DENDRO LOGIC
INTELLIGENT DATA

Beyond Sprints

A Workflow Playbook for Agentic Teams

Redesigning delivery for teams where agents write the code

By Mike McGreal

Dendro Logic Ltd
June 2026
Internal Engineering Playbook

https://dendro-logic.com

Contents

Part 1: Why Sprints Break in the Agentic Era

The Bottleneck Has Moved

Agile and Scrum were designed for a world in which writing code was the scarce, slow step. The two-week sprint, the story-point estimate, the backlog, the daily stand-up - almost every load-bearing Agile ritual exists to meter, prioritise and coordinate a fixed amount of human coding capacity. When that capacity is no longer the constraint, the rituals built around it start to manage the wrong thing.

AI coding agents remove the scarcity. An agent can generate, test and validate an implementation on demand, which means the binding constraint in software delivery has moved. It now sits upstream, in specification and intent, and downstream, in review, verification and integration. As Martin Fowler and his ThoughtWorks colleagues set out in "Exploring Generative AI", classic artefacts such as the Product Backlog and the Definition of Done lose much of their original purpose, because the agent already satisfies many of the "done" criteria as it works. The limiting factor becomes how fast a human can specify the work clearly and check it thoroughly.

This is the structural fact the rest of this playbook follows from. If coding is no longer the bottleneck, then the ceremonies and metrics that exist to manage coding capacity are optimising the wrong constraint, and the team's flow will not improve no matter how diligently those ceremonies are run.

The Reality Check: It Feels Faster, the Data Says Otherwise

Before redesigning anything, a senior team has to separate marketing from measurement, because the headline claim - that agents make delivery dramatically faster - is not supported by the best evidence available.

The most rigorous independent study to date is the METR randomised controlled trial (2025). Sixteen experienced open-source developers worked on 246 real issues drawn from their own large, mature repositories. Each issue was randomly assigned to an "AI-allowed" or "AI-disallowed" condition. With AI permitted, developers took roughly **19% longer** to complete their work. The most important finding was behavioural: the developers genuinely believed the AI had made them faster. Perceived speed and actual end-to-end throughput diverged sharply.

Google's DORA research points the same way. Across its 2024 and 2025 reports, DORA finds no measurable change to the four core delivery metrics attributable to AI tools on their own, and frames AI only as an "amplifier" of the surrounding sociotechnical system. An agent dropped into a team with weak testing and slow review will simply amplify the instability.

The honest position is this. The workflow shift is real and worth designing for as a fundamental change in how teams work. The productivity claim is unproven, and in the best controlled study so far it is negative. The value of agents is in leverage, scope and capability, embedded in a mature delivery system - not in raw speed. Design for that, measure it honestly, and do not promise speed you cannot evidence.

Why This Playbook Exists

The companion playbook in this series, *AI Agents in Development Teams*, covers adoption, guardrails and review - how to run agents safely. This playbook covers the layer above that: the operating model. It answers the question a team asks once the tools are in place and the guardrails are set: if sprints, story points and stand-ups were built for a constraint that no longer binds, what does the team run instead?

UK Context
The Stack Overflow 2025 Developer Survey found that 84% of developers are now using or planning to use AI tools, and the JetBrains State of Developer Ecosystem 2025 reported roughly 85% regular usage. Adoption is effectively settled. The open question for UK engineering leaders is no longer whether to use agents, but whether the delivery process around them is still fit for purpose.

Part 2: What Replaces the Sprint

Continuous Flow or Longer Fixed Cycles

When implementation time collapses, the fixed two-week timebox stops doing useful work. Two credible replacements have emerged, and a team can pick either, or combine them.
The first is **continuous flow**: work is pulled as capacity frees, prioritised continuously rather than batched into sprints, and measured by how quickly an item moves from idea to production. The second is a **longer fixed-time, variable-scope cycle**, of which Basecamp's Shape Up is the best documented example. Both abandon the sprint. Both plan by intent and appetite rather than by task breakdown.

Shape Up: The Proven Sprint-Free Model

Shape Up, created by Ryan Singer at Basecamp, is the one fully documented, battle-tested sprint-free model, and it maps cleanly onto agentic work. It runs on a six-week, fixed-time, variable-scope cycle followed by a short cool-down, with three phases:

- **Shaping.** A small team writes a "pitch" that defines the problem, the appetite (the time budget), a rough solution, the rabbit holes to avoid and the no-gos. The pitch is rough but solved.
- **Betting.** A betting table chooses which shaped pitches to fund for the next cycle. This replaces the continuously groomed backlog.
- **Building.** A small cross-functional team owns the work end to end and ships a usable slice. Done means deployed.

It deliberately omits the Product Backlog, sprint planning, daily stand-ups, velocity tracking and retrospectives. For an agentic team, the shaping pitch doubles as the high-level specification that drives the agents, and the six-week container comfortably absorbs the iterative evaluation loops that agent-authored code requires.

Appetite Over Estimation

The core idea worth taking from Shape Up, whichever cadence you adopt, is **appetite over estimation**. Instead of asking "how long will this take?" - a question traditional estimation answers poorly and which agents make harder still, because their output is

non-deterministic - the team asks "how much time and resource is this problem worth?" The appetite is fixed and the scope is hammered to fit it. That single inversion removes the dependency on story-point sizing that the rest of this playbook argues is now obsolete.

Part 3: Redesigning the Ceremonies

The Death of Story-Point Estimation

Story-point estimation assumes that the effort to implement a unit of work is the thing worth predicting. When an agent can produce a working implementation in minutes, that assumption fails. The ThoughtWorks Technology Radar and Fowler both reach the same conclusion: velocity-based estimation becomes meaningless, and teams should track flow metrics instead.

The table below summarises the ceremony-by-ceremony direction of travel. None of these is a wholesale deletion of coordination; each is a replacement of a coding-capacity ritual with an intent-and-flow equivalent.

Classic ceremony	Why it breaks	What replaces it
Sprint planning	No need to break work into estimable stories when the agent plans the implementation	Lightweight intent specifications or shaping pitches that record the desired outcome
Story-point estimation / velocity	Implementation time collapses; sizing predicts the wrong thing	DORA-style flow metrics (lead time, deployment frequency) and WIP limits
Daily stand-up	Synchronous status is redundant when agents update state in real time	Asynchronous telemetry and live dashboards as work is opened, compiled and merged
Backlog refinement	Agents can generate, group and summarise backlog items continuously	Continuous, AI-assisted backlog synthesis
Sprint review	No fixed demo cadence when working software lands continuously	Live stakeholder dashboards surfacing artefacts, tests and deployments
Retrospective	Improvement should be continuous, not periodic	A continuous-improvement loop driven by cycle-time and flow analytics
Fixed sprint	The timebox manages a constraint that no longer binds	Continuous flow, or a Shape-Up-style longer cycle

The practical message for a team lead: do not defend a ceremony because it is familiar. Defend it only if it still manages a constraint that genuinely binds. Most do not.

Part 4: Specification-Driven Development

Specs as the Durable Artefact

If unstructured "vibe coding" - prompting fast and accepting output without thorough review - is one pole, the disciplined opposite is **specification-driven development**. The principle is economic: when an agent can regenerate code cheaply, the code becomes a disposable commodity and the specification becomes the high-value, reusable asset. A good spec captures the stable business intent, the architectural constraints and the compliance rules that survive across regenerations. It is version-controlled, reviewable and durable in a way that generated code is not.
This is the new planning layer. The work of the team shifts from writing the implementation to writing the precise, machine-actionable specification that the implementation is generated from, and from accepting code to verifying it against that spec.

The Tooling Landscape

A set of specification-first tools is maturing quickly. Treat the specific feature sets below as indicative of direction rather than settled standards - this category is young and moving fast - but the shape is clear and consistent.

- **GitHub spec-kit** implements an open, version-controlled loop - specify, then plan, then tasks - where the specification, the generated plan and the generated tests are all committed alongside the code.
- **AWS Kiro** is an agentic IDE that unpacks a natural-language prompt into structured requirements (using EARS-style acceptance criteria), generates a technical design document, and sequences linked implementation tasks with their own tests. It uses persistent steering files for conventions and agent hooks for event-driven actions.
- **Tessl** (from Snyk founder Guy Podjarny) centres development on "skills as code" - the formal, version-controlled definition of what the software should do - on the premise that code is being commoditised and the durable craft is the intent.

The common thread across all three, and across Fowler's "Understanding Spec-Driven Development" framing, is the same: the specification, not the code, becomes the source of truth.

Part 5: Roles and Team Structure

The Engineer as Orchestrator

The developer's role moves from writing syntax to orchestrating, guiding and validating machine execution. The day-to-day becomes authoring specifications, steering agents, and reviewing output rather than typing implementations. This is not a junior task; it requires more systems-level judgement, not less.

Team Topologies as the Infrastructure for Agency

Team Topologies has moved explicitly into this space with its "Agentic AI" assessment, reframing its four canonical team types as the "infrastructure for agency" - the same patterns that make human teams effective applied as the guardrails for AI agents:

- **Stream-aligned teams** consume agent capabilities and own the business intent, prioritisation and prompt steering.
- **Platform teams** supply the tooling, security policy and observability that let agents run reliably.
- **Enabling teams** coach prompt engineering, agent-design patterns and cognitive-load limits.
- **Complicated-subsystem teams** own the specialised models, data pipelines and domain logic that act as guardrails.

Emergent Hybrid Roles

In practice, several organisations report the traditional multi-person squad compressing into a smaller cell of systems-level experts working across layers. The roles below are an emerging pattern rather than a settled standard, but they are a useful map for workforce planning.

Emerging role	Primary focus
System architect / agent designer	System architecture, task-decomposition paths, integration boundaries, escalation logic when agents fail
Spec engineer	Translating business intent into precise, machine-actionable specifications
Eval / quality thinker	Test-evaluation harnesses, regression suites, adversarial red-teaming, model-drift monitoring
Domain expert	Strategic domain context and verification that agent output is correct in context
AI engineering orchestrator	Hybrid workstreams, agent compute and token cost, coordination blocks

A word of caution: the common claim that agentic teams become "smaller and flatter" is plausible and reported anecdotally, but it is not yet supported by published head-count or span-of-control data. Plan for role change; measure team-size change before assuming it.

Govern Autonomy in Tiers

Not every task should be delegated to the same depth. A practical pattern, drawn from frameworks such as KPMG's TACO classification and Marc Bara's Human-AI Integration Framework, is to log every task under an explicit, reversible autonomy tier:

- **Tier 0 - Pure augmentation.** IDE autocomplete, administrative cleanups.
- **Tier 1 - Bounded execution.** The agent implements a single function under direct prompting and manual review.
- **Tier 2 - Collaborative autonomy.** The agent works asynchronously on a branch to implement a scoped story, verified by automated integration tests.
- **Tier 3 - Full orchestration.** Multi-agent networks deliver end-to-end features through deterministic orchestration gates.

Tiers should be demotable: if an agent's reliability on a class of work drops, move it down a tier. The point is governed, reversible delegation with a named human owner accountable for every output.

Part 6: Verification Becomes the Constraint

Apply the Theory of Constraints

Once code generation is cheap and abundant, the next constraint surfaces, and it is review, verification and integration. This is the single most important operational shift in the agentic era: the throughput of your delivery system is now set by how fast you can verify, not how fast you can write.
That creates a real risk that review becomes the new red tape. The job of the operating model is to design the verification throttle so it protects quality without re-creating the queue that sprints were invented to manage. The levers are:

- **Spec-as-test.** The specification serves as the executable acceptance suite the agent must satisfy, so verification is built into generation rather than bolted on after.
- **Automated quality gates.** CI/CD gates carry the routine load, reserving human review for genuine judgement.
- **Risk-tiered, sampled review.** Not every agent change needs the same scrutiny; route review effort by risk.

The New Failure Modes

Unconstrained generation introduces failure modes that traditional process did not have to manage. These figures come from third-party analyses (notably GitClear and ThoughtWorks) rather than from controlled trials, so treat them as directional warnings rather than precise constants - but the direction is well-attested:

- **Code churn and duplication.** GitClear's analysis reports a sharp rise in short-term code churn and in copy-pasted code, alongside a fall in refactoring, among heavy AI users. The risk is a codebase that grows fast and rots faster.
- **Security by default-of-least-resistance.** Agents tend to take the path that makes code compile, which is not the path that makes it secure. Industry and OWASP-aligned analyses warn that a meaningful share of AI-generated snippets carry a security weakness. Prompts alone do not fix this; secure-by-default templates and non-negotiable security gates do.
- **The testing-infrastructure mismatch.** CI pipelines tuned for human-speed commits stall when parallel agents push code and tests at machine speed. Plan for parallel, ephemeral test environments rather than a single shared queue.

Part 7: Measuring Flow, Not Velocity

DORA and SPACE, Not Story Points

Stop measuring raw activity - lines written, commits, points completed. In an agentic system those numbers are trivially inflated and tell you nothing about delivered value. Measure flow and outcomes with a balanced dashboard that combines DORA delivery metrics with SPACE developer-experience dimensions.

Category	Indicator	Why it matters in an agentic SDLC

Flow (DORA)	Deployment frequency, change lead time	End-to-end flow efficiency; most lead time is waiting, not building
Quality (DORA)	Change-failure rate, time to restore	System stability and reliability of rollbacks
Rework (SPACE)	Code reverted, refactored or rewritten within 30 days	High rework signals rubber-stamped, low-comprehension output
Balance (SPACE)	New-feature effort vs maintenance / debt	Whether agent leverage is funding progress or just more rework
Wellbeing (SPACE)	Tool friction, review fatigue, role clarity	The cognitive load of reviewing high volumes of agent output

The Perceived-Versus-Actual Trap

The METR result has one direct operational consequence: do not trust self-reported speed. Track the gap between perceived throughput and measured lead time explicitly. If the team feels faster while lead time is flat or rising, that is the signal to inspect where the work is actually waiting - almost always in review and integration, the new constraint.

Part 8: The Operating Model and Rollout

A Concrete Operating Model

No single published manifesto yet bundles all of this into one proven framework; what follows is a considered synthesis, not an evidenced standard. Adopt it as a starting design and adapt it to your context.

- **Cadence.** Replace fixed two-week sprints with continuous flow, or with Shape-Up six-week cycles plus a cool-down used deliberately for refactoring, debt and test-harness hardening.
- **Planning.** Replace sprint planning and backlog grooming with shaping pitches and intent specifications. Make the specification the durable, version-controlled artefact.
- **Ceremonies.** Drop story points, velocity and the daily stand-up; replace with telemetry and live dashboards. Keep a periodic prioritisation or betting decision. Turn the retrospective into a continuous-improvement loop.
- **Roles.** The engineer is an orchestrator and reviewer; product owners own intent and acceptance; QA owns human-in-the-loop verification. Govern delegation in reversible autonomy tiers with a named owner per output.
- **Metrics.** Flow and outcomes, not velocity. Watch the perceived-versus-actual gap.
- **Quality.** Treat verification as the binding constraint: spec-as-test, automated gates, risk-tiered review, secure-by-default templates.
- **Stance on speed.** Adopt for leverage and capability inside a mature system. Measure honestly; do not assume speed.

Rolling It Out

Process change fails when it is imposed wholesale. Treat the transition as a delivery project in its own right.

Phase	Focus
Pilot	One team, one product area. Run the new cadence and metrics alongside the old for one cycle to build a baseline.
Prove	Publish the flow metrics. Let lead-time and rework data, not advocacy, make the case.
Expand	Roll out cycle by cycle. Build internal champions who have run it, not evangelists who have read about it.
Embed	Retire the old ceremonies formally once their replacements are trusted. Leaving both in place is the most common failure.

Addressing the common objection
"Without estimates and sprints, how do we give the business predictability?" Predictability comes from a fixed appetite and a measured lead-time distribution, not from story-point forecasts that were never accurate. A team that ships continuously and publishes its lead-time percentiles gives stakeholders a more honest forecast than a velocity chart ever did.

Part 9: Quick-Reference Checklists

Transition Checklist

- [ ] Baseline current lead time, deployment frequency, change-failure rate and rework before changing anything.
- [ ] Pick a cadence: continuous flow or Shape-Up cycles. Stop running fixed sprints.
- [ ] Replace sprint planning with shaping pitches / intent specifications.
- [ ] Retire story-point estimation; plan by appetite.
- [ ] Stand up a flow + SPACE dashboard; remove activity-based metrics.
- [ ] Define autonomy tiers and a named-owner rule for agent output.
- [ ] Move verification to the centre: spec-as-test, automated gates, risk-tiered review.
- [ ] Run one pilot cycle in parallel with the old process before committing.

Ceremony Redesign Checklist

- [ ] Sprint planning to intent specification.
- [ ] Estimation to appetite and flow metrics.
- [ ] Daily stand-up to async telemetry and dashboards.
- [ ] Backlog grooming to continuous AI-assisted synthesis.
- [ ] Sprint review to live stakeholder dashboards.
- [ ] Retrospective to a continuous-improvement loop.

Verification Readiness Checklist

- [ ] Specifications double as executable acceptance tests.
- [ ] CI/CD quality gates carry routine review load.
- [ ] Secure-by-default templates and a non-negotiable security gate are in place.
- [ ] Test infrastructure supports parallel, machine-speed generation.
- [ ] The perceived-versus-actual throughput gap is tracked, not assumed.

Part 10: References and Further Reading

Controlled Evidence

- METR, "Measuring the impact of early-2025 AI on experienced open-source developer productivity" (2025) - the randomised controlled trial reporting a 19% slowdown.
- Google DORA, State of DevOps research 2024 and 2025 - AI as an "amplifier"; no measured change to the four core metrics.

Methodology and Frameworks

- Martin Fowler and colleagues, "Exploring Generative AI" and the linked memos including "Understanding Spec-Driven Development: Kiro, spec-kit, and Tessl" (ThoughtWorks).
- Basecamp, *Shape Up* (Ryan Singer) - the six-week, fixed-time, variable-scope model.
- Team Topologies - the four team types and the "Agentic AI" assessment ("infrastructure for agency").
- ThoughtWorks Technology Radar - flow metrics over velocity; reimagined ceremonies.

Tooling and Standards

- GitHub spec-kit; AWS Kiro; Tessl - specification-driven development tooling.
- The DORA and SPACE measurement frameworks for delivery and developer experience.

Third-Party Analyses (Directional, Not Controlled)

- GitClear code-quality research on AI-era churn, duplication and refactoring trends.
- KPMG TACO framework and Marc Bara's Human-AI Integration Framework for autonomy-tier governance.
- Industry and OWASP-aligned analyses of security weaknesses in AI-generated code.

Figures attributed to third-party analyses above are reported as published by those sources and have not been independently re-verified; treat them as directional. Controlled-evidence claims (METR, DORA) and primary-source methodology (Fowler, Shape Up, Team Topologies) are the load-bearing references.