From AI-Assisted to AI-Operated

The Seven Levels of Software Delivery Autonomy

ProductBuildersHQ

May 2026

Most organizations using AI for software development are stuck at the same place: AI writes code, humans review it. This feels like progress. Developers report productivity gains. Commit velocity increases. But the fundamental process remains unchanged—humans orchestrate every step, validate every output, and gate every deployment.

The real transformation begins when AI stops being a faster typewriter and starts changing how software delivery operates.

This article introduces the Software Delivery Autonomy Levels—a seven-level progression from traditional Agile teams to fully autonomous software delivery and operations. We focus specifically on Levels 4 through 7, where the process itself transforms, examining real-world implementations at AWS, Spotify, StrongDM, and the emerging frontier beyond them.

At Levels 6 and 7, we introduce ASDM (Autonomous Software Delivery Model)—a methodology with specific practices and artifacts for achieving autonomous coding, review, validation, and operations. Teams can adopt ASDM practices incrementally as they progress through the levels.

The shift from "AI-assisted" to "AI-operated" is not incremental improvement. It is a fundamental change in how software gets built, validated, and shipped.

Software Delivery Autonomy Levels: The Model

The seven levels describe a progression in who—or what—executes software delivery work, and how humans participate in the process.

Level	Name	Human Role	Defining Characteristic
1	Agile/Scrum	Execute work	Team coordination optimization
2	DevOps	Build + operate	Pipeline automation (CI/CD)
3	AI-Assisted	Primary implementer	AI augments existing process
4	AI-Native Workflows	Orchestrator	Process redesigned around AI
5	Agentic Engineering	Batch reviewer	Extended autonomous sessions
6	Autonomous Coding & Review	Specification owner	No human coding or code review

Level	Name	Human Role	Defining Characteristic
7	Autonomous Operations	Governor	Autonomous production operations, exceptions only

A useful lens is the loop position of human involvement:

- Levels 1-3: Human-in-the-loop (interactive, every step)
- Level 4: Human-in-the-loop (but process is AI-native)
- Level 5: Human-on-the-loop (batch review after autonomous sessions)
- Level 6: Human-over-the-code-loop (humans define intent and scenarios, not code)
- Level 7: Human-over-the-operations-loop (governance, incidents, and exceptions only)

Why We Focus on Levels 4-7

Levels 1 and 2 represent pre-AI organizational maturity. Agile optimized team coordination. DevOps automated delivery pipelines. These are established baselines that most software organizations have adopted in some form.

Level 3—AI-Assisted development—is where most organizations operate today. Developers use Copilot, Cursor, or similar tools to accelerate coding. AI generates boilerplate, suggests completions, and helps with documentation. Productivity improves, but the workflow remains fundamentally unchanged: humans drive, AI assists.

The inflection point occurs between Level 3 and Level 4. This is where AI stops augmenting existing processes and starts requiring new ones. The tooling changes. The team dynamics change. The validation systems change. Everything downstream of "how we build software" must be reconsidered.

Levels 4 through 7 represent this transformation—from AI as tool to AI as operator.

Validation Methods by Level

As autonomy increases, validation methods must evolve. Human judgment gives way to automated systems capable of operating without human attention.

Level	Primary Validation	Automated Validation	Human Gate
3	Human code review	Unit tests, CI checks	Every PR
4	Human code review	Unit tests, CI checks, prompt benchmarks	Every PR
5	Human batch review	Unit tests, CI checks, LLM-as-a-Judge for style/security	Before merge
6	Scenario satisfaction	Scenarios, DTU, LLM-as-a-Judge, probabilistic metrics	Before production
7	Operational satisfaction	Production telemetry, SLOs, canaries, rollback/remediation agents	Exceptions only

LLM-as-a-Judge emerges at Level 5 as a bridge technology. AI models evaluate code for style consistency, security vulnerabilities, and architectural compliance—tasks previously requiring human reviewers. At Level 6, LLM-as-a-Judge becomes one component of a broader autonomous validation stack. At Level 7, validation expands beyond code correctness into operational health: deployment safety, live telemetry, incident response, rollback, and continuous policy compliance.

Artifacts and Rituals by Level

Each level introduces new artifacts (documents, infrastructure) and rituals (practices, ceremonies) that enable increased autonomy.

Level	New Artifacts	New Rituals
4	Steering files (AGENTS.md), prompt libraries, context specs	Mob Elaboration, Mob Construction, prompt reviews
5	Overnight execution logs, batch review queues, agent traces	Queue-before-sleep, morning review sessions, high-bandwidth standups
6	Scenario libraries, Digital Twins, satisfaction dashboards, provenance records	Scenario authoring, DTU maintenance, policy governance, convergence monitoring
7	Operational policy engines, production guardrails, automated remediation playbooks	SLO governance, incident simulation, exception review, autonomous operations review

These artifacts and rituals accumulate—Level 7 organizations maintain everything from earlier levels while adding new capabilities.

Level Operating Profiles

The levels are easiest to distinguish by looking at what changes in behavior, output, metrics, and limitation:

Level	Behaviors	Primary Output	Useful Metrics	Key Shift	Limitation
1 — Agile/Scrum	Humans coordinate work through ceremonies, tickets, and planning rituals	Shipped increments from human teams	Sprint predictability, throughput, cycle time	Work becomes coordinated	Delivery remains human-executed
2 — DevOps	Teams automate build, test, release, and operations workflows	Reliable delivery pipeline	Deployment frequency, lead time, MTTR, change failure rate	Delivery becomes automated	Implementation remains human-executed
3 — AI-Assisted	Developers use copilots and chat tools inside existing workflows	Faster human-authored code	AI-assisted commits, time saved, PR throughput	AI accelerates individuals	Process and review model remain unchanged

Level	Behaviors	Primary Output	Useful Metrics	Key Shift	Limitation
4 — AI-Native Work-flows	Teams redesign planning, construction, context, and feedback around AI participation	AI-native delivery workflow	Prompt reuse, context quality, prompt benchmark pass rates, PR quality	AI becomes part of the process	Humans still orchestrate every significant step
5 — Agentic Engi-neering	Agents execute for hours; humans review completed batches	Agent-generated PRs or features	Autonomous session duration, batch review volume, generated PRs, review rejection rate	Humans move from interactive driver to batch reviewer	Human review remains the bottleneck
6 — Au-tonomous Coding & Review	Agents write and validate code without human code review	Scenario-satisfied software candidate	Human-written code %, human code-review %, scenario volume, satisfaction rate, token spend	Validation replaces code review	Production operations still retain human gates
7 — Au-tonomous Opera-tions	Systems deploy, monitor, remediate, and escalate under policy	Operating production system	Autonomous deployment %, roll-back/remediation success, SLO adherence, exception rate	Policy-governed systems operate software	Requires exceptional trust, observability, and compliance controls

Level 4: AI-Native Workflows

Defining characteristic: The development process is redesigned around AI participation across the entire lifecycle.

Human role: Orchestrator—humans direct AI across planning, implementation, testing, and deployment rather than executing each step.

Example: AWS AI-Driven Development Lifecycle (AI-DLC)

The AWS AI-DLC Methodology

In early 2026, AWS published their AI-Driven Development Lifecycle (AI-DLC) framework, arguing that existing Agile and Scrum practices are fundamentally misaligned with AI-native development. Their core critique:

> "Simply retrofitting AI as an assistant… reinforces outdated inefficiencies."

AI-DLC proposes replacing traditional sprint ceremonies with AI-native workflows:

Mob Elaboration replaces sprint planning. Instead of humans decomposing work into tickets, AI participates in requirements analysis, generates implementation plans, and identifies technical risks. Humans guide and validate rather than author.

Mob Construction replaces pair programming. Multiple AI agents can work on related components simultaneously while humans orchestrate priorities and resolve conflicts.

Bolts instead of Sprints. The two-week sprint cadence, designed around human cognitive limits and coordination overhead, becomes less relevant when AI can iterate continuously. Work flows in smaller, faster increments.

What Changes at Level 4

The shift from Level 3 to Level 4 requires rethinking several assumptions:

Steering files become infrastructure. Documents like AGENTS.md, CLAUDE.md, and COPILOT.md emerge as machine-readable governance. These encode repository conventions, architecture constraints, dependency rules, and operational safety requirements. In Level 3, these lived as tribal knowledge or scattered documentation. In Level 4, they become executable contracts that AI systems consume directly.

Context management becomes critical. AI systems require explicit context that humans infer implicitly. Level 4 organizations invest in persistent memory systems, architectural decision records, and structured specifications that AI can consume across sessions.

Prompt quality becomes operationally important. Bad prompts at Level 3 waste time. Bad prompts at Level 4 can create insecure code, hallucinated refactors, or architectural drift at scale. Organizations begin treating prompts as versioned assets—reviewed, tested, and benchmarked.

Feedback loops tighten. AI-DLC emphasizes adaptive workflows where production telemetry directly influences development priorities. The boundary between "building" and "operating" software blurs further than DevOps achieved.

The Skill of Calibrated Prompting

One of the underappreciated requirements of Level 4 is developing intuition for AI interaction. Joe Magherimov, who led AWS's Project Mantle, describes this as finding the "maximally supportable request":

> "One of the intuitions you build up... is that you build up that intuition of what is that maximally supportable [request]. Ask for too much and the model is likely to fail—it might run out of the context window or do weird stuff. Ask for too little and you're not getting quite the speed-ups because you still have to be constantly in the loop."

This calibration is not static. It varies by model, project, and domain. The only way to develop this intuition is practice—which means Level 4 requires investment in team skill development, not just tooling.

Ambiguity is another enemy of productivity. When multiple approaches could solve a problem, effective Level 4 practitioners brainstorm with the model before implementation. Magherimov describes starting with "here's the problem I have, list solutions" to eliminate dead ends before committing to an approach.

Context Window Discipline

Level 4 also introduces new technical practices around context management. Magherimov describes clearing the context window between every request, using files as persistent memory for multi-step work:

> "I would start with a file that describes a high-level approach. Then we break it down. I would start a prompt with: okay, here's our design we agreed on, we're now on step two out of seven, and here's what we're going to do in this step."

When a step completes, the code is committed, the context is cleared, and the next session begins fresh with the durable spec file reloaded. This mirrors how humans naturally decompose large problems into manageable pieces—but makes the practice explicit and systematic.

The Level 4 Bottleneck

Despite AI participation across the lifecycle, Level 4 maintains human orchestration at each step. The developer's job changes—from implementer to orchestrator—but humans remain in the loop for every significant decision.

This is intentional. AWS explicitly frames AI-DLC as "AI-powered execution with human oversight." The methodology assumes humans validate AI outputs before they progress.

For many organizations, Level 4 represents a sustainable equilibrium. AI dramatically accelerates work while humans maintain quality control. But the constraint is clear: every output waits for human attention.

Level 5 breaks this constraint.

Level 5: Agentic Engineering

Defining characteristic: AI agents execute autonomously for extended periods (hours, not minutes). Humans review outputs in batches rather than interactively.

Human role: Batch reviewer—humans validate completed work rather than guiding each step.

Examples: AWS Project Mantle, Spotify Honk

AWS Project Mantle: Overnight Autonomous Coding

Project Mantle offers one of the most detailed public accounts of Level 5 development. In January 2026, Joe Magherimov—VP and Distinguished Engineer at AWS—described how a team of approximately nine senior engineers built a new inference engine for Amazon Bedrock using AI-first development, claiming 10x commit velocity.

The team composition matters: one Distinguished Engineer plus eight Principal Engineers, all from EC2 with decades of collective experience. This was not junior developers experimenting with AI. This was senior engineers deliberately restructuring how they work.

The overnight pattern. Magherimov describes queuing prompts before sleep and checking results in the morning:

> "The thing that probably took me most by surprise is that asynchronous nature of software development. All of a sudden, I don't actually need to be present or even paying attention for 80% to 90% of the cycle of making a software change."

This represents the defining shift of Level 5: extended autonomous sessions. AI agents execute for eight or more hours without human guidance. The human checkpoint comes after the autonomous session completes, not during it.

Human accountability remains. Critically, Mantle maintained human responsibility for all code:

> "At the end of the day, any line of code committed into the repository has a human name attached to it. A human is ultimately responsible for the quality of the code."

This distinguishes Level 5 from Level 6. Agents execute autonomously, but humans still review everything before merge. The workflow is "fire, then review" rather than "review each step."

The magnification effect. Magherimov's colleague offered a key insight:

> "AI-assisted development is gonna magnify well-functioning teams much more than it magnifies not-so-well-functioning teams."

Teams with strong fundamentals—clear communication, fast feedback loops, solid testing—see those advantages amplified. Teams with dysfunction see that amplified too. Level 5 does not fix broken processes; it accelerates whatever exists.

Spotify Honk: Background Agents at Scale

Spotify's Honk system provides another Level 5 implementation, optimized for fleet-wide changes across their large codebase.

Background execution. Rather than interactive coding sessions, Honk runs agents in the background. Engineers specify transformations in natural language, and agents execute autonomously—running formatting, linting, generating diffs, and capturing traces.

Ready-to-review outputs. Agents generate pull requests automatically opened against target repositories. The surrounding infrastructure handles "targeting repositories, opening pull requests, getting reviews, and merging into production." Humans review before merge, but the creation is fully autonomous.

Scale metrics. Spotify reports over 1,500 merged AI-generated pull requests across migrations, with 60-90% time savings versus manual implementation. Hundreds of developers interact with the system for tasks ranging from language modernization to framework upgrades to UI component migrations.

The headline claim. In February 2026, TechCrunch reported that Spotify's "best developers haven't written a line of code since December, thanks to AI." These engineers shifted entirely to specification, review, and governance—the natural endpoint of Level 5.

The Level 5 Operating Model

Level 5 organizations exhibit several common patterns:

Batch review replaces interactive review. Instead of reviewing each AI response, humans review completed PRs or feature implementations. This dramatically increases throughput but requires higher trust in AI outputs.

Testing investment increases. Both Mantle and Honk teams report significant investment in testing infrastructure. When code velocity increases 10x, bug volume increases proportionally unless validation systems scale.

Local integration testing becomes critical. Mantle emphasized running the full stack locally with fast feedback loops—minutes, not the 20-30 minutes typical of CI pipelines. This enables rapid validation of AI-generated code before it affects other engineers.

Co-location or high-bandwidth communication. Mantle co-located the entire team for high-frequency whiteboard discussions. When code changes happen overnight, shared understanding becomes the bottleneck. Miscommunication compounds faster.

The Quality Challenge at Velocity

Level 5 velocity creates new quality challenges. Magherimov observed:

> "Even if the rate of bugs was lower than what would have been with a human, they still happen. And when you write a lot more code, you're gonna have a lot more bugs."

More critically, bugs at this velocity impact the entire team. A buggy check-in can break other engineers' workflows, creating "tools-down" scenarios where everyone chases down what changed. The team's response was to raise the bar on testing and verification—catching bugs before they're checked in, before they impact other engineers.

This is why Mantle allocated approximately 25% of team effort to improving development practices rather than feature work. They worked with build system teams to speed up feedback loops—minutes instead of twenty to thirty minutes for builds and integration tests.

As Magherimov acknowledges, none of this is new—the industry has been paying attention to testing and verification for years. But at these rates of change, these practices become even more critical. Without a way to curtail chaos, it explodes.

The Level 5 Constraint

Level 5 removes human involvement from execution but maintains it for validation. Every PR still requires human review before merge. Every deployment still gates on human approval.

This constraint exists for good reason: AI-generated code can be subtly wrong in ways that compile and pass tests but violate business logic or security requirements. Human review catches these failures.

But this constraint also creates a bottleneck. As agent throughput increases, human review capacity becomes the limiting factor. Organizations either hire more reviewers, accept slower throughput, or consider Level 6.

Level 6: Autonomous Coding & Review

Defining characteristic: Humans no longer write or review code. Validation shifts to scenario-based testing and probabilistic satisfaction metrics, but production operations still retain human gates or conventional operational controls.

Human role: Specification owner—humans define intent, scenarios, constraints, and acceptance thresholds. They do not write or review code.

Example: StrongDM Software Factory

The StrongDM Software Factory

StrongDM's Software Factory represents the most aggressive public implementation of autonomous coding and review. Their operating principles are explicit:

> "Code must not be written by humans." "Code must not be reviewed by humans."

This is not "humans review less" or "humans review only high-risk changes." This is a categorical exclusion of human coding and code review from the development process.

That is why StrongDM fits Level 6 rather than Level 7 in this model. The public material demonstrates non-interactive development: specifications and scenarios drive agents that write code, run harnesses, and converge without human review. It also describes Digital Twin clones of services such as Okta, Jira, Slack, Google Docs, Google Drive, and Google Sheets for high-volume validation. It does not, however, establish that StrongDM has delegated production operations for its enterprise offerings to autonomous systems

without human operational gates. Level 6 is therefore the right classification: autonomous coding and review, not yet fully autonomous operations.

How does software get validated without human review?

Scenarios replace tests. Traditional unit tests are written by humans (or AI) alongside the code. This creates a problem: AI might game narrow test conditions with shortcuts like hardcoded return values.

StrongDM uses "scenarios"—end-to-end user stories stored outside the codebase. These function as hold-out sets that the code-generating agents cannot see during implementation. The scenarios describe what users need to accomplish, not implementation details.

Probabilistic satisfaction replaces binary pass/fail. Instead of "did all tests pass?" the factory measures: "Of all the observed trajectories through all the scenarios, what fraction of them likely satisfy the user?"

This probabilistic framing acknowledges that software correctness is not binary. A system can satisfy 99% of user journeys while failing edge cases. Level 6 makes this tradeoff explicit and measurable before humans accept the system for operational use.

Digital Twin Universe (DTU) enables unlimited testing. Software rarely operates in isolation. Modern applications integrate with Okta, Jira, Slack, Google Workspace, and dozens of other services. Testing against production APIs has limits: rate limiting, API costs, inability to test failure modes.

StrongDM's DTU creates behavioral clones of third-party services. This enables testing at volumes exceeding production limits, safe failure mode testing, and running thousands of scenarios per hour without rate limiting or API costs. The DTU makes validation possible that would be impossible against real services.

The DTU is perhaps the most significant infrastructure investment required for Level 6. Without it, scenario-based validation cannot achieve the coverage necessary to replace human code review.

The Convergence Model

StrongDM describes their factory as a system where "specs + scenarios drive agents that write code, run harnesses, and converge without human review." The key word is converge.

Unlike traditional development where code is written once and reviewed, the factory operates iteratively. Agents generate code, run it against scenarios, observe failures, and generate improved versions. This cycle repeats until the satisfaction metrics reach acceptable thresholds.

This convergence model is only possible because of recent advances in LLM reliability. StrongDM credits Claude 3.5 (October 2024) with achieving "compounding correctness rather than error" in long-horizon coding workflows. Earlier models would accumulate errors over extended sessions, making convergence impossible. Recent models can maintain coherence across thousands of iterations.

The combination of Cursor's YOLO mode (continuous agent iteration without human confirmation) and reliable long-horizon models created the technical foundation for Level 6.

The Economics of Level 6

StrongDM suggests a provocative cost baseline:

> "If you haven't spent at least $1,000 on tokens today per human engineer, your software factory has room for improvement."

At $1,000/day per engineer, a 10-person team spends $200,000/month on tokens alone—before infrastructure, tooling, or salaries. This is dramatically higher than Level 3-5 token costs.

Simon Willison, in his analysis of the Software Factory, raises the obvious concern:

> "If these patterns really do add $20,000/month per engineer to your budget they're far less interesting to me."

The economics only work if the productivity gains exceed the costs. StrongDM's implicit claim is that autonomous coding and review delivers enough value to justify the investment. But this remains unproven at scale across industries.

What Level 6 Requires

Level 6 is not achievable by simply removing human review from Level 5. It requires fundamentally different validation infrastructure:

Scenario libraries. Comprehensive end-to-end user stories that define what "working software" means. These must be maintained separately from the codebase and cover edge cases that would traditionally rely on human judgment.

Digital Twin infrastructure. Behavioral clones of all external services the software interacts with. This is a significant engineering investment, potentially as complex as the software itself.

Satisfaction metrics. Systems for measuring probabilistic user satisfaction across scenario trajectories. This requires instrumentation, measurement frameworks, and thresholds that define "good enough."

Governance policies. Clear rules for what requires human escalation before production exposure. Even Level 6 systems need escape hatches for truly novel situations, security incidents, or regulatory requirements.

Provenance and auditability. When no human reviews code, you need perfect records of which agent made which change, which prompt caused it, which model version was used, and why. Compliance and debugging both depend on this.

Level 7: Autonomous Operations

Defining characteristic: Autonomous systems not only generate and validate code, but also operate software in production: deployment, monitoring, rollback, remediation, incident response, and policy enforcement run without routine human gates.

Human role: Governor—humans define business objectives, risk policies, escalation thresholds, and regulatory constraints. Humans handle exceptions rather than normal delivery or operations.

Example: Emerging frontier; no broadly documented enterprise example yet.

Level 7 is the true endpoint of software delivery autonomy. It is what people often mean when they use the "dark factory" metaphor: a production system where software moves from intent to implementation to operation without humans in the normal execution path. We avoid that term here because it carries unnecessary negative connotations and centers the absence of humans rather than the governance model that makes autonomy acceptable.

The important distinction is operational responsibility. Level 6 can produce code without human coding or code review, but humans may still decide when it is exposed to customers, monitor production behavior, approve remediations, and own incident response. Level 7 moves those operational gates into policy-governed autonomous systems.

What Level 7 Adds Beyond Level 6

Autonomous deployment. Changes that satisfy scenario and policy thresholds can progress through environments, canaries, and rollout stages without human approval.

Production telemetry as validation. Scenario satisfaction is necessary but not sufficient. Level 7 systems continuously compare live behavior against SLOs, user outcomes, security signals, cost budgets, and policy constraints.

Autonomous rollback and remediation. When telemetry degrades, systems can pause rollout, revert, generate a fix, validate it, and redeploy without waiting for an operator.

Incident simulation and preparedness. Digital Twins expand from third-party service validation into operational rehearsal: outage scenarios, degraded dependencies, permission failures, regional failures, abuse patterns, and cost spikes.

Policy-governed escalation. Humans are paged for exceptions, not routine decisions. Escalation policies define what the system may do alone, what requires notification, and what requires human approval.

Why Level 7 Is Rare

Autonomous operations requires trust across a wider surface than autonomous coding. Code correctness is only one part of production safety. Level 7 also needs observability maturity, deployment isolation, rollback confidence, incident playbooks, security policy automation, compliance evidence, cost controls, and organizational willingness to let systems act.

This is why StrongDM's Software Factory is an important Level 6 signal without being sufficient evidence of Level 7. It shows that code creation and review can move out of the human loop. The next frontier is whether production operation can do the same.

Introducing ASDM: The Methodology for Levels 6 and 7

ASDM (Autonomous Software Delivery Model) is the methodology for achieving and operating at Levels 6 and 7. Just as Scrum provides artifacts and practices for Agile teams, and DevOps provides practices for automated delivery, ASDM provides the framework for autonomous coding, review, validation, and operations.

Core ASDM Practices

Specification-first development. Software is defined by machine-readable specifications that agents consume directly. Humans author intent; agents implement.

Scenario-based validation. End-to-end user stories replace unit tests as the primary validation mechanism. Scenarios are holdout sets that implementation agents cannot access.

Probabilistic satisfaction metrics. Success is measured by the fraction of user journeys satisfied, not binary test passage.

Digital Twin testing. External services are cloned for unlimited, safe validation. Production APIs are never the bottleneck.

Zero human code review. Code is not read by humans as a required approval step. Validation systems replace human code review.

Governance-by-policy. Human oversight operates through policy definition, not code review. Policies define escalation triggers, risk thresholds, prohibited patterns, and operational authority.

Full provenance. Every change is traceable to its originating specification, prompt, agent, and model version.

Autonomous operations. At Level 7, deployment, monitoring, rollback, remediation, and incident response are governed by policy and telemetry rather than routine human approval.

ASDM Practices at Earlier Levels

Not all ASDM practices require Level 6. Teams can adopt components incrementally:

Practice	Level 4	Level 5	Level 6	Level 7
Steering files (AGENTS.md)	Introduce	Mature	Required	Required
Extended autonomous sessions	—	Core practice	Core practice	Core practice
Scenario-based validation	Experiment	Partial adoption	Required	Required
Digital Twin infrastructure	—	Experiment	Required	Required
Probabilistic satisfaction	—	—	Required	Required
Zero human code review	—	—	Required	Required
Governance-by-policy	Introduce	Develop	Required	Required
Full provenance	Good practice	Important	Required	Required
Autonomous operations	—	—	Experiment	Required

This incremental adoption path allows teams to build toward Levels 6 and 7 without committing to the full transformation immediately.

The ASDM Mindset Shift

Adopting ASDM requires more than new tools and processes. It requires a fundamental shift in how teams think about software quality and accountability.

From “Did a human verify this?” to “Do the scenarios and policies pass?” Traditional quality assurance relies on human judgment. Someone reads the code, considers edge cases, and attests that it meets requirements. ASDM replaces this with systematic scenario coverage and explicit policy gates. At Level 6, this determines whether code can be accepted without human review. At Level 7, it also determines whether software can operate, remediate, and evolve in production without routine human approval.

From implementation ownership to specification ownership. Engineers stop owning code and start owning specifications and scenarios. The quality of your work is measured by how well you defined what the software should do, not how elegantly you implemented it.

From debugging code to debugging scenarios. When production fails, the first question is not “what’s wrong with the code?” but “what scenario did we miss?” Failures indicate gaps in scenario coverage, not gaps in human review.

This mindset shift is perhaps the hardest part of Level 6 and Level 7 adoption. Many engineers derive professional identity from code craftsmanship. ASDM asks them to derive identity from specification, scenario, and operational policy craftsmanship instead.

The Critical Transitions

Moving between levels is not continuous improvement—each transition requires specific changes that can feel like regression before the benefits materialize.

Level 3 → Level 4: Accepting Process Change

The hardest part of moving from AI-Assisted to AI-Native is accepting that existing processes need replacement, not augmentation. Teams comfortable with Agile ceremonies resist replacing sprint planning with Mob Elaboration. Engineers skilled at code review resist restructuring around AI-generated code.

The transition requires leadership commitment to process experimentation and tolerance for temporary productivity loss while new patterns establish.

Level 4 → Level 5: Trusting Asynchronous Execution

Moving from interactive orchestration to overnight autonomous sessions requires trust that wasn't necessary before. Engineers must accept that code will be generated while they sleep, without the ability to course-correct in real-time.

This transition also requires infrastructure for extended agent execution—checkpointing, failure recovery, and reliable long-running processes that most organizations haven't built.

Level 5 → Level 6: Eliminating Human Code Review

Removing human code review feels like removing a safety net—because it is. Teams must build confidence that scenario-based validation catches what human review would catch, and accepts the cases where it doesn't.

This transition is not for every organization. Many will find Level 5 the appropriate equilibrium—significant productivity gains with maintained human oversight. Level 6 is for organizations where the economics and risk tolerance align with fully autonomous coding and review.

Level 6 → Level 7: Removing Routine Operational Gates

Level 7 is a different kind of trust boundary. A team may accept no human code review while still requiring human approval for deployment, rollout, rollback, customer exposure, and incident response. Moving to Level 7 requires confidence that operational policies, telemetry, SLOs, canaries, and remediation systems can make routine production decisions.

This transition is likely to lag Level 6 adoption. The blast radius is larger, the compliance questions are harder, and the failure modes are more visible to customers.

What Each Level Requires

The infrastructure and organizational requirements escalate significantly across Levels 4-7. Each level builds on the previous—Level 7 organizations need everything from Levels 4, 5, and 6, plus additional operational capabilities.

Requirement Category	Level 4	Level 5	Level 6	Level 7
Tooling	AI-native IDE, prompt management	Extended execution infrastructure	Convergence orchestration	Autonomous deployment and remediation
Documentation	Steering files (AGENTS.md)	Agent execution logs, traces	Scenario libraries, provenance records	Operational policies and incident playbooks
Testing	CI/CD optimized for velocity	Local integration tests, 25% effort allocation	Digital Twin Universe, satisfaction metrics	Incident simulation and production canaries
Validation	Prompt benchmarks	LLM-as-a-Judge, batch review tooling	Probabilistic satisfaction measurement	SLO and telemetry-based operational validation
Communication	Context specs, ADRs	High-bandwidth standups, co-location	Policy governance forums	Exception review and operational risk forums
Skills	Prompt calibration, context management	Trust in async execution, batch review	Scenario authoring, specification ownership	Operational policy design and systems governance
Governance	Prompt review practices	Escalation policies	Full governance-by-policy engine	Production authority and automated compliance evidence
Budget	Standard AI tooling costs	Moderate token increase	$1,000+/day per engineer in tokens	Level 6 plus continuous operations and simulation costs

The investment required for Level 6 is substantial. Level 7 adds production operations risk and cost on top. Organizations should honestly assess whether the productivity gains justify the infrastructure, tooling, token costs, and operational risk before committing.

Concerns and When Not to Advance

Higher autonomy is not universally better. Each level introduces new risks and may be inappropriate for certain contexts.

Cost Concerns

Level 6 token costs are an order of magnitude higher than Level 3-5. The $1,000/day per engineer baseline suggested by StrongDM represents $240,000/year in tokens alone per engineer—potentially exceeding salary costs. Level 7 may add continuous simulation, telemetry analysis, remediation, and incident rehearsal costs.

The economics only work if productivity gains justify the investment. Organizations should model expected throughput against token costs before committing.

Moat Erosion

Simon Willison raises a strategic concern:

> "Could competitors clone your newest features with a few hours of coding agent work?"

If autonomous coding commoditizes implementation, competitive advantage shifts to other factors: specifications, scenarios, Digital Twin quality, data, and go-to-market. Organizations relying on implementation complexity as a moat should consider this carefully.

Regulatory and Compliance Constraints

Some industries require human review for compliance reasons:

- Financial services with audit requirements
- Healthcare with patient safety regulations
- Defense and government contracting
- Any domain with legal liability for software failures

For these contexts, Level 5 or Level 6 may be the maximum achievable autonomy. Governance-by-policy can document intent, but regulators may require human attestation before deployment or during production operations.

Quality and Maintainability Risks

AI-generated code can accumulate technical debt invisibly. Systems optimized for scenario satisfaction may degrade in maintainability over time. Without human reading code, architectural drift goes unnoticed until it manifests as system failures.

Level 6 requires exceptional observability and architectural governance to catch degradation that human reviewers would otherwise flag. Level 7 raises the bar further because the system must detect, decide, and act while software is already serving users.

Choosing Between Levels 5, 6, and 7

Many organizations will find Level 5 the optimal target. Some will adopt Level 6 for constrained domains where scenario coverage is strong. Level 7 requires substantial additional investment and tradeoffs that few contexts will justify early.

Factor	Level 5 Optimal	Level 6 Optimal	Level 7 Optimal
Productivity goal	Significant gains sufficient	Maximum coding/review throughput	Maximum delivery and operations throughput
Error tolerance	Human review catches subtle bugs	Scenario coverage sufficient before production	Telemetry and rollback sufficient in production

Factor	Level 5 Optimal	Level 6 Optimal	Level 7 Optimal
Infrastructure budget	Moderate investment	High investment (DTU, scenarios, provenance)	Level 6 plus autonomous ops infrastructure
Regulatory context	Compliance requires human code attestation	No human review mandate	No human operational approval mandate
Codebase understanding	Team wants to maintain code literacy	Acceptable tradeoff for velocity	Acceptable tradeoff for operational speed
Token budget	Standard AI tooling costs	$1,000+/day per engineer	Level 6 plus continuous operations spend
Domain characteristics	Complex, ambiguous requirements	High-volume, well-specified domains	High automation tolerance and strong rollback paths
Risk tolerance	Conservative, proven patterns	Willing to remove code review	Willing to remove routine production gates

Most organizations should target Level 5 first and evaluate Level 6 only after operating successfully at Level 5 for an extended period. Level 7 should come later, after Level 6 validation infrastructure has proven itself and the organization has operational guardrails strong enough to tolerate autonomous production decisions.

How ASDM Compares to Dan Shapiro's Five Levels

Dan Shapiro's "Five Levels: from Spicy Autocomplete to the Dark Factory" is one of the clearest public framings of the same general shift. His model starts with an individual developer experience: from manual coding, to autocomplete, to pairing with AI, to managing coding agents, to a black-box system that turns specifications into software.

ASDM uses a different unit of analysis. It is an organizational software delivery model, not primarily an individual developer adoption ladder. That is why ASDM starts with Agile and DevOps as pre-AI organizational baselines, then separates AI-native workflows, agentic engineering, autonomous coding and review, and autonomous operations.

The two models rhyme, but they answer different questions:

Question	Shapiro's Five Levels	ASDM's Seven Levels
Primary lens	Individual developer workflow	Organizational delivery capability
Starting point	Manual coding and AI assistance	Agile and DevOps baselines
Middle transition	Developer becomes manager/PM of agents	Organization redesigns process, validation, governance
Factory endpoint	Specs turn into software without normal developer involvement	Intent flows through coding, validation, deployment, and operations under policy
Key distinction	How a person works with AI	What an organization can safely delegate to AI

Both models end near the same frontier: software production with humans outside the normal implementation path. ASDM splits that frontier into two levels because code generation and production operation are different trust boundaries. StrongDM is strong evidence for Level 6: no human coding or code review,

scenario validation, and Digital Twin testing. Level 7 requires a further claim: autonomous systems operate production software without routine human gates.

Conclusion

Software delivery is evolving from human-executed to machine-operated. This is not speculation—it is observable in the practices of teams at AWS, Spotify, StrongDM, and others.

The seven Software Delivery Autonomy Levels provide a framework for understanding this progression:

- Levels 1-2 established pre-AI baselines: team coordination and pipeline automation.
- Level 3 augmented existing processes with AI tooling.
- Level 4 redesigns processes around AI participation.
- Level 5 enables extended autonomous execution with batch human review.
- Level 6 eliminates human coding and code review, replacing it with scenario-based validation.
- Level 7 removes routine operational gates, replacing them with policy-governed autonomous operations.

Most organizations today operate between Levels 3 and 4. Leaders are reaching Level 5. A few pioneers are experimenting with Level 6. Level 7 remains the frontier.

ASDM provides the methodology for Levels 6 and 7—the practices, artifacts, and infrastructure required for autonomous coding, review, validation, and operations. Teams can adopt ASDM components incrementally as they progress through the levels.

The implications for product builders are significant. As implementation becomes increasingly automated, value shifts to:

- Clear specification of intent
- Comprehensive scenario definition
- Validation infrastructure
- Governance and policy design
- Operational telemetry and remediation design

The builders who thrive in this environment will be those who master not just AI tooling, but the organizational and infrastructural transformations that enable autonomous software delivery.

The factory is coming online. The question is not whether to engage with it, but at what level, and how fast.

References

- Magherimov, Joe. "Amazon Bedrock Mantle and Developing at the Speed of AI." AWS Podcast. Episode 753. AWS. January 2026. podcasts.apple.com/us/podcast/753-amazon-bedrock-mantle-and-developing-at-the-speed-of-ai/id1122785133
- AWS. "AI-Driven Development Life Cycle." Amazon Web Services. January 2026. aws.amazon.com/blogs/devops/ai-driven-development-life-cycle
- StrongDM. Software Factory. factory.strongdm.ai
- Willison, Simon. "The Software Factory." Simon Willison's Weblog. February 2026. simonwillison.net/2026/Feb/7/software-factory
- Shapiro, Dan. "The Five Levels: From Spicy Autocomplete to the Dark Factory." January 2026. www.danshapiro.com/blog/2026/01/the-five-levels-from-spicy-autocomplete-to-the-software-factory

- Spotify Engineering. “Honk: Spotify’s Path to Agentic AI Coding.” Spotify Engineering Blog. February 2026. engineering.atspotify.com/2026/02/honk-spotifys-path-to-agentic-ai-coding
- TechCrunch. “Spotify’s best developers haven’t written a line of code since December, thanks to AI.” TechCrunch. February 2026. techcrunch.com/2026/02/spotify-developers-ai-coding