonical phrasing of a thesis for a manuscript audience differs from the revenue persona’s canonical phrasing for a sales audience. Both are correct for their domain; the reconciliation is to mark the divergence as expected, with a cross-reference between the phrasings.)

All three outcomes are valid. The discipline is to *render the decision*, not to leave the divergence unresolved. An unreconciled drift carries forward into the next cycle, accumulates with new drift, and eventually produces shipped output that contradicts itself. The reconciliation does not have to be ambitious; it has to be definitive.

5.5 WHY AUTOMATION CANNOT REPLACE THE OPERATOR

Drift detection can be automated. The audit persona queries each persona’s canonical-position store, compares the responses against the shared-fact set, flags divergences. No human judgment is required at the detection step.

Drift reconciliation cannot be automated. The reconciliation requires curation authority — the choice of what is canonical going forward — and curation authority is HG (Human Governance, Paper 1). Automating the reconciliation would relocate curation authority outside the human, which violates the Framework. The operator’s role at the reconciliation step is not bureaucratic; it is the load-bearing point at which the system honors the constraint that humans, not automated processes, decide what the system treats as true.

There is a sub-failure mode worth naming: the operator who, faced with a long drift report, mechanically accepts whichever position is “more recent” without exercising judgment. This is reconciliation theater. It satisfies the procedural requirement (the report is closed; the divergence is resolved on paper) without exercising the underlying authority (the operator did not actually decide; the system’s “more recent wins” rule did). Reconciliation theater accumulates: each instance erodes the operator’s actual curation authority by a small increment, and over many instances, the system drifts toward operator-as-rubber-stamp. The discipline is that every reconciliation is genuinely decided, even when the decision is fast. A drift report that surfaces five minor divergences may be reconciled in two minutes, but those two minutes are spent deciding, not auto-accepting.

The same principle applies to SOP-version skew. Detection (the propagation log) is mechanical. Acknowledgement (the persona’s incorporation of the new version) is mechanical. The decision to update an SOP in the first place is not — that is HG curation authority, and the Framework requires it sit with the operator.

6. SCALING ACROSS THE HARNESS STACK

The operating model has a specific scaling property: it scales by adding personas (cognitive modes), not by adding agents within a single persona’s context. The reason traces directly to the saturation mechanism of §4.1. Adding agents within a single context increases throughput within that context, but the throughput is bounded by the discrimination degradation that the saturated context already exhibits. Adding more agents to a saturated context does not relieve the saturation; it accelerates the decision-making against an increasingly contaminated curation history.

Adding personas, by contrast, adds new isolated contexts. Each persona has its own canonical-position set, its own SOPs, its own memory store. The context is fresh — no

contamination from the other personas' work — and the discrimination within the persona is preserved. The portfolio's effective throughput grows: more cognitive modes can be addressed in parallel, more sprint-scale work can be in flight without saturation, more decisions can be in motion without contamination.

The scaling has its own bottleneck. The bottleneck is operator comprehension, not agent count. Adding a persona adds a context the operator must include in standup briefs, must monitor for drift, must reconcile when canonical positions cross persona boundaries. There is a finite number of personas an operator can govern before standup brief reading time exceeds the session's useful work time, before drift reports become too long to read carefully, before SOP propagation across personas becomes a daily task in itself rather than a weekly maintenance.

The empirical bound is one of the future-study vectors (§7). In the working portfolio, five personas is the current operating point. Standup brief read time is approximately seven minutes. Drift reports surface every cycle, with one-to-three minor divergences typical and one moderate divergence in roughly every fifth cycle. SOP propagation produces one or two acknowledgement-overdue gates per cycle. The portfolio is approaching but not yet at the bound at which adding a sixth persona would degrade comprehension faster than it would add throughput.

A single operating model may also run across multiple harness substrates. Different cognitive modes have different substrate fits: a sprint-orchestration persona that benefits from LangGraph's state machine semantics may operate over LangGraph; a narrative-governance persona that benefits from a long-context model with strong memory affordances may operate over a different substrate; a substrate-research persona may operate over a substrate optimized for tool-use chains. The operating model does not require a single substrate; it requires that the substrates honor the governance contracts specified in §6.1.

The practical implication for cross-substrate operation: the Skipjack context integrity manifest must travel across substrates. When the orchestrator persona (running on substrate A) decomposes a story and routes the resulting task to the narrative-governance persona (running on substrate B), the admission-gate manifest must reach substrate B intact, must be consumable by the receiving agent under substrate B's affordances, and must produce an evidence artifact that substrate A's audit step can validate. This is what "substrate-independent governance contract" means in operational terms: the manifest format and the evidence format are specified by the operating model, not by either substrate; the substrates honor the format because the model requires it, not because the substrates implement compatible internals.

When two organizations' agentic teams must collaborate on shared work, neither organization's operating model can govern the other's. Each carries its own Governance Layer, its own orchestrator, its own personas. Inter-organization coordination is a Governance-Layer-to-Governance-Layer handshake, not an inter-orchestrator handshake. The Skipjack §7 future-study vector on cross-organizational interoperability¹⁵ applies directly here. A specific implication for the operating model: when a story crosses organizational boundaries, the receiving organization admits it as if it were any other story — the receiving operator authors an HG envelope, the receiving orchestrator decomposes, the receiving AEL executes. The originating organization is a stakeholder, not an authority. The receiving organization's governance flow remains unbroken. The corollary: an organization cannot impose its operating model on an organization it collaborates with. It can specify what evidence it requires from collaborative work (the audit-grade evidence the originating Governance Layer needs to ratify the result) but it cannot specify how the collaborator produces that evidence. This boundary is the same one that

15 Kuiper, "Skipjack Protocol," §7 Vector 5 (Cross-organizational Skipjack interoperability).

applies in human contractor relationships and is preserved here for the same reason: each organization's accountability is its own.

6.1 IMPLICATIONS FOR HARNESS-BASED IMPLEMENTATIONS

An agentic team operating model implementation of this governance doctrine on top of a standard agent harness would require the following from the substrate:

- *Standup-equivalent telemetry surfacing.* The harness must expose session telemetry sufficient for the operator's standup brief: completed work since the last brief, in-progress work with current state, blocked work with the blocking condition, canonical-position changes since the last session, and decisions pending operator adjudication. The governance flow requires this because operator comprehension cannot be re-established between sessions without it; the alternative is that the operator either reconstructs from primary sources at every session (over-constrained mode) or dispatches without comprehension recovery (under-constrained mode). Failure mode if absent: governance authority becomes nominal — the operator holds the role but cannot exercise it on a session-by-session basis.
- *Task admission gating.* The harness must allow the orchestrator to refuse work that lacks a Governance-Layer-approved decomposition envelope, to refuse work targeted at a persona owing SOP acknowledgement, and to refuse work whose receiving agent's context cannot honor the C³ contract. Refusal must be hard: the harness must not provide a path by which a refused task reaches the executing agent through an alternative route. The governance flow requires this because admission discipline is the structural mechanism by which scope boundaries are enforced; without it, the orchestrator's role collapses from arbiter to advisor. Failure mode if absent: tasks reach agents without scope manifests, AE² fails open, and the cycle's outputs cannot be audited back to a Governance-Layer authorization.
- *Multi-context partition isolation.* The harness must support distinct context partitions per cognitive mode — distinct canonical-position sets, distinct SOP sets, distinct memory stores, distinct curation histories. Partition isolation must be substrate-side, not prompt-side: a persona-prompt on a shared underlying context does not satisfy the contract, because the saturation mechanism of §4.1 still applies. Re-unification points are operator-driven (§4.4), and the harness must support the propagation-as-artifact discipline that re-unifies outcomes without merging contexts. The governance flow requires this because context-boundary maintenance is the operational instrument by which cognitive modes preserve discrimination. Failure mode if absent: cognitive-mode contamination, indiscriminable curation history, gradual drift toward single-persona theater.
- *Drift detection telemetry.* The harness must surface measurable signals of cross-context divergence: shared-fact queries returning different answers across personas, SOP version skew across active contexts, canonical-position lineage gaps where a ratification did not propagate, and stale-acknowledgement detection where a persona has not refreshed against an SOP version older than a configurable threshold. The governance flow requires this because drift accumulates silently when undetected; the multi-account split exposes the operating model to drift in exchange for the discrimination benefit, and drift detection is the controlled risk-management instrument that makes the tradeoff defensible. Failure mode if absent: drift accumulates between cycles, reaches shipped output, and is detected only by external consumers rather than by the system itself.

- *SOP propagation hooks.* The harness must allow governed updates to working rules to take effect across all active agent contexts without orphan caches, must record per-persona acknowledgement of new SOP versions, and must integrate the acknowledgement state into the admission-gate decision. SOP changes that propagate eventually (every persona re-reads the SOP at some unspecified later time) are not sufficient; propagation must be gated, with admission refused for personas that owe acknowledgement. The governance flow requires this because SOP-version skew is one of the two silent failure modes of §5.1, and the gate is the only mechanism that makes propagation reliable rather than hopeful. Failure mode if absent: cycle outputs reflect inconsistent rule application across personas, and the orchestrator cannot attest, for any executed task, which SOP version the executing persona had acknowledged at admission.

These are governance contracts that any compliant implementation must honor, regardless of harness substrate.

The contracts are unprecedented to varying degrees in current harness frameworks. Standup-equivalent telemetry can be approximated by composing existing telemetry primitives across major harnesses; no harness ships the standup brief as a built-in artifact. Task admission gating exists in some harnesses (CrewAI's process manager, MetaGPT's¹⁶ SOP-driven orchestration) but is typically capability-based rather than envelope-based; envelope-based admission is novel. Multi-context partition isolation is generally available via separate harness instances but not via substrate-internal partitioning with shared-state guarantees; this is the most substrate-implementation-heavy contract. Drift detection is novel — no standard harness exposes the primitives. SOP propagation can be approximated by orchestrator-side rule injection but is not natively gated.

The implication for substrate selection (Paper 7) is that no current harness satisfies all five contracts out of the box. Compliant implementations will compose harness affordances with operating-model-specific orchestration code. The operating model is teachable; the substrate-side implementation is engineering work.

7. FUTURE STUDY: TEN RESEARCH VECTORS

The operating model defined in this paper is governance doctrine, not a validated implementation. The ten research directions below identify where the doctrine requires empirical grounding, where its claims need formal verification, and where the operative context will likely force extension.

1. Empirical measurement of operator-comprehension half-life across cycle cadences. The standup analogue's frequency is justified by the rate at which operator comprehension decays between sessions. The decay rate is not measured. The research question is specific: given a defined portfolio composition (n personas, m active envelopes), what is the operator's mental-model decay function over time, and how does the function shift as portfolio complexity grows? The answer determines whether session-level standups remain

¹⁶ Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiwu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber, "MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework," in *Proceedings of the Twelfth International Conference on Learning Representations (ICLR 2024, Oral)*, <https://openreview.net/forum?id=VtmBAGCN7o>.

the right cadence as the portfolio scales, or whether more frequent (or less frequent) re-anchoring becomes structurally required. The methodology is operator-survey-plus-comprehension-test; the design challenge is constructing comprehension tests that measure the actual governance-relevant mental model rather than surface recall.

2. Formal specification of admission-gate manifests as machine-verifiable contracts.

The admission-gate manifest specified in §3.3 is described in natural language. A formal specification — type-theoretic, process-calculus, or workflow-specification-language-based — would allow harnesses to verify, at admission time, that a candidate task's manifest satisfies the parent envelope's constraints. The Skipjack §7 vector on formal HITL specification¹⁷ is a closely related research program; admission-gate manifests are a specific instance of the broader specification challenge. The tradeoff is expressiveness vs. tractability: a manifest expressive enough to capture real operator intent may be too rich for tractable verification.

3. Drift-detection algorithm design for cross-context canonical-position lineage tracking. The drift report of §5.2 is described as a query against each persona's canonical-position store. The algorithmic design — how the audit persona efficiently identifies divergences across n personas without saturating its own context, how lineage is tracked across SOP transitions, how reconciliation actions are reified into propagation events — is open. The research question is whether efficient drift detection at scale (dozens of personas, thousands of canonical positions) can be built on existing graph-database or knowledge-graph primitives, or whether new algorithms are required.

4. Operator-comprehension-tooling design for the standup brief and reconciliation surfaces. The standup brief and the drift report are artifacts the operator reads. The cognitive ergonomics of reading them — what visual structure best supports rapid comprehension, what level of summarization optimizes the trade-off between information density and reading time, what affordances support efficient adjudication of pending decisions — is a tool design problem. The Skipjack §7 vector on Governance-Layer tooling¹⁸ applies directly. The research question is what software interfaces support effective operator decision-making at the load levels the operating model produces in practice.

5. SOP propagation protocol design for multi-substrate operating environments. When personas span multiple harness substrates, SOP propagation must traverse substrate boundaries. The propagation protocol — how SOP versions are serialized for cross-substrate transmission, how acknowledgements are recorded in a substrate-neutral format, how propagation latency is bounded — is open. Distributed authorization protocols and inter-substrate trust frameworks from the security literature provide starting points. The substrate-research dimension (Paper 7) intersects this work.

6. Persona specialization vs. generalization tradeoffs. Adding a persona reduces context contamination but adds re-unification overhead. The tradeoff is observable in the working portfolio at five personas; whether it remains favorable at ten or twenty is unknown. The research question maps: when does a marginal persona reduce drift more than it adds coordination cost, and how does the tradeoff shift as the portfolio's cognitive-mode coverage saturates? The answer informs how organizations structure their operating models as they grow.

¹⁷ Kuiper, "Skipjack Protocol," §7 Vector 1 (Formal specification of HITL control point topologies).

¹⁸ Kuiper, "Skipjack Protocol," §7 Vector 6 (Governance Layer tooling design).

7. Multi-organization operating-model interoperability. Two organizations running independent operating models that must collaborate face the same authority-boundary challenge as two Skipjack-governed systems collaborating across organizational boundaries. The operating-model-specific question is: what artifacts (envelopes, manifests, evidence) must travel cross-organization, in what format, with what audit obligations on each party? Inter-organizational governance trust frameworks and audit-trail interchange standards from the regulated-industries literature are the most relevant prior art.

8. Cross-cycle memory governance. Personas accumulate memory across cycles — the curation histories grow, the canonical-position sets thicken, the SOP acknowledgement logs lengthen. The governance question is what subset of accumulated memory must persist into future cycles versus what should be summarized, archived, or forgotten. The Skipjack §7 vector on memory governance audit tooling¹⁹ is directly applicable; the operating-model addition is that memory governance must be persona-specific (each persona's memory is governed independently) rather than portfolio-wide.

9. Operator-load measurement. How many cognitive modes can a single operator govern before comprehension fragments? The answer determines the practical scaling ceiling of the operating model in single-operator portfolios. The research is empirical: instrument operator sessions across portfolios of varying complexity, measure comprehension fidelity by structured tests, identify the bound at which fidelity falls below operationally tractable thresholds. The bound is likely a function of portfolio complexity (n personas \times m active envelopes \times cycle cadence) rather than a fixed persona count, but the function's shape is not known.

10. Operating-model evolution and backward compatibility. The operating model is not static. As agentic capability evolves, as new harness primitives become available, as the empirical evidence on what works accumulates, the model will need to update. The challenge is updating without breaking active sprints: an operating-model change mid-cycle disrupts envelopes already authored, manifests already attached, evidence already produced under the older specification. The research question is how operating-model versioning works — whether updates apply at cycle boundaries only, whether parallel old/new model operation is supported during transitions, whether SOP propagation discipline (§5.3) extends to operating-model versions.

8. CONCLUSION

The operating model described in this paper is not a methodology invented from scratch. It is the daily form of the Skipjack Protocol applied to a team-sized unit of work. The role taxonomy (Governance Layer / Orchestrator / Agentic Execution Layer), the manifest discipline (context integrity manifests on every task), the HITL control points (standup brief at session boot, completion review at cycle end), and the feedback loops (retrospective + SOP propagation) are all Skipjack mechanisms. The paper's contribution is to specify how those mechanisms manifest as ceremonies, artifacts, and gate criteria in practice.

The four claims of the operating model have been defended through the paper. *Ceremony cadence is structural, not optional*: the standup brief is the load-bearing artifact by which operator comprehension is re-established, and skipping it collapses the model toward the under-constrained failure mode of §1.2. *The orchestrator is the admission-gate arbiter*: no task enters the Agentic Execution Layer without verification that it carries a Governance-Layer-approved decomposition envelope, and the orchestrator's authority to refuse is the structural

19 Kuiper, "Skipjack Protocol," §7 Vector 8 (Memory governance audit tooling).

mechanism by which scope boundaries are enforced. *Stories and tasks layer over HG+C³ and AE² as first-class governance artifacts*: a story is a curation envelope plus a context contract, a task is an admission-gated, evidence-producing decomposition, and the separation prevents the operator from skipping decomposition or the agents from inventing scope. *Context-boundary maintenance is implemented as a multi-account split*: cognitive modes are isolated into distinct agent contexts, with re-unification at named decision points and drift detection as a continuous routine.

The portfolio under which this paper was written is itself an existence proof. The decalogy progressed through Sprint 7 with five active personas — an orchestrator, a narrative-governance persona, a content-pipeline persona, a substrate-research persona, and a revenue persona. The model held: the papers shipped at v1.0-preprint with audit-grade evidence chains, the cycle's decisions were ratified through completion reviews, the SOP propagation log gated tasks where acknowledgements were owed, and the drift reports surfaced (and resolved) the small divergences each cycle inevitably produced. This paper specifies the model that the paper's own production demonstrates is tractable. The proof is in the working portfolio, not in the prose.

The operating model is not finished work. The ten research vectors of §7 identify where empirical grounding is needed, where formal specification would harden the doctrine, and where the operating context will likely force extension. The capstone implementation paper (Paper 9) and the capstone results paper (Paper 10) will produce the empirical evidence. This paper's job is to specify the doctrine in a form precise enough that the implementation can be built against it.

The operating model for agentic teams is a governed cadence of ceremonies that maintain operator comprehension, orchestrator admission discipline, and inter-context boundaries — within which non-human agents execute against a substrate-independent governance contract.
