Governed Agentic Delivery: The Agentwerke Method and Platform Definition

Isartor AI · Version 1.0 · July 2026

ABSTRACT

AI agents can now plan, code, test, review, and open pull requests. AI-native methods such as AWS's AI-Driven Development Lifecycle (AI-DLC) reimagine the software lifecycle around that capability: AI initiates and drives the work while humans oversee it at critical junctures. This paper argues that in an enterprise, oversight cannot remain a convention — it must be enforced by infrastructure. We define **Governed Agentic Delivery**, the method behind **Agentwerke**, a self-hosted control plane in which every agent job runs a versioned BPMN process, every tool call crosses a policy gate, sensitive work executes in sandboxes, humans hold explicit approval choke points, and every run emits an audit-ready evidence pack. The result is not slower agents; it is autonomous delivery an engineering organization can inspect, interrupt, and defend.

I. CONTEXT

The evolution of software engineering has been a continuous effort to let developers focus on the problem rather than the plumbing. Large Language Models first delivered the **AI-Assisted** era — completion, bug detection, test generation — where AI accelerates fine-grained tasks under continuous human direction. The frontier has since moved to the **AI-Driven** era: agents that elaborate requirements, decompose work, write and test code, and open pull requests with minutes of human attention rather than hours.

Method work has begun to catch up. AWS's *AI-Driven Development Lifecycle (AI-DLC) Method Definition* is a notable example: it reverses the conversation direction so that AI initiates and drives the workflow, compresses iterations from weeks to hours ("Bolts"), and positions human oversight as a *loss function* — catching errors at critical junctures before they snowball downstream. We agree with this reimagination, and Agentwerke is designed to execute exactly such methods.

But a method definition answers *what should happen*. In a regulated engineering organization, a second question decides whether autonomous delivery is adoptable at all: *what enforces it?* When rituals live in prompts and conventions, the gaps show up quickly:

- **Opaque decisions.** Agent reasoning hides in prompt transcripts. No versioned process describes what an agent was allowed to attempt, or why it did what it did.
- **Weak auditability.** Merged code arrives with no defensible trail of prompts, tool calls, costs, and approvals — exactly what compliance and incident review need.
- **Uncontrolled tool access.** Credentials, networks, repositories, and deployment paths sit one unguarded tool call away from an autonomous process.
- **Data exposure.** Prompts and context flow to third-party model endpoints with no redaction boundary, residency control, or secret-handling discipline.
- **Inconsistent process.** Every team wires agents differently; handoffs between planning, build, review, and deployment stay manual and low-visibility.
- **No human choke points.** High-risk steps complete without an explicit decision. Approval is bolted on after the fact instead of being part of the process.

"Lights-out" manufacturing proved that a factory can keep running without people on the floor. Philip K. Dick's 1955 story *Autofac* warned what happens when automated production continues after meaningful human control has slipped away. Agentwerke takes the name in a different direction — *agent* plus *Werke*, the engineered works where software agents operate. The thesis is not an autonomous factory beyond control. It is a professional software factory where agents plan, build, integrate, and deliver **under enterprise governance**.

The point is not to make agents harmless. The point is to make powerful automation **inspectable, interruptible, and accountable**.

This paper defines Governed Agentic Delivery in three parts: the principles that shape it (Section II), the core framework of artefacts, components, and workflow that implements it (Section III), worked green-field and brown-field scenarios (Sections IV–V), its relationship to AI-native methods such as AI-DLC (Section VI), and an adoption path (Section VII).

II. KEY PRINCIPLES

1. Govern rather than trust. Agents can act — plan, code, call tools, open pull requests — but they never get direct access to credentials, networks, repositories, or deployment paths. Every capability is mediated. Trust is placed in the enforcement layer, not in the model's disposition, and therefore survives model upgrades, prompt changes, and adversarial inputs.

2. Workflow over vibes. Software delivery is modeled as versioned BPMN 2.0, not hidden in a prompt transcript. The process is an artefact: it can be designed, reviewed, diffed, versioned, and — critically — *executed*. Agent steps, approval gates, wait states, timers, and gateways are first-class nodes. Because BPMN is an open standard, the process model is portable and readable by tools and auditors that predate the AI era.

3. Policy is evaluated at execution time, not asserted in a prompt. A prompt instruction ("do not push to main") is a request; a policy gate is a guarantee. In Agentwerke, every tool and connector call is brokered through a gateway and evaluated *before* it runs. Decisions are **allow, escalate, or reject**, each carrying a purpose-confidence and risk score, and each recorded. Policies are data, not code, with a **draft → simulate → publish** lifecycle and impact analysis, so a policy change can be rehearsed against history before it governs production. Per-run cost and token budgets bound the economics the same way: model calls halt once the budget is exceeded.

4. Sandboxes by default. Agent and tool execution happens inside controlled environments — Docker, OpenSandbox, or Kubernetes — selected per step by sandbox profile and network policy. The blast radius of any single step is a configuration, not a hope.

5. Humans at the right choke points. High-risk steps pause for an explicit decision. Approval gates, wait states, and escalations are part of the process model — not bolted on later — and the decider is written into the audit trail. Agents can also *ask*: a blocking question suspends the run until a person answers, so human judgment is available mid-process without holding a thread open. This operationalizes the oversight-as-loss-function idea: errors are caught and pruned at defined junctures before they snowball.

6. Evidence as exhaust. Every run emits a schema-versioned, tamper-evident **evidence pack**: prompts (redacted in the persisted snapshot), model and tool calls, policy decisions with rationale, sandbox executions, approvals with decider and timestamp, per-run token and cost accounting, artifacts, and outcomes — with the workflow's BPMN hash bound in for integrity. Evidence is produced as a by-product of execution, not reconstructed after the fact. It is the difference between "the agent wrote this" and "here is exactly how, under which policy, and who approved it."

7. Self-hosted by design. The factory runs on your infrastructure, with your model keys and your data boundary. Workflow definitions, runs, approvals, audit records, and run context stay in your PostgreSQL database; artifacts stay in your storage. Data leaves the boundary only through integrations and model providers you explicitly enable — which is what makes residency, secret-handling, and redaction requirements satisfiable rather than aspirational.

8. Open core, open standards, no method lock-in. Agentwerke is open-core (Apache-2.0) and builds on open standards: BPMN 2.0 for process, the Model Context Protocol (MCP) for tools, OIDC for identity. It does not prescribe an opinionated lifecycle. Whatever method an organization adopts — AI-DLC, V-model, a regulated variant of Scrum — Agentwerke executes it as a modeled, governed process. The method is the content; Agentwerke is the enforcement.

III. CORE FRAMEWORK

Agentwerke is a layered .NET control plane. This section defines its artefacts, components, and the end-to-end workflow of a run.

1. Artefacts

Workflow definition. A BPMN 2.0 process carrying Agentwerke extension metadata (`agentwerke.de/bpmn/extensions/v1`). The extensions turn a BPMN service task into an *agent work cell* (which agent profile, which skill, which sandbox profile) and a BPMN user task into a *governed approval gate*. Definitions are versioned; a run is always bound to the exact version — and hash — it executed.

Run. One execution of a workflow definition, durable and event-sourced. A run advances through nodes,

pauses at approvals, wait states, and blocking questions, and resumes from external signals (a webhook, green CI, a merged PR) or timers. Runs survive restarts; state lives in PostgreSQL.

Agent profile and skills. The configuration an orchestrated step assembles before the model receives work: the agent's role, its skill (task instructions), its run context, and the tools it is permitted to request.

Policy. A data-driven rule set governing actions. Policies are authored as drafts, *simulated* against historical traffic to see what they would have allowed, escalated, or rejected, then published. Every subsequent decision records which policy version produced it.

Approval. An explicit human decision attached to a gate: approve or reject, with the decider, timestamp, and context captured — including interactive approve/reject from Slack.

Evidence pack. The schema-versioned record of a run described in Principle 6, retrievable per run via API (`GET /api/runs/{runId}/evidence-pack`), designed to be handed to an auditor.

2. Components

BPMN workflow runtime. A bounded, Postgres-backed, in-process engine with event-sourced checkpoints executes versioned BPMN. Runs are dispatched asynchronously via a transactional outbox drained by a background worker, so intake returns immediately and the engine advances the run — and resumes it on approval, external event, or timer. External waits are bounded by interrupting boundary timers. An enterprise Camunda 8 adapter is available where organizations standardize on it.

Agent orchestrator. When a run reaches an agent task, the orchestrator assembles the profile, skill, run context, and permitted tools, then drives the model loop. Model providers are pluggable — Anthropic Claude natively, plus any OpenAI-compatible endpoint (OpenAI, Azure OpenAI, or a LiteLLM proxy) — and a deterministic mock provider supports zero-cost demos and CI. Per-run cost and token budgets halt model calls once exceeded.

Policy-enforced tool gateway. The single door between an agent and the world. Every tool and connector call — repository writes, CI triggers, chat messages, MCP tools — is brokered, evaluated against published policy, executed in the right boundary, and recorded with its decision and rationale.

Sandboxed execution. Isolated Docker, OpenSandbox, or Kubernetes environments host agent and tool work, selected per step by sandbox profile and network policy, with a dedicated runner image as the entrypoint.

Human interaction. Approval gates pause runs for decisions; `human.ask` lets an agent pose a blocking question that suspends the run until answered; `human.notify` sends a non-blocking heads-up. Agents can also coordinate with each other over a run-scoped message bus and delegate bounded sub-tasks. All of these exchanges persist as auditable interaction records that also back the run's conversation view.

Enterprise integrations. GitHub and Jira intake; GitHub branches, pull requests, and reviews; CI/CD; Slack and Microsoft Teams; and any MCP-compatible tool — all brokered through the same policy gateway.

Enterprise foundation. OIDC/JWT SSO against your identity provider, role-based access control (Viewer, Operator, Approver, Admin) written into the audit trail, LDAP/AD group-to-role mapping, secret handling through a secret store (the API exposes status and fingerprint, never raw values), and artifact storage on your filesystem or S3 bucket. Deployment is Docker Compose for a single host or Kubernetes/Helm for production.

Operations cockpit. A web UI provides a BPMN workflow designer, run board, run detail with live per-step status on the diagram, approvals dashboard, policy management, and audit views.

3. The Workflow

Every run follows the same governed pipeline:

> **Trigger** (API call or webhook) → **BPMN run** (versioned process) → **Agent task** (model loop) → **Policy gate** (allow / escalate / reject) → **Sandbox + tool gateway** (isolated execution) → **Human approval** (explicit decision) → **Evidence pack** (audit + artifacts)

Three properties of this pipeline carry the governance argument. First, the process is *declared before execution*: what the run may attempt is readable from the workflow definition, not discovered from a transcript. Second, enforcement is *positioned at execution time*: the policy gate sits between the agent's intent and the world's state change, and the same gate serves every integration. Third, the record is *produced by the pipeline itself*: evidence accumulates as the run advances, and the pack binds to the BPMN hash of the exact process version that ran.

IV. IN ACTION: GREEN-FIELD DELIVERY

Consider the canonical flow: a GitHub issue becomes a reviewed pull request.

1. **Intake.** A labeled GitHub issue fires a webhook; the API validates the payload and starts a run of the issue-to-PR workflow. The intake responds immediately; the outbox worker advances the run asynchronously.
2. **Analysis (agent task).** The orchestrator assembles the analyst profile and the run context (issue body, repository metadata) and drives the model loop. The agent reads the repository through policy-gated tools and produces an implementation plan as an artifact.
3. **Plan approval (human gate).** The run pauses. An Approver reviews the plan — in the cockpit or directly from Slack — and approves. The decision, decider, and timestamp enter the audit trail.
4. **Implementation (agent task, sandboxed).** A sandbox is provisioned per the step's profile and network policy. The agent edits code, runs tests, and commits — every repository write brokered through the tool gateway and checked against policy. An attempt to touch a path or branch outside policy is escalated or rejected, and recorded either way.
5. **Pull request.** The agent opens a PR through the GitHub connector. The run enters a wait state bounded by a timer.
6. **External signals.** Green CI and the PR review arrive as webhooks and resume the run. A failed check can route back to the implementation task; a timeout escalates to a human.
7. **Evidence.** The run completes and its evidence pack is retrievable: plan, prompts (redacted), every tool call, every policy decision, the sandbox profile, the approvals, the per-run cost, and the artifacts — bound to the workflow version that produced them.

The deliverable is not just the pull request. It is the pull request *plus* the defensible record of how it came to exist.

V. IN ACTION: BROWN-FIELD CHANGE

Brown-field work — a defect fix, an enhancement, refactoring within an existing system — follows the same pipeline with a different emphasis. The analysis task begins by building context from the existing code before proposing a change plan; policies are typically tighter (protected paths, constrained dependencies, mandatory human review on migration or configuration changes); and wait states integrate with the organization's existing CI and review process rather than replacing it. Because the process is a modeled artefact, a team encodes its brown-field rules once — in the workflow and the policy set — instead of re-asserting them in every prompt.

VI. RELATIONSHIP TO AI-NATIVE METHODS

Governed Agentic Delivery is not a competitor to AI-DLC and similar AI-native methods; it is their execution substrate. The mapping is direct:

- An AI-DLC **Intent** arrives as a trigger payload; the **Level 1 plan** is the workflow definition a team has modeled and versioned.
- **Units and Bolts** decompose into BPMN subprocesses and agent tasks that can run in parallel where the model allows.
- **Mob Elaboration and validation rituals** — the human oversight AI-DLC places at critical junctures — become approval gates and blocking questions: enforced pauses, not calendar conventions.
- **Oversight as a loss function** becomes machinery: policy gates prune disallowed actions at the moment of execution, and approval gates prune wrong directions between phases.
- The artefact trail AI-DLC calls **context memory** persists as run context, interaction records, artifacts, and the evidence pack.

A method definition tells an organization what good AI-native delivery looks like. A governed runtime is what lets a compliance function sign off on actually running it. Enterprises need both.

VII. ADOPTION

Agentwerke is designed for incremental adoption:

1. **Start tokenless.** The quickstart ships a minimal hello-SDLC workflow that runs against the deterministic mock model — the full pipeline (process, gates, evidence) with zero model cost and no keys.
2. **Govern one real flow.** Connect a repository and run issue-to-PR with a real model, a conservative policy set, and human approval on every gate. Inspect the evidence packs; tune policies with the simulate lifecycle.
3. **Widen autonomy deliberately.** As confidence and policy maturity grow, relax gates where evidence supports it — approvals concentrate on genuinely

high-risk steps, and the loss function moves earlier into policy.

4. **Scale into the enterprise.** SSO, RBAC, LDAP/AD mapping, residency controls, and Kubernetes/Helm deployment carry the same pipeline to organization scale.

Because the platform is open-core under Apache-2.0, teams can adopt, audit, and extend the factory floor itself — and bring their own method to run on it.

APPENDIX A: A MINIMAL GOVERNED WORKFLOW

The Agentwerke extension metadata inside standard BPMN 2.0 is what turns a process diagram into a governed factory line. A minimal example — one agent work cell and one human gate:

```
<bpmn:definitions
    xmlns:bpmn="http://www.omg.org/spec/BPMN/20100524/
MODEL"
    xmlns:agentwerke="https://agentwerke.de/bpmn/exten
sions/v1">
  <bpmn:process id="HelloSdlc" name="Hello SDLC" isExe
cutable="true">
    <bpmn:startEvent id="Start" name="Start" />
    <bpmn:serviceTask id="Analyze" name="Analyze">
      <bpmn:extensionElements>
        <agentwerke:agentTask agentProfile="analyst"
                              skill="analyze-issue" />
      </bpmn:extensionElements>
    </bpmn:serviceTask>
    <bpmn:userTask id="Review" name="Review Approval">
      <bpmn:extensionElements>
        <agentwerke:approvalTask role="Approver" />
      </bpmn:extensionElements>
    </bpmn:userTask>
    <bpmn:endEvent id="Done" name="Done" />
  </bpmn:process>
</bpmn:definitions>
```

An ordinary BPMN editor reads this; the Agentwerke runtime executes it; the evidence pack records it.

REFERENCES

1. Raja SP, *AI-Driven Development Lifecycle (AI-DLC) Method Definition*, Amazon Web Services.
2. Agentwerke documentation — docs.agentwerke.de
3. Agentwerke source repository — github.com/isartor-ai/agentwerke
4. Philip K. Dick, “Autofac,” *Galaxy Science Fiction*, 1955.