Transforming enterprise software delivery through agentic AI

Authors - Mike Buob & Kevin Benner, PhD, Sogeti part of the Capgemini Group

Executive Summary

Enterprise software delivery is at an inflection point. While generative AI has delivered modest productivity improvements through AI assistants, the true transformation emerges when AI agents operate as disciplined members of the software delivery team, executing within a governed SDLC.

Agentic Software Engineering (Agentic SE) represents a shift from human executed tasks to agent executed workflows, where humans remain accountable for intent, architecture, and governance. When implemented with rigor — particularly through spec driven development and strong architectural boundaries — Agentic SE compresses delivery timelines from months to weeks, improves quality at the source, and dramatically reduces engineering friction.

This paper describes what Agentic Software Engineering is. It presents a real world case study demonstrating dramatic SDLC acceleration. It explores governance, orchestration, and architectural patterns required for enterprise adoption. And concludes with practical guidance for applying Agentic SE to both greenfield and legacy environments as part of a modern Product Operating Model.

1. From AI assistance to agentic software engineering

Most enterprises encounter generative AI first through AI assistants embedded in IDEs, documentation platforms, and chat interfaces. In this model, developers invoke GenAI through copy-and-paste prompts or inline suggestions to write snippets of code, explain existing logic, or generate test ideas. While useful, execution, sequencing, and validation remain firmly human-driven. These tools assist individuals, but they do not meaningfully alter how software is planned, built, tested, or released.

As a result, the gains from AI assistance tend to be incremental. Organizations typically see productivity improvements in the range of ten to twenty percent as cognitive load decreases and tasks complete slightly faster. However, the underlying delivery model remains unchanged. Development continues to be linear, coordination costs remain high, and quality issues are often discovered late. AI assistants optimize tasks, but they do not transform the system of delivery.

Agentic software engineering represents a fundamentally different operating model. Rather than humans directing work step by step, **specialized AI agents take on well-defined responsibilities across the software development lifecycle**. These agents are aligned to familiar enterprise roles — such as product discovery, architecture, design, feature development, quality engineering, security, and DevOps — and collaborate with humans through explicit, versioned artifacts rather than informal conversation. The shift is not merely technological; it is organizational. Work is no longer executed by individuals with tools, but by coordinated humans and agents operating as a delivery team.

Crucially, Agentic SE is not a model of unchecked autonomy. **Humans retain accountability for intent, architectural decisions, and governance**, while agents are responsible for execution, synthesis, iteration, and consistency within those bounds. This clear division of responsibility compresses the delivery lifecycle not by pushing people to work faster, but by removing friction between roles, eliminating redundant handoffs, and enforcing consistency at the source. What once unfolded over months can now be delivered in weeks.

Early uses of agents often take the form of "vibe coding," where small groups experiment with generalist agents to rapidly produce small applications or prototypes. While effective for exploration, this approach intentionally trades rigor for speed and is unconcerned with maintainability, enterprise constraints, or long-term governance. Scaling beyond experimentation requires a different discipline.

Enterprise adoption depends on spec-driven development, in which requirements, constraints, architectural decisions, and policies are captured explicitly and maintained as living artifacts. Specifications — not prompts — become the enduring contract between humans and agents. Architecture provides the separation of concerns that allows parallel workstreams to proceed safely, with each agent operating against a bounded context rather than an undefined global state. This spec-based, architecture-first model is what enables Agentic Software Engineering to operate reliably in mission-critical environments.

The speed of Agentic SE accelerates business outcomes by changing the cadence of delivery itself. Traditional software delivery—even when augmented with AI—remains largely linear: define requirements, build, test near release, and deploy in a single cutover. We talk about iterative development, but more typically we really mean incremental delivery. This is because the rework associated with applying the learning from one iteration to another is expensive in traditional models and we are often unwilling to pay that cost. Agentic SE dramatically lowers that cost.

Agentic Software Engineering replaces this with a **continuous delivery loop** in which requirements, implementation, testing, and deployment evolve together. As agents execute work, specifications are continuously refined through feedback rather than frozen early. Testing runs alongside development and deployment becomes progressive. Because re-coding and regression testing is automated, applying learnings from one iteration to the next is not expensive, thus allowing feedback to inform changes in requirements, design, or architecture.

This evolution in delivery mechanics is accompanied by a shift in how work is planned and governed. Conventional models emphasize task management—breaking work into tickets, assigning owners, and tracking completion. In an agentic model, planning becomes autonomy management. **Humans design product intent, constraints, and success criteria, while architecture serves as the scaffolding that enables agents to operate safely and independently within those boundaries.** Quality moves beyond static acceptance checks to continuous behavioral evaluation and governance, ensuring that agent-produced outcomes remain correct, compliant, and aligned as the system evolves.

Taken together, these changes explain why Agentic Software Engineering drives materially faster business outcomes, not just because tasks are executed more quickly, but because delivery is continuous, adaptive, and intentionally designed for safe autonomy rather than linear control.

2. Case study: delivery in weeks, not months

In one representative case study, a poorly performing application which was originally built in 10 months was rebuilt with significantly richer functionality in just 5 weeks.

The application supported on site engineers documenting requirements for building out a data center during site walk throughs. Engineers were required to capture dozens of detailed site requirements under time pressure and in complex physical environments. The legacy solution was manual and error prone. Engineers recorded information on paper while on site and later re entered those notes into a system, doubling effort and introducing inconsistencies, omissions, and rework. Latency between site visits and

completed proposals was high, and downstream teams frequently encountered unclear or conflicting inputs.

The desired outcome was a new application that could be used directly on laptops or smartphones during site visits and that enabled engineers to capture requirements in a structured, consistent, and immediately usable way. The business objectives were to reduce proposal turnaround time, eliminate duplicate data entry, improve data quality at the source, and minimize downstream rework.

The original application had been built using traditional delivery methods and required approximately ten months to complete. With an Agentic Software Engineering approach, the application was rebuilt from scratch in just five weeks. The solution was delivered as a single React / React Native codebase supporting web and mobile channels, fully responsive across devices. Structured, guided data capture replaced paper notes, ensuring requirements were captured consistently and available immediately for downstream processes. Delivery was executed by a small, focused team consisting of one architect, three software engineers, and one test engineer, with agents handling much of the execution work across design, build, and testing activities under human governance.

The results for the business were significant. Cycle time for creating new proposals was compressed enabling faster quoting and solution design. Manual steps and associated error rates were dramatically reduced, resulting in higher data fidelity at the point of capture. Most importantly, the engagement demonstrated the real world potential of Agentic Software Engineering to accelerate the SDLC.

3. Operationalizing agentic software engineering at enterprise scale

Successfully scaling Agentic Software Engineering in an enterprise environment requires more than introducing agents into existing tools. It requires rethinking how knowledge, constraints, and responsibilities are structured across the SDLC.

Agents as first-class delivery team members

Operationalizing Agentic Software Engineering begins with a clear understanding of what these “specialized agents” actually are. In an enterprise context, agents are not general purpose chatbots or ad hoc copilots responding to isolated prompts. They are role aligned, task scoped software actors designed to mirror the responsibilities that already exist within a mature software delivery organization. Each agent is purpose built around a specific SDLC role—such as product discovery, architecture, design, feature development, quality engineering, security, or DevOps—and operates against explicit artifacts rather than conversational intent.

Agentic Software Engineering frameworks define how these agents collaborate across the lifecycle. Rather than acting independently, agents operate as part of a coordinated system in which work is sequenced, parallelized, and governed through shared, versioned artifacts. Product and discovery agents help frame problems, articulate intent, and define success criteria. Architecture agents translate that intent into solution structures, boundaries, and contracts. Design agents define personas and journeys. Build and test agents execute against those specifications, generating code and validating behavior, while security and compliance agents continuously assess adherence to nonnegotiable enterprise constraints. Throughout this process, humans retain authority over intent, architecture, and approval, but they are no longer the primary execution engine.

Critically, orchestration is not achieved through conversational hand offs or implicit understanding. It is driven by an orchestration role—often implemented as a supervised PM or coordination agent that manages dependencies, assigns work, enforces sequencing rules, and governs when agents may proceed autonomously versus when human intervention is required. This orchestrator reasons over artifact state, architectural boundaries, and declared dependencies, ensuring that agents behave like disciplined team members operating within a shared delivery system, rather than as isolated tools reacting to prompts.

This distinction is what allows agentic SDLC to scale. By anchoring agent behavior to explicit roles, artifacts, and governance mechanisms, Agentic SE frameworks transform collections of intelligent tools into a coherent delivery capability.

In practice, this orchestration model manifests as a clear division of responsibility between humans and agents across the SDLC, with each role operating against explicit artifacts and governance rules. For example,

Role	**Human / Agent**
Product Owner	Human
PM / Orchestrator	Agent (often supervised)
Architect	Human + Agent
Feature Dev	Agent
Test Engineer	Agent
Security / Compliance	Agent
Reviewer	Human

For human team members, teams tend to be smaller as agents pick up significant responsibilities. Humans focus their attention on product feature prioritization, design intent, architecture, orchestration and governance rather than implementation. During review processes feedback may be provided conversationally, but if the feedback contains new requirements, constraints, or architecture decisions, they are added to the appropriate artifact.

Architecture as the system of record

At the core of this model is the idea that a **living specification is the system of record.** Requirements, architectural decisions, policies, code, tests, and operational artifacts all live together in a shared, versioned repository. Conversations are transient; artifacts are authoritative. Agents operate by reading, producing, and updating these artifacts, with humans validating at defined control points.

Architecture plays a central role in making this model safe and scalable. Traditional architecture has always focused on separation of concerns, but in a world of large language models, this separation becomes essential for managing LLM context. Enterprise systems involve vast bodies of knowledge: security policies, regulatory constraints, coding

standards, data governance rules, platform guardrails, and shared services. Loading all of this into an agent's context at once dramatically increases the risk of hallucination and faulty behavior.

A well designed architecture solves this problem by decomposing enterprise knowledge into bounded contexts (aka breaking the solution into separate pieces with well-defined connections). Business architecture defines domains and capabilities. Application architecture defines services and systems. Data architecture defines ownership and data products. Design defines personas, journey maps, service blueprints and user flows. Security architecture defines trust zones and controls. Each architectural boundary determines what an agent needs to know, and equally important, what it does not.

For example, in a health plan environment, an agent working on a member eligibility service is provided with HIPAA requirements, PII handling rules, and audit logging policies, but not provider credentialing policies or claims adjudication rules.

Agent interaction with enterprise knowledge is deliberately incremental. Agents do not preload the full enterprise corpus. Instead, when work is assigned — typically by a supervising PM or orchestration agent only the necessary specifications, standards, and decision records are provided. This just in time approach keeps agents effective, auditable, and aligned.

Reframing agile in the context of high-speed agentic delivery

One of the most substantiative changes with the adoption of Agentic SE is the speed with which agentic powered teams iterate. Entire epics and features are built in a day or two. Traditional sprint planning, most agile ceremonies, planning poker and burn-down charts don't make sense in this context. Sprints were two weeks because it took that long (even for good teams) to create enough working code to show stakeholders. Now that same output or more is available in a day.

At this new speed, stakeholders need to engage nearly daily providing meaningful feedback on the working system. Product owners focus on defining goals, constraints, autonomy boundaries, success metrics, and the business context.

With this speed and efficiency, enterprises can now step up to a full product operating model and all the disciplines it requires to create customer centric solutions.

Realizing a product operating model

Many enterprises aspire to adopt a modern Product Operating Model (POM), as articulated by thought leaders such as Marty Cagan, but struggle to realize it consistently in practice. Effective product organizations depend on a set of disciplines that must operate in concert: continuous product discovery, strong product management and ownership, user research and experience design, outcome oriented planning, rapid experimentation, architectural alignment, and close collaboration between business and technology. While well understood conceptually, executing all of these disciplines at scale is time consuming and expensive, often limiting adoption to a small number of flagship teams.

Agentic Software Engineering materially changes this equation by **systematizing core product management disciplines into the delivery fabric itself.** Rather than relying solely on scarce human capacity, the principles, constraints, and expectations of an effective product operating model are embedded directly into specialized agents. Product and discovery agents continuously frame problems, articulate hypotheses, and refine success criteria. UX agents institutionalize user-centered design and accessibility practices. Architecture agents ensure technical choices reinforce product intent. Quality and

measurement agents reinforce outcome-based evaluation rather than output-based completion.

By encoding these disciplines into agents and governing them through spec-driven artifacts, Agentic SE allows product practices to be applied **consistently and continuously**, not episodically or heroically. Product discovery is no longer frontloaded and forgotten; it runs alongside delivery. The result is a practical path to adopting a true Product Operating Model at scale—one where product thinking is not dependent on exceptional teams but is reinforced by the delivery system itself.

Agent-driven architecture refactoring

One of the most impactful ways Agentic Software Engineering changes the traditional development mindset is through its ability **to rapidly refactor software architecture in response to major new requirements**. In conventional delivery models, large scale architectural refactoring is often avoided or deferred because it is slow, risky, and highly disruptive to ongoing development. This is the root cause of much tech debt. Agentic SE changes this dynamic by treating refactoring as a coordinated, governed activity executed by multiple specialized agents operating within clearly defined architectural and process boundaries which can be accomplished in hours, not days and weeks.

Architectural refactoring in an agentic model begins with deliberate impact analysis. An architecture agent evaluates the proposed change, assesses downstream dependencies, and produces a refactoring plan that makes explicit which services, contracts, and shared artifacts are affected. This analysis is not merely descriptive; it becomes a first class artifact that informs sequencing, ownership, and governance for the rest of the effort.

To prevent unintended interference from parallel development, the refactoring process introduces explicit locking rules within the artifact repository enforced by the orchestration or PM agent. Affected artifacts are marked as temporarily locked, with the rationale for the lock captured directly in the repository metadata. Other agents are instructed to avoid these areas until the refactor is complete. This approach replaces informal coordination and tribal knowledge with explicit, machine enforceable rules that reduce risk while allowing the rest of the organization to continue moving forward.

Actual refactoring work occurs within dedicated branches, where architecture agents and core build agents operate in relative isolation. Code changes, contract updates, and internal restructuring proceed in a controlled environment, while testing agents update and extend test coverage in parallel branches aligned to the evolving architecture. This separation allows deep architectural work to progress without destabilizing active feature development.

Reintegration is handled in stages rather than as a single disruptive merge. As refactored modules reach readiness, they are re enabled in deliberate waves. Feature agents resume work only after affected components are merged and validated, ensuring that downstream development builds on the new architectural foundation rather than competing with it.

What emerges from this pattern is a synthesis of familiar enterprise disciplines—GitOps, SDLC governance, and architectural control—augmented by agent driven execution. Architectural refactoring becomes repeatable, auditable, and significantly faster, not because control is abandoned, but because coordination and execution are handled systematically by agents operating within well defined roles and constraints.

The refactoring example illustrates that, agentic SE scales when agents are treated as disciplined team members, not magical interns. Roles are explicit. A shared repository provides the backbone for managing artifacts. Branching strategies isolate concurrent

work. Architectural changes are staged and controlled. A coordinating PM agent tracks dependencies, artifact ownership, branch state, risk levels, and merge readiness, enforcing rules about when humans must intervene and when agents may proceed autonomously.

Applying agentic software engineering to legacy and brownfield systems

Agentic Software Engineering is particularly powerful in brownfield environments where legacy systems dominate, and institutional knowledge is often concentrated in the hands of a small number of long tenured developers. In these settings, documentation is incomplete or outdated, architectural intent has eroded over time, and the risk associated with change increases as subject matter experts approach retirement or leave the organization.

Agentic SE addresses these challenges by treating understanding and formalization of legacy systems as first class engineering activities. Architecture and analysis agents can ingest existing codebases, runtime behavior, historical change patterns, and operational data to reconstruct architectural models, surface implicit assumptions, and generate living documentation that was never formally captured. This knowledge is then externalized into versioned specifications, decision records, and contracts—shifting critical understanding from individual memory into durable artifacts.

As a result, brownfield modernization becomes safer and more repeatable. Refactoring and enhancement can proceed incrementally, with agents continuously validating behavior against reconstructed intent. New team members ramp faster, dependency on a shrinking pool of experts is reduced, and the organization retains control over systems that would otherwise become opaque and brittle. In practice, Agentic Software Engineering does not simply accelerate new development — it provides a pragmatic path for enterprises to **stabilize, understand, and evolve the systems they depend on.**

4. In conclusion

Agentic Software Engineering is not simply an evolution of AI assisted development—it is a structural change to how enterprise software delivery is designed, governed, and executed. By shifting execution to specialized agents operating within explicit architectural boundaries and spec driven artifacts, enterprises can achieve materially faster delivery while strengthening quality, compliance, and control. Delivery cycles shrink from months to weeks while improving quality.

sogeti labs
Part of Capgemini

About Sogeti

Sogeti is a leading provider of technology and engineering services. Sogeti delivers solutions that enable digital transformation and offers cutting-edge expertise in Cloud, Cybersecurity, Digital Manufacturing, Digital Assurance & Testing, and emerging technologies. Sogeti combines agility and speed of implementation with strong technology supplier partnerships, world class methodologies and its global delivery model, Rightshore®. Sogeti brings together more than 25,000 professionals in 15 countries, based in over 100 locations in Europe, USA and India. Sogeti is a wholly-owned subsidiary of Capgemini SE, listed on the Paris Stock Exchange.

Learn more about us at

www.sogeti.com