段取り

# DANDORI

## A Team Coordination Methodology for AI-Augmented Engineering

Version 1.0 | March 2026

# 段取り

## *dandori*

*Japanese noun. Literally: "preparation" or "setup."*

In Japanese manufacturing and the Toyota Production System, dandori refers to the meticulous preparation work done before production begins. It encompasses everything from arranging tools and materials to verifying that the setup is correct before the first piece is produced. The principle is simple: the quality of the output is determined by the quality of the preparation, not by what happens on the production line.

In the context of this methodology, Dandori carries the same meaning into software development. When AI handles a growing share of code production, the human contribution that determines quality is no longer the act of writing code. It is the act of preparation: specifying precisely what needs to be built, deciding what matters and what does not, and validating that the output matches intent.

Dandori is the recognition that in AI-augmented software development, preparation is the work.

> *In manufacturing, they discovered that 80% of quality problems originate in preparation, not production. In software, we are learning the same lesson.*

# Table of Contents

# Executive Summary

Scrum, as practiced in most organizations, is broken. The word has lost its meaning. Every company and every team interprets it differently, and the gap between what the Scrum Guide describes and what teams actually do has become so wide that saying "we do Scrum" communicates almost nothing about how a team works.

At the same time, AI is joining software teams as a participant, not just a tool. Whether through coding assistants, autonomous agents, or embedded AI features, a growing share of implementation work is now shared between humans and AI systems. The same team can now deliver dramatically more, and that amplified capacity changes what coordination looks like. Scrum, designed to coordinate human-only teams producing at human-only speed, has no model for this new reality.

Dandori is a development framework built for this convergence. Named after the Japanese manufacturing concept meaning "preparation" or "setup," Dandori reflects the core insight of the Toyota Production System: the quality of output is determined by the quality of preparation, not by what happens on the production line.

In Dandori, the specification is the unit of work. Human judgment is the scarce resource. AI is the execution engine, though humans remain in the loop across the full spectrum from AI-assisted to fully human-implemented work. The framework provides a complete operating model: a five-stage specification lifecycle, three defined roles, a product alignment model, a redefined engineering management function, and a set of ceremonies that transform rather than eliminate Scrum's core practices. Each element exists to optimize the flow of human thinking through a system where machines handle an increasing share of production.

> *Scrum optimized for production. Kanban optimized for flow. Dandori optimizes for preparation, because when AI handles production and flow is continuous, the quality of what you prepare determines everything.*

# What Dandori Is and Is Not

## Dandori Is Not a Spec-Driven Development Tool

The emergence of AI in software development has produced a wave of spec-driven development (SDD) tools and frameworks: GitHub Spec Kit, AWS Kiro, the BMAD Method, OpenSpec, Tessl, and others. These tools focus on the developer workflow. They answer the question "how does a developer go from a specification to working code with AI assistance?" They provide CLIs, agent workflows, specification templates, and implementation pipelines.

Dandori is none of these things.

Dandori does not generate specifications. It does not execute code. It does not provide a CLI. It does not define a spec format or a prompt template. It does not depend on any specific AI tool, model, or vendor.

Dandori answers a different question entirely: **how does a team coordinate when the way software gets built is changing?**

Spec Kit tells a developer how to structure a specification for GitHub Copilot. Dandori tells a team how to decide what gets specified, who writes the spec, who reviews the output, how priorities flow, when to meet, and how to know if the system is healthy. These are complementary, not competitive. A team could use Spec Kit or Kiro or BMAD for their developer workflow and use Dandori for their team coordination. Or they could use no SDD tool at all and still use Dandori.

The distinction matters because confusing the two leads to wrong expectations. Someone expecting a developer tool will be disappointed. Someone looking for how to organize their team in a post-Scrum world will find exactly what they need.

## Dandori Is Tool-Agnostic

Dandori does not prescribe which AI tools a team should use. It does not assume GitHub Copilot, or Claude Code, or Cursor, or any specific model or platform. It does not even assume AI tools that follow the spec-driven development pattern.

A team using AI through embedded features in their CI/CD pipeline can use Dandori. A team using autonomous coding agents can use Dandori. A team using AI only for code review and test generation can use Dandori. A team using AI in ways that have not been invented yet can use Dandori.

This is deliberate. AI tools change every quarter. Any framework that ties itself to a specific tool or workflow pattern will be obsolete within a year. Dandori ties itself to the coordination problem, which is durable, not to the tools, which are not. The five-stage lifecycle (Intent, Specification, Execution, Validation, Integration) describes how work flows through a team regardless of whether the Execution stage is handled by an AI agent, a human developer with AI assistance, or a human developer with no AI at all.

## Dandori Works Without AI

This is the claim that most clearly separates Dandori from the SDD ecosystem: a team that uses no AI whatsoever can still benefit from Dandori.

The framework was born from two converging observations. The first is that AI is joining software teams. The second, and arguably the more fundamental one, is that Scrum is broken. These are independent problems. AI makes Scrum's dysfunction more visible and more costly, but the dysfunction existed before AI arrived.

Consider a team that writes no AI-generated code but suffers from the classic Scrum problems: ceremonies that have lost their meaning, estimation rituals that produce no useful signal, sprint boundaries that create artificial urgency or artificial slack, and a coordination model that optimizes for human labor allocation when the real bottleneck is decision quality and specification precision.

That team can adopt Dandori's principles (the specification as the unit of work, human judgment as the scarce resource, preparation determines quality), its roles (Spec Owner, Reviewer, Prioritizer), its ceremonies (Pipeline Sync, Integration Review, Spec Retrospective), and its metrics (cycle time, first-pass success rate, decision latency) without touching a single AI tool. The methodology will make them more effective because it coordinates around the activities that actually determine software quality: thinking clearly about what to build, specifying it precisely, and validating that the output matches intent.

When that team later adopts AI tools, Dandori is already in place. The coordination model does not need to change. The team simply discovers that the Execution stage gets faster, which means the surrounding stages (Specification, Validation) become proportionally more important, which is exactly what the methodology was designed to optimize.

This is what makes Dandori a team coordination methodology rather than an AI development tool. It is a response to the modern reality of software delivery, of which AI is one part but not the only part. It works for teams with AI. It works for teams without AI. It works for teams that are somewhere in between. The coordination problem it solves, how to organize humans to produce high-quality software through precise specification, fast decisions, and rigorous validation, is universal.

PART I

# WHY DANDORI

*The case for a new framework*

# 1. Scrum Is Already Broken

Most engineering leaders already know this, even if they do not say it out loud. Scrum, as practiced in the vast majority of organizations, is broken. Not because the Scrum Guide is flawed, but because the word itself has lost its meaning.

Every company has its own interpretation of Scrum. Every team within the same company has different agreements about what Scrum means. Ceremonies that share the same name look nothing alike from one team to the next. Role boundaries are routinely blurred. Sprint lengths are debated endlessly. The gap between Scrum as described in the official guide and Scrum as practiced on the ground has grown so wide that saying "we do Scrum" communicates almost nothing about how a team actually works, how decisions get made, or how quality is maintained.

When a framework becomes infinitely malleable, it stops being a framework. It becomes a vocabulary for describing the status quo, not a system for improving it. The practices have drifted so far from their original intent that the word "Scrum" now serves as organizational shorthand for "we have sprints and standups" rather than a coherent methodology with predictable outcomes.

Dandori does not claim to be immune to this problem. But it starts from a clear-eyed acknowledgment that the problem exists, and it defines its practices precisely enough that adoption means something specific rather than something everyone can reinterpret to mean whatever they are already doing.

# 2. AI Is Joining the Team

Simultaneously, and independently of Scrum's dysfunction, a fundamental change is happening in how software teams work. AI is becoming a team member, not just a tool. This is not a prediction about the future. It is a description of the present.

The forms this takes vary widely. Some teams use AI coding assistants like GitHub Copilot, Cursor, or Windsurf as intelligent autocomplete that accelerates individual developers. Some use agentic tools like Claude Code, Kiro, or autonomous coding agents that can take a task description and produce working code with minimal human guidance. Some use AI through embedded features in their existing tools: CI/CD systems that auto-generate tests, code review tools that flag issues, documentation generators, and monitoring systems that suggest fixes. Some are building custom AI agents that participate in their specific workflows.

The specific tool matters less than the pattern. In all of these cases, a significant portion of what used to be human-exclusive work, writing code, writing tests, writing documentation, debugging, and even some design decisions, is now shared between humans and AI systems. The AI is not replacing the humans. It is joining them. And that changes the coordination model.

Scrum was designed to coordinate teams of humans. Its ceremonies, roles, and artifacts all assume that the people in the room are the ones doing the work, and that the primary challenge is aligning their efforts, managing their capacity, and creating space for them to do their best work. When AI handles a growing share of implementation, those assumptions weaken. The primary challenge shifts from coordinating human labor to coordinating human judgment: what to build, how to specify it precisely enough for AI to execute well, and how to validate that the output is correct and safe.

The result is that the same team delivers dramatically more. A team of five engineers that previously shipped X now ships X multiplied by an AI-amplification factor. This is not about replacing people. It is about each person becoming capable of more, with AI handling the mechanical work while humans focus on the judgment work. This amplified capacity needs a coordination model that matches: one that can handle higher throughput of decisions, specifications, and reviews without drowning in ceremony designed for a slower pace.

Dandori is designed for this convergence. It does not assume that all work is AI-executed. It does not assume any specific AI tool or capability level. It assumes that software teams will increasingly include AI as a participant in one form or another, whether as an independent agent, an embedded assistant, a custom automation, or any tool that leverages AI capabilities. The same team that delivered X before AI can now deliver X multiplied, and the coordination model needs to account for this amplified capacity.

# 3. The Economic Argument

The math is simple and hard to argue with. Scrum was designed when implementation was the dominant cost in software development. A team of seven engineers at an average fully-loaded cost of $150,000 represents roughly one million dollars annually. Sprint ceremonies, estimation, and coordination overhead consumed 15-20% of that capacity, but the remaining 80% was spent on the expensive thing: humans writing code. The ceremony cost was justified because it improved the utilization and alignment of that expensive resource.

When AI handles a growing share of implementation, the cost structure shifts. The expensive resource is increasingly the thinking that precedes and follows code production: specification, decision-making, review, and architectural judgment. If a team spends a significant portion of its time in ceremonies designed to optimize code production, and code production is becoming less of a bottleneck, that represents a meaningful investment in coordination infrastructure that optimizes for the wrong constraint.

Dandori redirects that investment. Instead of ceremonies that ask "are our developers productive?" it runs ceremonies that ask "are our specifications precise, our decisions fast, and our reviews thorough?" That is optimizing for the actual bottleneck.

# 4. The Failure Mode Argument

This is the most persuasive argument for skeptics, because it does not ask anyone to believe the new methodology is better. It asks them to see that the current system is actively creating problems.

Scrum's ceremonies were designed to coordinate human implementation work. When AI joins the team and the team's output capacity multiplies, those ceremonies do not just become less useful. They actively produce dysfunction.

Sprint planning becomes misaligned when the team operates at two speeds: AI-executed work that completes in hours and human-implemented work that takes days. The timebox, designed to contain human implementation uncertainty, becomes an artificial constraint that neither matches the speed of AI execution nor provides meaningful protection for human work.

Standups lose their purpose when the work to coordinate is increasingly about specification quality and decision pipeline health rather than individual implementation progress. The format

was designed for tracking who is doing what. In a spec-driven world, the question is not who is doing what but whether the system is flowing.

Sprint reviews create false cadence when features can ship daily but are held for a biweekly demo. Retrospectives miss the new failure modes when they focus on team dynamics rather than the human-AI collaboration patterns that now dominate delivery outcomes.

Dandori is not proposing change for novelty. It is proposing change because the current coordination model was built for a different team composition and a different production model. When the team changes and the production model changes, the coordination model must change too.

# 5. The Empirical Argument

Honesty is required about what can and cannot be proven. No one has longitudinal data on spec-driven frameworks because they are too new. But the empirical case can be built from adjacent evidence.

First, the data on Scrum's actual effectiveness is weaker than most people realize. Industry surveys consistently show that organizations struggle with Scrum adoption, that the most common "agile" implementation is really "Scrum ceremonies bolted onto waterfall thinking," and that the correlation between Scrum adoption and delivery outcomes is modest at best. Dandori does not need to prove it is better than ideal Scrum. It needs to prove it is better than Scrum as actually practiced, which is a much lower bar.

Second, the Kanban and flow-based evidence supports continuous flow over batched delivery for teams with high variability in task duration. That describes both AI-augmented work and traditional development: some tasks take hours, some take weeks, and forcing them into uniform sprint containers creates waste. The queueing theory behind this is well-established in operations research, even if it is underappreciated in software methodology discussions.

Third, organizations that track their own AI-assisted delivery metrics can generate internal evidence. When feature delivery time compresses from weeks to days for AI-augmented work, while other features still require traditional human implementation timelines, the question becomes not "should we change?" but "how do we coordinate a team that operates at two different speeds?" Scrum has no answer for that. Dandori does.

# 6. The Talent Argument

This argument matters for leadership buy-in because it touches retention and competitiveness.

Strong senior engineers increasingly find Scrum ceremonies patronizing and wasteful. They do not need a standup to stay aligned. They do not need planning poker to understand what they can deliver. They do not need a Scrum Master to remove blockers they can resolve themselves. The overhead was always most burdensome on the highest-performing team members, and AI amplifies this disparity because senior engineers benefit most from AI-augmented workflows.

Dandori gives senior engineers what they actually want: clear priorities, autonomy in how they approach specifications, fast feedback on their output, and minimal process overhead. It treats them as the high-judgment professionals they are rather than as interchangeable sprint resources.

At the same time, the framework provides more structure for junior and mid-level engineers who are still developing their specification and review skills. The Spec Handoff ceremony, the Spec Retrospective, and the growing library of specification patterns create a learning environment that is more concrete and actionable than Scrum's generic "inspect and adapt."

The talent argument to leadership is direct: if your best engineers are frustrated by process overhead that does not match how they actually work, and your competitors are offering lighter, faster, more AI-native working models, Dandori is a retention and recruitment advantage.

# 7. The Strategic Argument

Organizations that learn to operate in a spec-driven, AI-augmented model faster than their competitors gain a compounding advantage. It is not just that they ship faster. It is that they develop organizational muscle in the activities that remain uniquely human: precise specification, rapid decision-making, effective AI collaboration, and quality judgment. Those capabilities compound over time in ways that raw coding speed does not.

The analogy is the transition from waterfall to agile in the 2000s. The companies that adopted agile early did not just deliver faster. They developed a culture of iteration, feedback, and adaptability that became a durable competitive advantage over the following decade. The companies that adopted late spent years catching up not just on practices but on the cultural and skill foundations that made those practices work.

The same dynamic is playing out now. The organizations that figure out how to work effectively with AI as a team member, not just as a tool, will build a capability gap that is very hard to close. Dandori is the codification of that capability. Adopting it is not just a process improvement. It is an investment in organizational learning.

# 8. The Philosophical Argument

Scrum was a humane response to the software crisis of the 1990s. It said: stop treating developers as interchangeable resources in a waterfall machine. Give them autonomy, self-organization, and a sustainable pace. That was the right philosophy for its era, and the values it championed remain sound.

But Scrum's practices assumed a specific production model: teams of humans writing code in iterative cycles. When that model changes, when AI joins the team as a participant capable of handling significant implementation work, the practices must change even if the values do not. Dandori is the humane response to the AI transformation of the 2020s. It says: stop applying human-only coordination frameworks to human-AI teams. Recognize that when AI handles a growing share of implementation, the uniquely human contributions are thinking, judgment, creativity, and decision-making. Build your process around amplifying those contributions.

Equally, Dandori is a response to the honest observation that Scrum's practices had already drifted far from its values in most organizations. The original values of transparency, inspection, and adaptation are preserved in Dandori. What changes is the mechanism for achieving them: specifications instead of stories, pipeline visibility instead of sprint commitments, continuous calibration instead of periodic ceremonies that have lost their meaning.

This is not anti-Scrum. It is post-Scrum. It inherits Scrum's respect for the humans in the system, acknowledges that the system now includes AI participants, and updates the operating

model for a reality that is both organizationally different and technologically different from the one Scrum was designed to address.

# 9. The Problems Scrum Never Solved

Beyond Scrum's drift and AI's emergence, there are structural problems that Scrum never addressed. Most teams have learned to live with them. Dandori does not ask them to.

## The Visibility Problem

One of the most common complaints from business stakeholders about Scrum is the lack of visibility. A sprint starts. Two weeks pass. At the sprint review, the team demonstrates what they built. Sometimes it is what the stakeholder expected. Sometimes it is not. The sprint is a black box. Even stakeholders who attend every ceremony cannot confidently answer the question: "What will I have on this date?"

This is not a failure of discipline. It is a structural property of how Scrum works. The sprint is designed to protect the team from mid-sprint changes, which means the business cannot see or influence what is happening inside the sprint without violating the framework's own rules. Scrum chose team stability over business visibility.

In Dandori, work is visible at every stage because the pipeline is the coordination mechanism, not the sprint boundary. The business can see how many intents are in the queue, which specs are being written, which are in execution, which are in validation, and what integrated this week. Instead of "what will I get at the end of this sprint?" the question becomes "what is the cycle time for a spec of this complexity, and how many are ahead of this one?" That is a more honest answer, and it does not require waiting until a sprint review to discover the truth.

## The Non-Developer Problem

The Scrum Guide defines one role for everyone who does the work: "Developers." Whether you design interfaces, test software, architect systems, or write code, Scrum calls you a Developer. The framework has no model for specialized roles that contribute on fundamentally different timelines.

In reality, software teams include UX designers who need to research and prototype before engineers build, architects who need to define patterns that span multiple features, QA engineers who need to validate quality across dimensions beyond acceptance criteria, UAT specialists who need to verify business requirements in realistic scenarios, and technical writers who need to document what was built. Scrum's answer is: put everyone on the same team, work in the same sprint. In practice, either the specialized roles get compressed into the sprint cadence (designers rush, QA gets squeezed into the last two days, UAT happens after the sprint when nobody is paying attention) or they work on a parallel timeline that drifts out of sync.

Dandori's lifecycle has natural integration points for every discipline because the specification is where all perspectives converge before execution begins. UX designers contribute interaction patterns and accessibility requirements to the spec. Architects contribute interface contracts and performance constraints. QA engineers contribute test scenarios and edge cases. UAT specialists contribute business validation criteria. All of these become part of the specification before execution starts, not separate deliverables on separate timelines.

The Pre-Mortem ceremony is particularly valuable for cross-discipline teams. When a designer, an architect, a QA engineer, and a Spec Owner sit together for 20 minutes and ask "what could go wrong with this spec?" they surface cross-discipline risks that no single perspective would catch.

## The Parallel Workstream Problem

This is perhaps the most expensive problem Scrum never solved. The pattern is painfully familiar. Designers work ahead of the sprint, creating designs for features engineers will build later. By the time engineers start building, the designs may have changed. Or the engineers discover the design is technically infeasible. Or the designers have moved on and are not available to answer questions.

The same pattern plays out with QA. Engineers build during the sprint. QA tests after. By the time QA finds issues, the engineers have moved on to the next sprint's work. Fixing the issues requires expensive context-switching or creates technical debt. And with UAT, stakeholders validate weeks later, discovering that what was built does not match what they needed, not because of engineering mistakes but because requirements were ambiguous and assumptions diverged.

In all three cases, the root cause is the same: work that should be synchronized is happening on parallel timelines, and the sprint boundary does not synchronize them. The sprint synchronizes the engineering work. Everything else floats alongside it, loosely coupled and frequently misaligned.

In Scrum, the timeline looks like this: Design researches and prototypes, then hands off to the sprint. Engineers build during the sprint. QA tests after the sprint. UAT validates even later. Each workstream operates semi-independently. Design changes after handoff cause rework. QA after build creates feedback delay. UAT after everything creates the most expensive feedback delay of all.

In Dandori, all disciplines converge on the specification before execution begins. The designer's research becomes part of the Intent. The designer's prototype becomes part of the Specification. The QA engineer's test scenarios become acceptance criteria. The UAT requirements become validation criteria. When a designer wants to change a specification after execution has started, the change is visible. It is a spec revision, not an informal update that catches engineers by surprise. The team can assess the cost of the change rather than discovering the misalignment after delivery.

## The Spec as the Synchronization Point

The fundamental insight is that Scrum used time (the sprint) as the synchronization mechanism, and it only works for people who operate on the same timeline. Designers, QA, UAT, and architects do not operate on the engineering timeline. Forcing them into it creates the parallel workstream problem.

Dandori uses the specification as the synchronization mechanism. Everyone contributes to the spec. Everyone validates against the spec. The spec holds the complete picture of what needs to be built, including design, architecture, quality, and business criteria. When the spec is complete, everyone has already agreed on what "done" looks like. This does not eliminate all rework, but it eliminates the most expensive category: the kind that happens because different disciplines were working from different understandings of what was being built, on different timelines, with different artifacts.

These are not implementation failures. Scrum could not solve these problems because they are structural consequences of its core design decisions. The sprint as atomic unit created the visibility problem. "Developers" as the sole role created the non-developer problem. The sprint boundary as synchronization mechanism created the parallel workstream problem. Dandori makes different decisions and addresses all three.

## PART II

# WHAT IS DANDORI

*The methodology defined*

# 9. Core Principles

Dandori rests on four foundational principles.

## The Specification Is the Unit of Work

In Scrum, the unit of work is the user story. In Dandori, it is the specification. A specification is a structured, precise document that serves simultaneously as the design document, the AI execution instruction, the review criteria, and the team communication medium. Every other element of the methodology exists to optimize the creation, execution, and validation of specifications.

## Human Judgment Is the Scarce Resource

Scrum treated developer time as the scarce resource and built its entire coordination model around allocating and protecting that time. Dandori treats human judgment as the scarce resource: the ability to specify precisely, decide quickly, review effectively, and maintain architectural coherence across a system that AI agents are changing rapidly. Every ceremony, role, and practice in the methodology exists to amplify and protect human judgment.

## Preparation Determines Quality

This is the principle that gives the methodology its name. In the Toyota Production System, dandori refers to the meticulous preparation work done before production begins. Toyota discovered that investing in better preparation reduced waste, rework, and cycle time far more than optimizing the production line itself. The same principle applies to AI-augmented development: the quality of the specification determines the quality of the output. Investing in better specs pays more than investing in better AI tools or faster execution.

## Flexible Constraints

A framework has to be flexible, but it also has to restrict. If you push here, there will be a constriction. If you push there, there will be an expansion, but it will only fit a certain amount. This is the design principle that separates a framework from a recipe book and from anarchy.

Scrum failed not because it was too rigid but because it was too permissive. It defined ceremonies, roles, and artifacts but allowed organizations to reinterpret every one of them until "doing Scrum" meant whatever you were already doing. The flexibility that was supposed to make Scrum adaptable made it meaningless.

Dandori takes a different position. Certain elements are non-negotiable: the separation between Spec Owner and Reviewer, the Pipeline Sync, the Spec Handoff, the Integration Review. These exist because removing them breaks the quality model. Other elements are adaptive: sprint boundaries, ceremony frequency, the specifiability classification thresholds. These can be adjusted based on data and team experience.

The distinction is explicit. The ceremony classification system (Non-Negotiable, Required with Adaptive Frequency, Conditionally Required, Earned Optional) is the mechanism by which Dandori says "you can adjust here, but not there." Each constraint exists because a specific failure mode occurs without it. Each flexibility exists because different teams legitimately need different configurations.

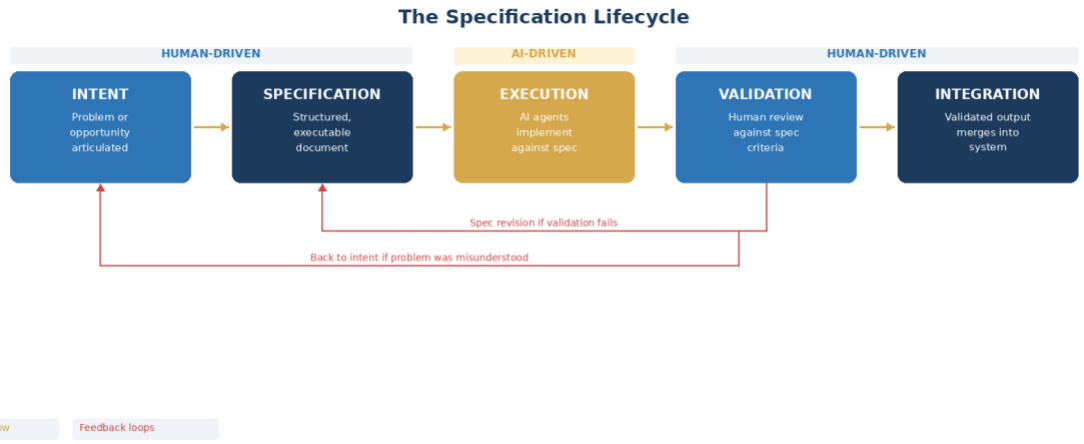
This is how Dandori avoids Scrum's fate. By being explicit about what bends and what does not, the methodology maintains its coherence even as teams adapt it to their context. If adoption can mean anything, adoption means nothing. Dandori defines what adoption means.

# 10. The Specification Lifecycle

Instead of Scrum's sprint cycle, Dandori follows the lifecycle of a specification through five stages. These are not phases in a waterfall sense; multiple specs move through them concurrently, and the flow is continuous.

| Stage | Purpose | Output |
|---|---|---|
| **Intent** | Articulate the problem or opportunity in plain language with enough business context to reason about constraints. | A clear statement of what needs to change and why, with measurable success criteria and explicit constraints. |
| **Specification** | Formalize intent into a structured, executable document precise enough for AI agents to implement against. | Requirements in structured notation, architectural decisions, interface contracts, constraints, and testable acceptance criteria. |
| **Execution** | AI agents implement against the spec. Human role is minimal and supervisory. | Working code, tests, and documentation generated by AI agents following the specification. |
| **Validation** | Human review of AI output against the spec's acceptance criteria and broader system coherence. | Approved output ready for integration, or feedback that cycles back to the Specification or Intent stage. |
| **Integration** | Validated output merges into the system. The spec repository updates so future specs can reference prior decisions. | Shipped feature with living documentation. The spec becomes a permanent record of design decisions. |

The critical discipline is that if execution produces unexpected results, the team goes back to the specification, not to the code. The instruction gets fixed, not the output. This inverts the Scrum-era instinct of debugging code and replaces it with debugging specifications.




# Spec-Driven Development in Practice

Before diving into the methodology's roles and ceremonies, it is essential to understand what spec-driven development looks like in practice: what a specification is, how it differs from a user story, and how work flows through the Dandori lifecycle with a concrete example.

## What Is Spec-Driven Development?

Spec-driven development is the discipline of writing structured, precise specifications before implementation begins, and treating those specifications as the primary artifact around which all coordination happens. It draws from formal methods, design-by-contract, and behavior-driven development. What makes it newly relevant is that AI execution engines can now implement against well-written specifications with high fidelity, making the specification the most leveraged artifact in the development process.

The core difference from traditional agile approaches is precision. A user story says "As a customer, I want to filter search results by price so I can find affordable options." That is enough for a human developer who can interpret intent, ask clarifying questions, and make reasonable assumptions. It is not enough for an AI agent, and frankly, it was never enough for consistent outcomes even with human developers. Teams that relied on vague stories compensated with verbal conversations, tribal knowledge, and implicit assumptions that new team members had to rediscover.

A specification makes all of that explicit. It captures not just what the user wants, but the constraints, the business rules, the edge cases, the acceptance criteria, and the architectural decisions that shape how the feature gets built.

## What a Specification Looks Like

There is no single "correct" format for a specification. What matters is that the spec is precise enough that someone (human or AI) can implement against it without guessing, and that a reviewer can validate the output against explicit criteria. Here is an example for a realistic feature: adding a price filter to a travel search results page.

### Example: Price Range Filter for Search Results

**Spec ID:** SEARCH-042

**Intent:** Customers searching for flights frequently abandon results pages when prices are higher than expected. Adding a price range filter allows customers to narrow results to their budget, reducing abandonment and increasing booking conversion on price-sensitive routes.

**Requirements:** The search results page must display a price range filter that allows the customer to set a minimum and maximum price. The filter must apply to the total trip price including taxes and fees, not the base fare. The filter must update results in real time without requiring a page reload. When the filter is active, results outside the price range must be hidden, not removed from the underlying dataset, so the customer can adjust the range and see results reappear. The filter must display the price distribution as a histogram above the slider.

**Constraints:** The filter range must be bounded by the minimum and maximum prices in the current result set, not hardcoded values. Currency must match the customer's selected currency. Filter state must be preserved when the customer navigates to a booking page and returns using the browser back button. Performance: filtering must complete in under 100ms for up to 500 items. The filter must be keyboard navigable and screen-reader compatible.

**Acceptance Criteria:** (1) Given prices ranging from 50 to 800 EUR, the slider must show the full range on load. (2) Setting maximum to 200 EUR must hide results above 200 within 100ms. (3) Browser back button must restore filter state. (4) Filtering 500 items must complete in under 100ms. (5) Screen reader must announce the new range and visible result count. (6) The histogram must reflect results after other active filters are applied.

**Architectural Decision:** Client-side filtering is chosen over server-side because the full result set is already loaded and round-trip latency would degrade the real-time experience. If future requirements include paginated results, this decision must be revisited.

This specification is longer than a user story. That is the point. The time invested in writing it is time saved in implementation (no guessing), in review (explicit criteria to check against), and in debugging (failures trace to specific requirements that were missed).

## How This Flows Through Dandori

### Stage 1: Intent

The product manager identifies that price-sensitive routes have a 23% higher abandonment rate. They write an intent document with the problem statement, success criteria (reduce abandonment by 10 percentage points within 60 days), and constraints (must not increase page load time by more than 200ms). This intent enters the queue at the **Prioritization Reset**. The Prioritizer classifies it as Clear and assigns a Spec Owner.

### Stage 2: Specification

The Spec Owner writes the specification shown above. During this stage, they encounter one ambiguity: should the histogram show the full result set or the set after other filters are applied? This triggers a **Decision Request** to the PM, structured as: decision needed, options (A: full set, simpler; B: filtered set, more accurate), recommendation (B), and what is blocked. The PM responds within 4 hours (the tactical SLA): Option B confirmed. The spec is finalized.

### Stage 3: Execution

The completed spec goes through a **Spec Handoff** with the Reviewer (10-15 minutes). The Spec Owner highlights the architectural decision, accessibility requirements, and performance constraint. The Reviewer confirms understanding. The spec then goes to execution. In a team using AI agents, the spec is the implementation instruction. In a team without AI, a developer implements against the spec. In either case, the specification is the single source of truth.

### Stage 4: Validation

The Reviewer evaluates the output against the six acceptance criteria. Four pass. Two fail: browser back button does not restore state (the component does not persist to the URL), and the screen reader uses the wrong aria-live mode. The Reviewer does not fix the code. The spec goes back to Execution with the specific failures noted. The Spec Owner adds a note to the team's spec patterns: any filter component must specify URL state persistence behavior. The second pass passes all six criteria.

### Stage 5: Integration

The validated component merges. The specification is archived as a permanent record. Future specs touching the search results page can reference SEARCH-042 for context.

This entire cycle took three days. Under Scrum, the same feature would have been a story estimated at 5-8 points, discussed in sprint planning, potentially carried across sprints, and delivered with implicit assumptions about accessibility and performance that might not match what the PM intended.

## How It Appears in the Weekly Ceremonies

At the **Integration Review**, the team examines SEARCH-042 alongside other specs shipped that week. They note the first-pass failure on two criteria, which is normal. Spec quality was good because the criteria caught real issues. The team's weekly first-pass success rate was 71%, within the target range.

At the **Spec Retrospective**, the team discusses the URL state persistence miss. This is not the first time a client-side component spec has missed browser navigation behavior. They add a new pattern to their spec library: any component that modifies the visible result set must specify behavior on browser back and forward navigation. This pattern prevents recurrence.

## The Spec Is Not the Code

The specification is not pseudocode. It does not describe implementation steps. It describes what the output must do and the constraints it must satisfy. The "how" is left to the implementer (human or AI), except where architectural decisions are explicitly captured. This is what makes specs reusable: if the team rewrites the component in a different framework, or if AI models improve, the specification still applies. The spec captures intent, constraints, and criteria. The code is one possible expression of those.

## Spec-Driven Development Without AI

Everything in this example works identically if the Execution stage is handled by a human developer. The developer receives the same specification, implements against the same criteria, and the Reviewer validates against the same acceptance tests. The difference is speed: a human developer might take two days where an AI agent takes two hours. But the

coordination model, the roles, the ceremonies, and the quality outcomes are the same. This is why Dandori works for teams at any point on the AI adoption spectrum.

# 11. Roles

Three roles replace Scrum's Scrum Master, Product Owner, and Developer triad.

## Spec Owner

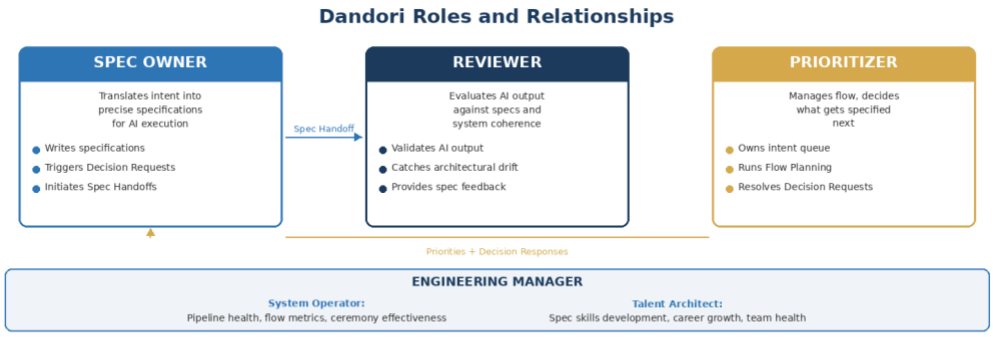
The Spec Owner combines product thinking with technical precision. This person translates business intent into specifications that AI can execute without ambiguity. They do not write code; they write the instructions that produce code. This role requires both domain expertise and an understanding of what AI agents can and cannot do well, a new and genuinely scarce skill. In most organizations, this maps to senior engineers or tech leads.

## Reviewer

The Reviewer is the quality gate. They evaluate AI output against specs and against broader system coherence. They catch the things specs cannot fully express: architectural consistency, performance implications, security concerns, edge cases the specification did not anticipate. The review discipline changes in Dandori because evaluating AI-generated code has different failure patterns than evaluating human-written code. AI code tends to be locally correct but globally inconsistent; it solves the specified problem but may not respect unwritten system conventions.

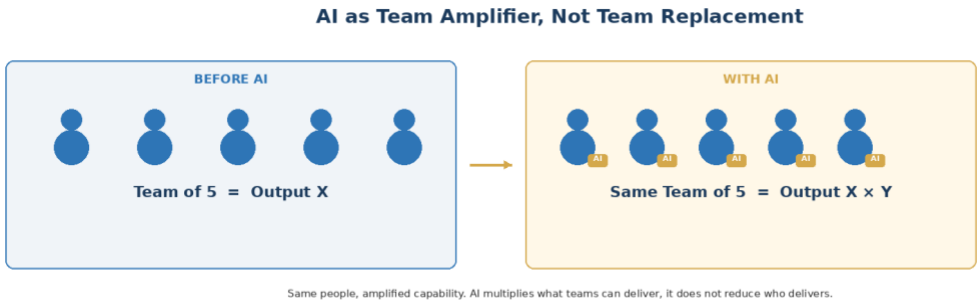
## Prioritizer

The Prioritizer replaces the Product Owner and Scrum Master combined. They manage the flow of specs through the pipeline, decide what gets specified next, and ensure the team is not optimizing locally while missing the bigger picture. This is a leadership function shared between the engineering lead and the product manager. The engineering lead owns technical sequencing and dependency management. The product manager owns business priority and success criteria. They negotiate continuously at the Intent stage, before specs are written, rather than in batch planning sessions.




| Dimension | Scrum | Dandori | Key Shift |
|---|---|---|---|
| Unit of work | User story | Specification | From intent description to execution instruction |

| | | | |
|---|---|---|---|
| Scarce resource | Developer time | Human judgment | From labor allocation to thinking amplification |
| Time model | Fixed sprints | Adaptive flow (sprints optional) | From rigid timebox to adaptive cadence |
| Estimation | Story points | Specifiability classification | From effort prediction to complexity triage |
| Primary metric | Velocity | Flow efficiency | From throughput of labor to throughput of judgment |
| Team size | 5-9 mixed seniority | Same team, amplified output | From skill coverage to judgment + AI collaboration |
| Coordination | Scrum Master | System Operator (EM) | From process facilitation to pipeline optimization |




## 11.4 How Roles Evolve

A common misconception is that AI-augmented development eliminates the need for junior engineers or demands that all team members be senior. The reality is more nuanced and more interesting.

The need for junior engineers remains. Every team needs people who are learning, growing, and bringing fresh perspectives. What changes is what "junior" means. A junior engineer in Dandori is not someone who writes simple code under supervision. They are someone who is learning to write precise specifications, developing judgment about what AI output to trust and what to question, and building the domain knowledge that makes specifications accurate. These are learnable skills, and junior engineers bring something valuable to the learning process: they ask the questions that reveal assumptions seniors have stopped noticing.

Equally important is the recognition that many senior engineers are, in a real sense, junior again. An engineer with fifteen years of experience writing code is a beginner at writing specifications for AI execution. They are a beginner at reviewing AI-generated code, which has different failure patterns than human-written code. They are a beginner at collaborating with AI agents effectively. The seniority that matters in Dandori, the ability to specify precisely, review critically, and make architectural decisions in an AI-augmented context, is new enough that almost everyone is still developing it.

This is healthy. It levels the playing field in a way that creates opportunity rather than threat. It means that career growth is available to everyone on the team, not just those climbing a traditional coding ladder. And it means that the team's collective capability grows as everyone, junior and senior alike, gets better at the skills that Dandori values: thinking clearly, specifying precisely, and judging wisely.

# Team Size: Where Dandori Works and Where It Breaks

Every framework has a range. Scrum says 5-9 people per team. Kanban is more flexible but assumes a single team managing a shared board. Dandori has its own range, and being honest about where it works, where it stretches, and where it breaks is more useful than claiming it works for everyone.

## The Minimum: 3 People

Dandori has three roles: Spec Owner, Reviewer, and Prioritizer. The critical constraint is that the Spec Owner and the Reviewer must be different people. If the person who writes the spec also reviews the AI output, the entire quality model collapses. They will unconsciously validate what they intended rather than what was produced.

The Prioritizer can be one of the two engineers wearing a second hat, or it can be the PM. So the absolute minimum is two engineers plus someone handling prioritization. That is three people. Below three, an individual can use spec-driven development practices, but they cannot run Dandori because the methodology's quality model depends on the separation between specification and review.

## The Sweet Spot: 4-8 People

This is where Dandori works best. The team is large enough that roles are distinct (multiple Spec Owners, at least one or two dedicated Reviewers, a clear Prioritizer), and small enough that the ceremonies scale naturally. The Pipeline Sync takes 15 minutes. The Spec Retrospective works well with Troika Consulting. The Integration Review covers the week's output without rushing. The Prioritizer can hold the full pipeline context in their head.

Cross-discipline teams fit well at this size. A team of 4-8 might include three or four engineers, a UX designer who contributes to specifications, a QA engineer who shapes acceptance criteria, and a PM who shares the Prioritizer role with the engineering lead.

## Stretched but Functional: 9-12 People

At this size, Dandori still works but requires adjustments. The Pipeline Sync needs tighter facilitation and the 1-2-4-All Liberating Structure becomes essential. The Spec Retrospective may need rotating subgroups instead of full-team attendance every week. The Prioritizer role strains because one person cannot hold the context of 15-20 specs in flight, so lightweight tracking systems become necessary. The Intent Review becomes more important because shared context does not happen organically in a group this large.

## Where Dandori Breaks: Beyond 12

Beyond 12 people, the single-team model breaks for specific reasons. The pipeline becomes too large for any one person to hold. Ceremonies become broadcasts rather than working sessions. Architectural coherence checked in a single weekly session degrades when the volume of changes is too high.

The likely shape of a scaling approach involves splitting into smaller teams (each in the 4-8 range), adding a cross-team synchronization ceremony for dependencies and shared specs, a periodic cross-team Integration Review for architectural coherence, and a shared spec pattern library. The methodology deliberately does not prescribe a scaling model like SAFe or LeSS because these carry overhead that may not apply. The scaling approach should emerge from the experience of teams that outgrow the single-team model.

## Team Size and AI Amplification

AI amplification changes the relationship between team size and output. A team of 5 using AI effectively may produce the output that previously required a team of 12. This means many teams that currently feel too large for a single-team framework may not need to scale Dandori at all. They may find that their effective team size sits comfortably in the 4-8 sweet spot where Dandori works best. This is not a recommendation to reduce headcount. It is an observation that the coordination model for a team of 8 doing the work of 15 is still a single Dandori pipeline.

# 12. Product Management Alignment

In Scrum, product management and engineering meet at the user story. The PM writes stories, the team estimates and commits, and the sprint creates a shared contract. The story is deliberately imprecise, describing what the user needs without prescribing how, and the team fills in the gap during implementation.

With AI execution, that imprecision becomes the primary source of waste. An AI agent cannot ask clarifying questions mid-implementation. It executes literally against whatever it is given. The interface between product and engineering must get much more precise, much earlier.

## From Stories to Intent Documents

The PM's output shifts from user stories to structured intent briefs. An intent document includes a clear problem statement with measurable success criteria, explicit constraints and business rules, priority signals that help the Spec Owner make architectural tradeoffs, and a definition of what "good enough" looks like for the first iteration versus the long-term vision. This is not a heavyweight PRD. It is a structured brief that gives the Spec Owner enough context to write a precise specification without reverse-engineering the PM's thinking.

## The Decision Queue

Instead of batching decisions into planning sessions, the PM maintains a continuously prioritized queue of decisions that need to be made. Some are strategic, some are tactical, and some are clarifications. The Spec Owner pulls from this queue whenever a spec needs a product decision, and the PM's job becomes keeping that queue current and unblocked. This mechanism replaces the sprint planning conversation as the primary interface between product and engineering.

## Discovery and Delivery Merge

When delivery becomes cheap and fast through AI execution, the economics of product discovery change completely. Teams can spec and build a prototype in hours rather than spending weeks on mockups and research to decide if something is worth building. The PM's role evolves from curating a carefully sequenced backlog to becoming a sense-maker who interprets rapid experimental results and adjusts direction. The fastest way to validate an idea is now building a working version and putting it in front of users. This is a more dynamic and arguably more rewarding product role, one that demands sharper judgment and faster pattern recognition.

## A Shared Challenge: Precision at the Interface

Spec-driven development changes the tolerance for ambiguity at the product-engineering interface. In Scrum, ambiguity in a user story was absorbed by the development team during implementation. Developers interpreted intent, made reasonable assumptions, and course-corrected during the sprint. This worked because humans are good at filling gaps. When AI executes against a specification, that ambiguity is not absorbed. It is amplified. The AI builds exactly what the spec says, and if the spec was imprecise, the output will be precisely wrong.

This raises the bar for everyone involved in the specification process. PMs need to externalize decisions that used to emerge implicitly during implementation conversations. Engineers need to ask sharper questions when translating intent into specs. The Spec Owner role exists specifically to bridge this gap. The result, when it works well, is that product intent becomes

more visible, more testable, and more aligned with what actually gets built. Both product and engineering benefit from the increased clarity.

# 13. The Engineering Manager's Role

The EM role does not just evolve in Dandori. It splits into two fundamentally different functions, and most EMs are only prepared for one of them.

## The System Operator

This is the person who owns the flow of the spec-driven pipeline. They monitor the health of the intent-to-spec-to-execution-to-validation cycle. They identify where the system is clogged: are specs waiting on product decisions? Is validation becoming a bottleneck because reviewers are overloaded? Are certain types of specs consistently producing poor AI output, indicating a spec pattern problem? They optimize the throughput of the overall system, not the productivity of individual humans. This function is much closer to what a TPM or delivery lead does today than what most EMs do. It requires systems thinking, data fluency, and the ability to see the whole pipeline.

## The Talent Architect

This is the people-development side, radically redefined. The EM is not developing people's coding skills; AI is commoditizing that. The EM is developing people's specification skills, their judgment in reviewing AI output, their ability to make architectural decisions under uncertainty, and their capacity to work effectively with AI agents as collaborators. Career ladders must change as well. "Senior engineer" can no longer mean "writes complex code well" when AI writes complex code. It must mean "produces specifications that consistently result in correct, maintainable AI output" and "catches the failures that automated testing misses."

## Where the EM Gains Power

The EM becomes the primary quality governor of the entire system, tracking spec failure patterns, identifying which engineers write specs that AI executes well versus poorly, and calibrating the team's overall spec maturity. The EM also becomes the bridge between the AI capability frontier and organizational adaptation. Leadership wants to know: how much more can we do with AI? Where are the limits? What should we invest in? The EM is the only person with enough ground-level visibility into AI execution patterns and enough organizational context to answer those questions honestly.

## The Evolving Scope

In Dandori, the EM's scope expands. Because AI amplifies what each team can deliver, the coordination surface grows even if headcount stays constant. A team that previously shipped three features per sprint might now ship ten per week, and each one needs specification, review, and integration oversight. The EM is not managing fewer people. The EM is managing a faster, more complex pipeline with the same people. That requires a higher level of systems thinking and a different set of management instincts than Scrum demanded.

# 14. Relationship to Existing Methodologies

## 14.1 Relationship to Agile

Dandori is fundamentally Agile in its values. It prioritizes working software over documentation, responding to change over following a plan, and individuals and interactions over processes and tools. Where it diverges from Agile orthodoxy is on one specific point: Agile favors "just enough" upfront design and trusts that working software emerges through iteration. Dandori inverts this, investing heavily in upfront specification because the cost of iteration has changed. When AI builds the wrong thing, the execution time lost is minutes, but the review and debugging time untangling something that looked right but was not can be hours. The economics now favor getting the spec right.

Both positions are empirically driven responses to the economics of their era. Agile was right for 2001. Dandori is right for 2026. The underlying principle is the same: optimize for where the actual cost and risk live.

## 14.2 Parallels with Scrum

Dandori preserves several things from Scrum that were genuinely good ideas, even if it changes their form. Scrum's definition of done survives as acceptance criteria embedded in every specification, but written for AI agents that execute literally rather than humans who interpret intent. The sprint review survives as the weekly Integration Review, shifted from biweekly demos to weekly system health checks. The retrospective survives but changes target, from team dynamics to human-AI collaboration patterns. Scrum's role clarity survives fully, with Spec Owner, Reviewer, and Prioritizer providing clear accountability.

## 14.3 Differences from Scrum

The most fundamental difference is that Scrum's atomic unit is time (the sprint) while Dandori's atomic unit is the specification. This changes the downstream mechanics significantly. The fixed sprint commitment gives way to a continuously reprioritized queue of specifications at various lifecycle stages, though teams may retain sprint boundaries as adaptive timeboxes during transition. Estimation transforms from story points into specifiability assessment, a simpler classification that drives process decisions rather than capacity negotiation. The team composition evolves to emphasize specification and AI collaboration skills alongside traditional engineering skills. The sprint backlog is supplemented or replaced by a live pipeline view showing specs at each lifecycle stage.

## 14.4 Parallels with Kanban

This is where Dandori has the strongest kinship. Kanban's focus on flow over batching is exactly what Dandori does. The lifecycle stages function as Kanban columns with explicit entry and exit criteria. WIP limits translate directly but apply differently, preventing the review bottleneck from growing unbounded rather than preventing human overload. Kanban's emphasis on measuring flow becomes the primary management instrument. Kanban's evolutionary "start with what you do now" approach applies to adoption.

## 14.5 Differences from Kanban

Kanban is deliberately minimal on roles and practices. Dandori is more opinionated, defining specific roles, a specific lifecycle, and specific cadences. This is necessary because the human-AI collaboration introduces failure modes that pure Kanban does not address. Kanban also does not address the product-engineering interface or the nature of the work items flowing through the board. Dandori explicitly includes the Intent stage and product alignment mechanisms because the PM's ability to produce clear intent is the upstream constraint on the whole system.

## 14.6 The Synthesis

Dandori inherits Agile's philosophy, borrows Kanban's flow mechanics, preserves Scrum's role clarity and cadence discipline, and adds a new layer that none of them had: the specification as a first-class artifact that serves simultaneously as the design document, the AI execution instruction, the review criteria, and the team communication medium.

The genuinely new contribution is the recognition that AI execution changes which human activities are valuable. Agile, Scrum, and Kanban all optimized for a world where human implementation was the expensive part. Dandori optimizes for a world where human thinking, deciding, and judging are the expensive parts, and implementation is nearly free.

# How Traditional Documents Fit Into Dandori

Software teams produce and consume many documents beyond code: PRDs, ADRs, RFCs, design documents, test plans, runbooks, and others. Dandori does not eliminate them. It repositions them within the specification lifecycle so they serve clear purposes at clear stages rather than floating as disconnected artifacts.

## PRD (Product Requirements Document)

The PRD maps directly to the **Intent** stage. It is the structured articulation of what needs to change and why. However, Dandori's Intent document is leaner than a traditional PRD. It focuses on the problem statement, measurable success criteria, explicit constraints, and priority signals. It does not specify the solution, because solution design happens in the Specification stage.

For large initiatives spanning multiple specifications, the PRD serves as a parent document that frames the overall problem, and individual Intent documents reference it. The PRD answers "what is the initiative and why does it matter?" Each Intent answers "what is this specific piece?" Each Specification answers "what exactly are we building for this piece?"

## ADR (Architecture Decision Record)

ADRs are embedded directly in specifications. Every specification has an Architectural Decision section that captures what was decided, what alternatives were considered, and why this option was chosen. This means ADRs are created at the point of decision (during Specification), reviewed at the point of validation (the Reviewer checks whether the implementation respects the decision), and archived at the point of integration (the spec becomes the permanent record). The decision never drifts from the code it influences because it lives in the same artifact.

For architectural decisions that span multiple specifications, a standalone ADR still makes sense. But it functions as a constraint that individual specifications reference, not as a disconnected document.

## RFC (Request for Comments)

RFCs map to the transition between Intent and Specification, specifically for **Complex** and **Uncertain** work that needs broader input before a specification can be written. When a Spec Owner encounters a problem too complex to specify alone, they write an RFC that circulates during the **Intent Review** ceremony. Once the RFC reaches consensus, it becomes the foundation for one or more specifications.

The critical difference: in Dandori, an approved RFC immediately enters the specification queue. It does not float indefinitely. The Prioritizer ensures that approved RFCs are either scheduled for specification or explicitly deprioritized with a reason. This closes the gap between "we agreed this is a good idea" and "someone is actually working on it."

## Design Documents and Artifacts

Design documents are absorbed into the specification. The Specification is the design document. For complex features requiring extensive design exploration, the work happens as an exploration spike during the Uncertain classification path. The output is a specification, not a separate design doc. UX artifacts (wireframes, prototypes, interaction specifications) feed into the Specification stage as inputs the Spec Owner incorporates. They are not separate deliverables floating alongside the spec.

## Test Plans

Test plans are embedded in the specification as acceptance criteria. The QA engineer contributes test scenarios, edge cases, and quality dimensions to the spec during the Specification stage, before execution begins. For system-level testing that spans multiple specifications, a separate test plan is appropriate, but it references the specifications it covers. Exploratory testing happens during Validation alongside criteria-based validation.

## Runbooks and Operational Documentation

Operational documentation is an output of the **Integration** stage. The spec itself becomes the primary operational record. For concerns beyond the spec (monitoring, alerting, incident response), operational documentation is produced as part of the integration checklist. The specification repository becomes the living documentation of the system.

## The Specification as Connective Tissue

The pattern across all of these documents is that the specification serves as the connective tissue. It references the PRD for business context, embeds the ADR for architectural decisions, incorporates design artifacts for UX requirements, includes test criteria for quality validation, and produces operational documentation for system maintenance. In Scrum, these documents existed as independent artifacts with implicit connections. In Dandori, the specification makes these connections explicit. Nothing floats. Everything connects.

| Document | Where It Lives | When Created | Who Owns It |
|---|---|---|---|
| **PRD** | Intent stage (parent for initiatives) | Before intents are written | PM |
| **Intent** | Intent stage | When a problem is identified | PM or Spec Owner |
| **RFC** | Between Intent and Specification | When work is Complex/Uncertain | Spec Owner, reviewed by team |
| **Specification** | Specification stage (central artifact) | Before execution begins | Spec Owner + all disciplines |
| **ADR** | Embedded in spec, or standalone for cross-cutting | During specification writing | Spec Owner or Architect |
| **Test Plan** | Acceptance criteria in spec, or standalone for system-level | During specification | QA Engineer |
| **Runbook** | Output of Integration stage | During integration | Spec Owner or designated member |
| **Spec Patterns** | Output of Spec Retrospective | Ongoing, updated weekly | Team (maintained by EM) |

PART III

# HOW DANDORI WORKS

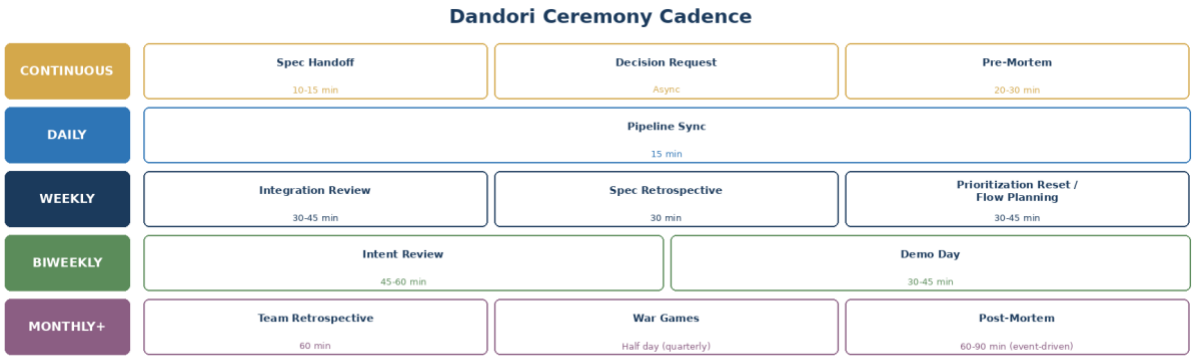
*Implementation and practice*

# 15. Ceremonies

Every ceremony in Dandori must earn its place by addressing a specific failure mode that emerges from AI-augmented spec-driven work. If it does not prevent a real problem, it is waste.

Ceremonies are organized across multiple frequencies: continuous, daily, weekly, biweekly, monthly, quarterly, and event-driven. No ceremony exists solely to satisfy the methodology's rules. If a ceremony stops surfacing useful information, it should be dropped. Teams running sprints will align some of these ceremonies with the sprint boundary; teams on continuous flow run them independently.

| Ceremony | Frequency | Duration | Participants |
|---|---|---|---|
| Spec Handoff | Continuous | 10-15 min per spec | Spec Owner + Reviewer |
| Decision Request | Continuous | Async (4-24hr SLA) | Spec Owner + Prioritizer/PM |
| Pre-Mortem | Per high-risk spec | 20-30 min | Spec Owner + Reviewer + domain expert |
| Pipeline Sync | Daily | 15 min | Full team |
| Integration Review | Weekly | 30-45 min | Full team + PM |
| Spec Retrospective | Weekly | 30 min | Spec Owners + Reviewers |
| Prioritization Reset / Flow Planning | Weekly | 30-45 min | Prioritizer + PM + senior Spec Owner |
| Intent Review | Biweekly | 45-60 min | Full team + PM |
| Team Retrospective | Monthly | 60 min | Full team |
| Demo Day | Biweekly/Monthly | 30-45 min | Team + stakeholders + adjacent teams |
| War Games | Quarterly | Half day | Senior engineers, cross-team |
| Post-Mortem | Event-driven | 60-90 min | Incident responders + relevant Spec Owners |




## 15.1 Ceremony Classification: Required, Adaptive, and Optional

Not every ceremony is mandatory at all times. Dandori avoids recreating Scrum's rigidity by classifying ceremonies into four tiers. Teams should start with all ceremonies active and earn the right to reduce frequency or omit specific ones based on data and team maturity.

| Tier | Ceremony | When It Can Be Reduced or Omitted | What Breaks If You Remove It |
|---|---|---|---|
| **Non-Negotiable** | Pipeline Sync | Never. This is the heartbeat of the system. | Specs pile up unnoticed. Bottlenecks go unseen. Flow stops. |
| | Spec Handoff | Never. Every spec entering Execution needs this. | Reviewers misunderstand intent. Correct AI output passes review because nobody caught the misalignment. |
| | Integration Review | Never. Can reduce from 45 to 30 min when stable. | Architectural drift across AI-generated changes goes undetected. System coherence erodes. |
| **Required, Frequency Adapts** | Spec Retrospective | Move to biweekly when spec first-pass success rate exceeds 75% and rework rate is below 15%. | Spec quality stops improving. Same failure patterns repeat indefinitely. |
| | Prioritization Reset / Flow Planning | Move to biweekly when the intent queue is stable and few new intents arrive each week. | Queue becomes stale. Team works on outdated priorities. Capacity misalignment grows. |
| **Conditionally Required** | Pre-Mortem | Only needed for Complex and Uncertain specs. Clear specs skip it entirely. | High-risk specs enter execution with unexamined assumptions. Expensive rework follows. |
| | Decision Request | Triggered by need, not scheduled. Cannot be omitted, only resolved faster. | Specs bake in unvalidated assumptions. AI output looks authoritative but encodes wrong decisions. |
| | Intent Review | Can pause when the team is small (2-3 people), working in a well-understood domain, with few new intents arriving. | Reviewers encounter specs without context. Shared understanding degrades. Spec quality drops. |
| **Earned Optional** | Team Retrospective | Move to quarterly after the first 6 months of adoption, once the team has stabilized emotionally and operationally. | Human side of the transition goes unprocessed. Disengagement and identity friction build silently. |
| | Demo Day | Can reduce to monthly or pause during low-output periods. Resume when stakeholder alignment needs reinforcing. | Human contribution becomes invisible. Morale suffers. Leadership loses sight of what the team delivers. |

| | | | |
|---|---|---|---|
| | War Games | Can move to semiannual for stable, well-understood systems. Return to quarterly after major architectural changes. | Systemic interaction risks between independently correct specs go undetected until production failure. |
| **Event-Driven** | Post-Mortem | Triggered by incidents only. Cannot be omitted when triggered. | Root causes go unexamined. Same failures recur. Spec patterns do not improve from real-world feedback. |

The principle governing all of these decisions is: **start with everything, reduce based on evidence.** A team that drops the Spec Retrospective because "we are too busy" is making a different decision than a team that drops it because their first-pass success rate has been above 80% for three consecutive months. The first is avoidance. The second is earned confidence. The EM's System Operator function includes monitoring whether ceremony reductions are justified by data or are signs of drift.

When in doubt, keep the ceremony. The cost of a 30-minute weekly meeting is low. The cost of losing a feedback loop that catches spec failures, architectural drift, or team disengagement is high and often invisible until it becomes a crisis.

## 15.2 Continuous Ceremonies

### Spec Handoff

Triggered every time a specification moves from the Specification stage to Execution. The Spec Owner walks the Reviewer through the spec before AI execution begins. The sole purpose is to ensure the Reviewer understands the intent well enough to evaluate the AI's output. In Scrum, this alignment happened implicitly because the same developer who understood the story also wrote the code. In spec-driven work, the person who wrote the spec and the person who reviews the AI output may not be the same person, and the AI will not ask clarifying questions mid-implementation.

This ceremony prevents the most expensive failure mode in the whole system: correct AI execution of a misunderstood spec that passes review because the reviewer did not grasp the intent.

**Anti-pattern to watch for:** the handoff becoming a rubber stamp. If the Reviewer never pushes back on spec clarity, either the specs are perfect (unlikely) or the ceremony has become performative. The EM should track how often handoffs surface spec issues versus how often they are just walkthroughs.

### Decision Request

Triggered whenever a Spec Owner hits an ambiguity that requires a product or architectural decision. This is not a meeting. It is a structured async request with a specific format: what decision is needed, what the options are, what the Spec Owner recommends, and what is blocked until it is resolved. The Prioritizer or PM has a defined SLA for responding: 4 hours for tactical decisions, 24 hours for strategic ones.

This ceremony exists because decision latency is the dominant blocker in spec-driven work. In Scrum, a developer could work around an ambiguity by making a judgment call and validating at

sprint review. In Dandori, baking an assumption into a spec without explicit resolution means the AI will execute on that assumption and the output will look authoritative even if the assumption was wrong.

### Pre-Mortem

Triggered for high-risk or high-complexity specs before they move to Execution. The Spec Owner convenes a 20-30 minute session with the Reviewer and any relevant domain expert. They assume the AI will execute the spec perfectly as written and then ask: what could still go wrong? What did we assume that might not be true? What edge cases did we not specify? What happens if this interacts badly with recent changes from other specs?

Not every spec warrants a pre-mortem. Simple, well-understood changes do not need the overhead. But anything touching core business logic, cross-system interfaces, or customer-facing behavior benefits from this ceremony. The Prioritizer or EM should flag which specs require a pre-mortem as part of the Prioritization Reset.

## 15.3 Daily Ceremony

### Pipeline Sync

This replaces the standup. Fifteen minutes, ideally async-first with a synchronous option for when the team needs discussion. The Pipeline Sync is structured around the five lifecycle stages, not around individual status. The team looks at the actual pipeline and answers: what moved forward since yesterday, what is stuck and why, and where is the highest-priority unblocked work right now.

The critical discipline is that this ceremony tracks specifications, not people. Nobody says "yesterday I worked on X." Instead, the board shows that Spec 47 moved from Execution to Validation, Spec 52 is blocked on a Decision Request pending for 18 hours, and Spec 39 failed validation and is back in Specification for revision. The EM facilitates by watching for systemic issues: are specs piling up in Validation? Is one person's Decision Requests consistently slow to resolve? Is a particular type of spec repeatedly failing execution?

If the board is healthy and nothing is stuck, the sync takes five minutes. Nobody needs to justify their existence by filling airtime.

## 15.4 Weekly Ceremonies

### Integration Review

Thirty to forty-five minutes weekly. The whole team plus the PM attends. This ceremony replaces both the sprint review and the sprint demo as an operational mechanism. The team examines everything that reached the Integration stage in the past week: what shipped, what the AI produced well, what required significant human correction, and what the product impact looks like.

The key difference from a sprint review is that this is not about celebrating delivery. It is about calibrating the system. The most important questions are: which specs produced clean AI output on the first pass? Which required multiple rounds of revision? What patterns emerge in specs that fail execution or validation? The PM evaluates whether the delivered output matches their intent, and any gaps feed back as learnings for how intent documents need to improve.

This is also where architectural coherence gets checked. AI agents execute individual specs in isolation. The Integration Review is where humans look at the collection of changes and ask whether the system as a whole still makes sense.

### Spec Retrospective

Thirty minutes, same day or adjacent to the Integration Review. Attendance is the Spec Owners and Reviewers; no PM required. This is a pure engineering ceremony focused on improving the spec-to-output pipeline.

The guiding question is: how do we write better specs? The team examines the week's spec failures, categorizes them (ambiguous requirement, missing constraint, wrong architectural assumption, AI model limitation), and identifies actionable improvements. Over time, this ceremony produces a growing body of spec-writing patterns that become the team's institutional knowledge.

**Anti-pattern to watch for:** the retro becoming a blame session about whose spec was bad. The EM must frame this as systems improvement. The question is never "who wrote a bad spec?" It is "what was missing from our spec patterns that allowed this failure?"

### Prioritization Reset with Flow Planning and Intent Triage

Thirty to forty-five minutes weekly, attended by the Prioritizer, the PM, and optionally one senior Spec Owner for technical context. This transforms sprint planning rather than replacing it entirely.

The ceremony has three components. First, the Prioritization Reset: the group reviews the current intent queue and re-ranks based on what was learned during the week, identifies intents ready to move into Specification, and retires or deprioritizes intents that no longer matter.

Second, Flow Planning: a lightweight capacity check that asks how many specs the team can write, execute, and validate in the coming period, given that some work requires significant human involvement. For teams running sprints, this aligns with the sprint boundary. For teams on continuous flow, it is a weekly calibration. The key difference from traditional sprint planning is that the team plans around judgment and review capacity, not implementation labor. How many specifications can we produce at sufficient quality? How much validation bandwidth do we have? Which items need human implementation and how does that affect the overall flow?

Third, Intent Triage: when new intents enter the queue, the group spends 10-15 minutes per intent assessing specifiability. Is this well-enough understood to specify? What unknowns need to be resolved first? Does this have dependencies on other in-flight specs? Is this one spec or actually three specs pretending to be one? Each intent receives a Specifiability Classification (Clear, Complex, or Uncertain) that drives downstream process decisions.

## 15.5 Biweekly, Monthly, Periodic, and Event-Driven Ceremonies

### Intent Review

Forty-five to sixty minutes biweekly. The full team plus the PM attends. This is the transformed version of backlog grooming, focused not on estimation but on collective sense-making. The team reviews upcoming intents that are likely to enter the Specification stage in the next one to two weeks.

The guiding question is: do we understand this problem space well enough that any of us could write a spec or review AI output for it? The team discusses the business context, technical

implications, potential risks, and cross-system dependencies. Each intent receives or updates its Specifiability Classification (Clear, Complex, Uncertain). Uncertain intents may trigger exploration spikes before specification begins.

This ceremony serves a function that cannot be replicated by smaller planning groups: it builds shared context across the team. A Reviewer who participated in an Intent Review about a feature two weeks ago will do a significantly better job evaluating AI output than one encountering the feature for the first time. The collective understanding also helps Spec Owners write better specs because they benefit from the team's diverse perspectives on edge cases, dependencies, and potential failure modes.

### Team Retrospective

Sixty minutes monthly. This is the human-centered ceremony that the Spec Retrospective deliberately does not cover. It addresses how the team is feeling, whether trust is intact, whether the pace is sustainable, and whether the shift to spec-driven work is creating anxiety or frustration. These questions matter more during a transition like this, not less.

Engineers who used to derive satisfaction from writing elegant code are now writing specs and reviewing AI output. That is a real identity shift, and if the EM does not create space to process it, it festers into disengagement. The format should lean toward facilitation techniques like Liberating Structures rather than the standard "what went well, what didn't" template. This is the one ceremony where the EM should be in Talent Architect mode, not System Operator mode.

### Demo Day

Thirty to forty-five minutes, biweekly or monthly. The audience expands beyond the immediate team and PM to include adjacent teams, leadership, and anyone who benefits from seeing what is being built. This is a cultural ceremony, not an operational one. It asks "what did we create, and can we be proud of it?"

The format change is important: Demo Day in Dandori should highlight the spec-to-outcome journey, not just the outcome. "Here is the spec we wrote, here is what the AI produced, here is where we had to intervene, and here is the final result." This makes the human contribution visible in a world where AI gets the visible credit for generating code. That visibility matters for morale and for helping leadership understand where human value lives.

### War Games

Half day, quarterly. When AI generates a larger volume of code faster, the surface area for system-level failures grows. Individual specs might each be correct, but their interactions can produce emergent problems that no single spec anticipated. War games stress-test those interactions.

Traditional war games simulate infrastructure failures. In Dandori, a new category is added: what happens when a spec was subtly wrong in a way that passed validation? What if the AI-generated pricing logic has a rounding error that only manifests at scale? What if three independently correct search ranking changes interact to produce terrible results for a specific route type? Participants should include senior engineers from all affected teams, not just the team that owns the system under test.

### Post-Mortem

Event-driven, triggered by incidents. The format survives from established practice but the causal analysis expands. Post-mortems must explicitly ask: was this a spec failure, an AI

execution failure, a validation failure, or a traditional operational failure? Each has different corrective actions. A spec failure means spec patterns need updating. An AI execution failure means guardrails are needed. A validation failure means the review process missed something.

One thing to preserve fiercely: blameless culture. This is even more important when AI is involved because there is a temptation to blame the AI rather than examining the human decisions that led to the failure. The AI is a tool. The spec, the review, and the decision to ship were all human choices.

# 16. What Dandori Transforms

Most teams operate in a hybrid reality: some work is fully AI-executable, some requires heavy human implementation with AI assistance, some is purely human, and some is exploratory. Dandori must work across this spectrum rather than only at the far end. The following Scrum practices are not eliminated. They are transformed to fit a world where AI is a team member but humans remain in the loop.

## Sprints Become Optional and Adaptive

Dandori does not mandate dropping sprints. For teams with a mixed workload of AI-executed specs and human-implemented work, short timeboxes still provide a healthy forcing function that prevents gold-plating specs, endless refinement cycles, and the tendency to keep exploring rather than shipping. What changes is the sprint's purpose and length.

A team that is heavily AI-augmented might run one-week sprints or move to pure continuous flow. A team in hybrid mode keeps sprints but uses them differently: the sprint contains a mix of AI-executed specs and human-implemented work, and the planning conversation acknowledges that these two types of work have fundamentally different flow characteristics. As AI adoption matures, the sprint can loosen or disappear naturally. The methodology supports the full spectrum from strict sprints through loose timeboxes to pure flow, and teams should move along that spectrum based on data, not ideology.

## Sprint Planning Becomes Flow Planning

Sprint planning's question of "can we fit this into the next two weeks?" transforms into a more nuanced capacity check. Flow Planning, which can happen within the Prioritization Reset, asks: how many specs can our team write, execute through AI, and validate in this period, given that some work will require significant human involvement? This is still a capacity question, but it accounts for the reality that the bottleneck is review and validation bandwidth, not implementation bandwidth.

For teams running sprints, Flow Planning aligns with the sprint boundary. For teams on continuous flow, it is a weekly calibration. The key difference from traditional sprint planning is that the team plans around judgment capacity, not labor capacity. How many specifications can we produce at sufficient quality? How much validation bandwidth do we have? Which items need human implementation and how does that affect the overall flow?

## Backlog Grooming Becomes Intent Review

Backlog grooming served a function that the initial version of Dandori undervalued: it was the space where the team collectively built shared understanding of upcoming work. Intent Triage,

conducted by only the Prioritizer, PM, and one senior Spec Owner, is efficient but it means the broader team does not develop context on work before it arrives as a spec to execute or review.

Intent Review is a regular ceremony, recommended biweekly, where the broader team sees and discusses upcoming intents before they enter the Specification stage. This is not estimation. It is context-building. The team asks: do we understand this problem space well enough that any of us could write a spec or review AI output for it? If the answer is no, that signals more preparation is needed before the intent is ready. A Reviewer who participated in an Intent Review about a feature two weeks ago will do a significantly better job reviewing the AI output than one who has never heard of it.

## Story Points Become Specifiability Assessment

Story points measured complexity and uncertainty, not just effort. That signal is still valuable because complexity and uncertainty are exactly the factors that determine whether a spec will succeed or fail on first AI execution. What is broken is the specific implementation: the planning poker ritual, the velocity tracking, and the implicit conversion to time that everyone does despite being told not to.

Dandori replaces story points with a three-tier Specifiability Classification:

| Classification | Definition | Process Implication |
|---|---|---|
| **Clear** | Well-understood problem, can be specified precisely, high confidence in AI execution. | Flows straight to Specification. No Pre-Mortem needed. Likely AI-executed. |
| **Complex** | Requires design decisions, has multiple dependencies, moderate risk of spec ambiguity. | Gets a Pre-Mortem before Execution. May involve human implementation with AI assist. |
| **Uncertain** | Requires research or exploration before specification is possible. High likelihood of spec revision. | Gets an exploration spike before anyone writes a spec. Likely requires significant human work. |

This classification drives process decisions rather than capacity planning. It helps the team calibrate how much preparation different work items need, and it acknowledges that not all work is equally suited to AI execution. Uncertain work may need human implementation with AI assistance. Clear work may be fully AI-executed. The methodology accommodates both.


## Velocity Becomes Flow Metrics

Formal velocity tracking is replaced by a richer set of flow and quality metrics, but the underlying purpose, measuring whether the team is healthy and improving, remains. The specific metrics are detailed in Chapter 17.

### The Scrum Master Role Dissolves into the System Operator

This is the one transformation that is closer to elimination. The coordination function that the Scrum Master performed dissolves into the EM's System Operator role. Pipeline health, ceremony effectiveness, and blocker removal are system-level concerns, not a dedicated role. However, teams in early stages of Dandori adoption may benefit from having someone explicitly own the transition, a function that maps to what a Scrum Master or Agile Coach would do. As the team matures, this function naturally integrates into the EM's responsibilities.

# 17. Metrics and Measurement

Dandori supplements (or, in mature teams, replaces) velocity with a set of flow and quality metrics that measure the health of the spec-driven pipeline. Teams retaining sprints may continue tracking a form of throughput alongside these metrics.

| Metric | What It Measures | Healthy Signal |
|---|---|---|
| **Cycle Time** | Average time from Intent to Integration for a spec | Decreasing or stable over time |
| **First-Pass Success Rate** | Percentage of specs where AI output passes validation without revision | Increasing over time (target: 70%+) |
| **Decision Latency** | Average time between a Decision Request and its resolution | Within SLA (4hr tactical, 24hr strategic) |
| **Stage Dwell Time** | Average time a spec spends in each lifecycle stage | No single stage dominates; balanced flow |
| **Validation Rework Rate** | Percentage of specs that cycle back from Validation to Specification | Decreasing over time (target: under 20%) |
| **Spec Failure Classification** | Distribution of failures by type: spec ambiguity, AI limitation, review miss, operational | Spec ambiguity decreasing as patterns mature |
| **Pipeline WIP** | Number of specs in each stage at any point | No stage exceeding WIP limits |
| **Specifiability Accuracy** | How often the initial Clear/Complex/Uncertain classification matched actual execution experience | Increasing accuracy indicates team is calibrating well |
| **Execution Mode Ratio** | Proportion of specs fully AI-executed vs. human-implemented with AI assist vs. fully human | Tracks AI adoption maturity; no single target, monitored for trend |

These metrics serve the EM's System Operator function. They are reviewed in the Pipeline Sync (daily, for immediate issues) and the Integration Review (weekly, for trends). The metrics should be visible to the entire team on a shared dashboard, not hidden in management reports.

The Specifiability Accuracy and Execution Mode Ratio metrics are unique to Dandori. Together they help the team understand how well they are classifying work and how the balance between AI-executed and human-implemented work is evolving. Teams should not set a target ratio for AI versus human execution. Instead, they should monitor the trend and use it to inform

decisions about where to invest in better spec patterns, which types of work benefit most from AI execution, and where human implementation remains the right choice.

## Translating Metrics into Business Language

Engineering metrics are necessary but not sufficient. Cycle time, first-pass success rate, and pipeline WIP tell the EM whether the system is healthy. They tell the business nothing it cares about. Business stakeholders ask different questions: when will I get this feature? What happens if I change priorities? What is the risk? If the methodology cannot translate its metrics into answers, the business will fill the gap with their own narratives, and those narratives will often be wrong.

### When Will I Get This?

The honest answer comes from cycle time data broken down by specifiability classification. Clear specs typically integrate in 2-3 days. Complex specs take 5-7 days. Uncertain specs take 10-15 days or more. Instead of a sprint commitment that may or may not hold, the business gets a data-driven range. This is less satisfying than "it will be done by Friday" but far more honest. And unlike sprint commitments, the business can check the pipeline at any time and see where their priority stands.

### What Happens If I Change Priorities?

The pipeline makes reprioritization costs visible. When a stakeholder wants to insert a new priority, the Prioritizer can show the current pipeline state: specs in execution (interrupting them has a cost), specs in specification (can be paused with minimal waste), intents in queue (can be reordered freely). The conversation shifts from "can you squeeze this in?" to "here is what moves and here is what it costs."

The Decision Latency metric also surfaces a common but invisible cost. If the business is slow to respond to Decision Requests, the pipeline stalls. Showing stakeholders that three specs have been waiting for their decisions for 48 hours, blocking everything downstream, translates engineering frustration into business language: your features are waiting for your decisions.

### What Is the Risk?

Risk is measurable through the Spec Failure Classification and the Validation Rework Rate. If the rework rate is climbing, the business should know that output quality is at risk and the team is investing more time in corrections than in new features. If spec failures are dominated by ambiguity rather than AI limitations, the risk is upstream: the intents are not clear enough, which means the business and product are not providing sufficient clarity.

This feedback loop distributes accountability. In Scrum, engineering "took the hit" when things were late because the sprint commitment was an engineering promise. In Dandori, the metrics show where the system is constrained. If the bottleneck is decision latency, that is a product or business problem. If the bottleneck is spec quality, that is an engineering problem. If the bottleneck is validation throughput, that is a review capacity problem. The data makes it visible, which makes it actionable, and prevents any one party from controlling the narrative at the expense of the others.

### Who Controls the Narrative?

In many organizations, strategy and product control the narrative about software delivery. Engineering's perspective gets filtered, simplified, or lost. This is a structural consequence of who writes the status reports and who presents to leadership.

Dandori's pipeline visibility addresses this by making the system's state observable to everyone. The pipeline is not an engineering tool that gets summarized for the business. It is a shared artifact that all parties can read directly. When the pipeline shows that five specs are blocked on Decision Requests from product, that is not engineering's narrative. It is the system's state. When the pipeline shows that three Complex specs integrated this week with a 75% first-pass success rate, that is not engineering's spin. It is measured reality.

The Integration Review, which includes the PM and can include stakeholders, is the ceremony where this shared visibility is discussed. It replaces the sprint review's demo format, which was engineering presenting to the business, with a joint inspection of the pipeline, which is everyone looking at the same data and discussing what it means. This does not eliminate organizational politics. No framework can. But it reduces the information asymmetry that enables narrative control, and it gives engineering a factual basis for its perspective that is harder to dismiss than a verbal status update.

# 18. Facilitation: Liberating Structures in Dandori

Dandori's ceremonies need to be short and focused, the framework promises less overhead than Scrum, while also generating genuine insight rather than performative participation. This is a tension that most methodologies ignore: they define what a ceremony should produce but not how the human interaction within it should work.

Liberating Structures, a set of facilitation microstructures developed by Henri Lipmanowicz and Keith McCandless, solve this problem. They are designed to include everyone, surface patterns that traditional meetings miss, and produce results in compressed timeframes. Each structure distributes participation by design, which means the Pipeline Sync surfaces issues from the quiet Reviewer who noticed a pattern, not just from the loudest Spec Owner.

What follows is a mapping of specific Liberating Structures to each Dandori ceremony, with adaptation notes for the spec-driven context.

## 18.1 Pipeline Sync

**1-2-4-All, adapted to the pipeline.** Instead of going around the room reporting individual status, the team spends one minute silently scanning the pipeline board. Then pairs discuss what is stuck or at risk for two minutes. Then the full group surfaces the top one or two systemic issues for the remaining time. This keeps the sync under 15 minutes while ensuring that patterns noticed by quieter team members are not drowned out by the most vocal person.

**Min Specs** is useful as a periodic calibration, not for every day, but once a month during the sync slot. The team asks: what is the minimum set of things that must be true for our pipeline to be healthy? This recalibrates what the team watches for and prevents the daily sync from drifting into rote pattern-matching.

## 18.2 Spec Handoff

**What I Need From You (WINFY), adapted for two people.** The Spec Owner presents the spec and explicitly states what they need the Reviewer to pay attention to. The Reviewer responds with what they need clarified to do an effective review. This structured exchange prevents the handoff from becoming either a rubber stamp or an unfocused walkthrough. The explicit "here is what I need from you" framing creates accountability on both sides.

## 18.3 Decision Request

This ceremony is async, so traditional Liberating Structures do not apply directly. But the underlying principle of **Helping Heuristics** shapes the format. The Decision Request template should force the requester to specify: am I asking you to decide (just give me the answer), advise (tell me what you think and I will decide), or clarify (help me understand the constraints so I can decide)? This prevents the common failure where the Spec Owner wants a decision, the PM thinks they are being asked for advice, and both leave thinking the other handled it.

## 18.4 Pre-Mortem

**TRIZ** is almost perfectly designed for pre-mortems. The first prompt asks: "How could we guarantee that this spec produces a catastrophic failure?" The team generates the worst possible outcomes, which is psychologically easier and more creative than asking what might go wrong. Then the second prompt inverts: "Which of these failure modes are we actually at risk of, given our current spec?" This surfaces risks that polite, constructive thinking misses because it gives people permission to think destructively before switching to constructive analysis.

**15% Solutions** works as a follow-up when the pre-mortem reveals risks that feel overwhelming. After identifying what could go wrong, each person identifies what they can do immediately, within their own authority, to reduce the highest-priority risk. This prevents the pre-mortem from becoming a list of fears that nobody acts on.

## 18.5 Integration Review

**What, So What, Now What (W3)** maps directly to the Integration Review's purpose. "What" is what shipped this week and what the AI produced. "So What" is what patterns the team sees in spec success, failure, and rework. "Now What" is what the team changes about specs, reviews, or priorities going forward. The three-stage structure prevents the review from becoming either a pure celebration of output, ignoring quality signals, or a pure critique session, ignoring what went well.

For the architectural coherence check within this ceremony, **Critical Uncertainties** helps the team identify which of the week's changes interact with each other and where emergent risk might live. The team maps changes on two axes: confidence in the change and degree of interaction with other changes. The combination of high interaction and low confidence gets immediate attention.

## 18.6 Spec Retrospective

**Troika Consulting** is powerful here. Each Spec Owner takes two minutes to describe a spec that failed or required significant rework. Two colleagues act as consultants, asking questions and offering observations, while the Spec Owner listens silently. Then rotate. This structure prevents the retro from becoming a blame session because the format explicitly separates the

person asking for help from the people providing it. It also generates more actionable advice than group discussion because the consultants are focused on one specific case rather than debating in the abstract.

For pattern identification across multiple spec failures, **Ecocycle Planning** helps the team see which spec patterns are mature and working, which are emerging and need investment, which are rigidly stuck, and which should be retired. This prevents the retro from treating every issue as equally important and helps the team focus improvement effort where it will have the most impact.

## 18.7 Prioritization Reset with Flow Planning

**25/10 Crowd Sourcing, adapted for a small team.** Each person writes their top candidate for "most valuable thing to specify next" on a card with a brief rationale. Cards circulate, each person scores them 1-5, and the highest-scored intents rise to the top. This prevents the prioritization from being dominated by whoever argues most forcefully, which is the failure mode of most priority-setting conversations.

For the Flow Planning capacity check, **Impromptu Networking** adapted to pairs works well. Pairs spend three minutes each discussing: what is my biggest constraint this week? What could the team do differently to help me move specs through faster? Two rounds with different partners, then the group surfaces the top constraints. This is faster and more honest than asking each person to report their capacity in front of the group.

## 18.8 Intent Review

**Discovery and Action Dialogue (DAD)** fits the intent review's purpose precisely. For each intent being reviewed, the team works through: "How do we know this is a real problem?" to ground the discussion in evidence, "What have we already tried or discussed?" to avoid reinvention, "What are the barriers to specifying this well?" to surface unknowns, and "Who else has useful knowledge about this?" to identify missing context. This structured dialogue prevents the intent review from becoming either a superficial walkthrough or an unbounded design discussion.

**Shift and Share** works when the team is reviewing multiple intents in a single session. Split the intents across stations, each hosted by someone with context. Team members rotate through stations, spending 10 minutes at each. This covers more ground than sequential review and gives everyone exposure to upcoming work without requiring everyone to sit through every discussion.

## 18.9 Team Retrospective

This is the ceremony where Liberating Structures have the most latitude because it is the most human-centered. **Conversation Cafe** works well for the identity and emotional processing that the monthly retro needs to address. The structured rounds, with a talking object and explicit invitation for each person to speak, create safety for engineers to express how the shift to spec-driven work is affecting their professional identity and satisfaction. The "no cross-talk during rounds" rule prevents the retro from being dominated by the most vocal people.

For deeper exploration when a specific tension has surfaced, **Heard, Seen, Respected (HSR)** in pairs creates space for people to process experiences that are difficult to discuss in a group. Each person takes five minutes to tell a story about a moment when they felt challenged,

frustrated, or energized by the new way of working. The listener's only job is to listen, not to solve or advise. This is particularly valuable during the transition period when people are processing real changes to their professional identity.

## 18.10 Demo Day

**Appreciative Interviews** can open the Demo Day. Before the demos begin, pairs spend three minutes each interviewing each other: "What is something you are proud of from the work we are about to see?" This primes the room for genuine appreciation rather than performative approval and surfaces the human story behind each demo.

**Celebrity Interview** works for the demo format itself. Instead of the traditional "here is what we built" presentation, one person presents and another plays the interviewer, asking questions the audience would want to know: "What was the hardest part of the spec? Where did the AI surprise you? What would you do differently?" This format is more engaging than a walkthrough and naturally highlights the spec-to-outcome journey that Demo Day in Dandori is meant to showcase.

## 18.11 War Games

**Wicked Questions** is essential preparation for war games. Before running failure scenarios, the team identifies the paradoxes in their system: "How is it that our specs are individually correct but the system as a whole is becoming more fragile?" or "How is it that we ship faster than ever but have less confidence in what we have shipped?" These questions reveal the systemic tensions that war game scenarios should stress-test.

During the war game debrief, **Purpose-to-Practice (P2P)** provides structure. For each failure scenario, the team works through: what is our purpose (why does this system exist), what principles guide our response, what participants are involved, what structure supports the response, and what practices make the response reliable? This prevents the debrief from being a vague "we should be more careful" and turns it into actionable improvements to both the system and the spec patterns.

## 18.12 Post-Mortem

**Nine Whys, adapted for spec-driven development.** Instead of asking "why did the system fail?" five times, ask "why did the spec not prevent this?" nine times. This forces the team deeper into the specification and review process that produced the failure, rather than stopping at surface-level causes like "the AI generated bad code" or "we did not test enough." The deeper questioning reveals the spec patterns, decision points, and review practices that need to change.

**User Experience Fishbowl** works for post-mortems where multiple teams or roles were involved. A small group, the people closest to the incident, sits in the center and discusses what happened while the outer group listens. Then the groups swap. This structure prevents the post-mortem from becoming a cross-team blame session and gives each group space to tell their version of the story before hearing the other's perspective. In Dandori, this is especially important because a spec failure might involve the Spec Owner, the Reviewer, the PM who wrote the intent, and the AI execution layer, each with a different view of what went wrong.

# 19. Prerequisites: Understanding Spec-Driven Development

Dandori is built on the foundation of spec-driven development. Before adopting the framework, everyone who will work within it or be affected by it, engineers, product managers, engineering managers, and stakeholders, needs to understand the basic principles of spec-driven development. Without this shared understanding, teams will drift into the same problem that plagued Scrum: everyone interpreting the framework differently, reinventing practices that already have clear definitions, and losing the coherence that makes the system work.

Spec-driven development is the discipline of writing structured, precise specifications before implementation begins, and treating those specifications as the primary artifact around which all coordination happens. It is not a new idea. It draws from formal methods, design-by-contract, and the structured engineering practices that predated agile. What makes it newly relevant is that AI execution engines can now implement against well-written specifications with high fidelity, making the specification the most leveraged artifact in the entire development process.

The minimum understanding every team member needs includes: what a specification is and how it differs from a user story or a ticket, why precision in specification matters more when AI is executing than when humans are implementing, how the five lifecycle stages work and what each stage produces, and what their role is within the methodology. This is not a heavy training burden. It is a shared vocabulary and a shared mental model that prevents the methodology from fragmenting into incompatible interpretations.

Teams that skip this step and go straight to adopting Dandori's ceremonies will find themselves performing the rituals without understanding the reasoning. That is exactly how Scrum lost its meaning, and Dandori should not repeat the pattern.

# 20. Conclusion

Dandori is not a rejection of what came before. It is an evolution driven by a fundamental change in the economics of software development. When AI handles a growing share of implementation, the activities that create value shift toward preparation: precise specification, fast decision-making, rigorous validation, and architectural judgment. But the transition is a spectrum, not a switch. Teams operate in a hybrid reality where some work is fully AI-executed, some is human-implemented with AI assistance, and some remains purely human. Dandori is designed for that spectrum.

The framework transforms Scrum's practices rather than discarding them. Sprints become adaptive. Estimation becomes specifiability assessment. Backlog grooming becomes intent review. The core discipline of planning, reflecting, and improving survives. What changes is the object of that discipline: from optimizing human labor to optimizing human judgment.

The framework provides the structure for this new reality without burdening teams with ceremony that does not earn its place. It respects the Agile values that Scrum championed while updating the practices for a world where the most important work happens before and after AI executes, not during execution.

The name itself is the thesis. Dandori. Preparation is the work.

> *In the Toyota Production System, they discovered that the quality of preparation determines the quality of production. In AI-augmented software development, we are learning the same lesson: the quality of the specification determines the quality of the code. Dandori is the methodology that takes this lesson seriously.*