WHITEPAPER • ARCHITECTURE & SOFTWARE GOVERNANCE

DEOVD: A Software Development Operating Model for the AI Era

Discover → Engineer → Orchestrate → Validate → Deliver

Author: Ananth Godavari • Enterprise Systems Architecture

Target Domain: Agentic Software Engineering

Date: June 2026

EXECUTIVE SUMMARY / ABSTRACT

Artificial intelligence is changing software development more profoundly than simply making programmers faster. Traditional software delivery methodologies were built around a world in which human developers performed most implementation work directly. AI-assisted and agentic development fundamentally changes that assumption. Controlled research demonstrates substantial acceleration in individual coding tasks, while current DORA research characterizes AI as an *organizational amplifier*: it can increase throughput, but it can also amplify instability when the underlying engineering system is weak. The challenge is no longer simply how efficiently developers write code, but how effectively an organization defines the right problem, engineers the context, orchestrates AI-assisted execution, proves the system works, and places it safely into operation. **DEOVD** is a software delivery model designed specifically around that reality.

1. DISCOVER	2. ENGINEER	3. ORCHESTRATE	4. VALIDATE	5. DELIVER
Problem & Context	Architecture & Spec	Guided Execution	Continuous Proof	Operational Capability

1. Why the Software Development Model Must Change

Modern DevOps recognizes planning, development, delivery, operations, source control, automated testing, and continuous integration as parts of an integrated software lifecycle. AI does not make those disciplines obsolete; it fundamentally changes the economics of the work inside them.

Historically, software implementation was expensive because producing software required significant human manual effort. A developer had to understand a requirement, determine system behavior, locate code, write implementation, debug, test, and integrate. The speed of manual implementation placed a natural constraint on organizational output.

AI **substantially reduces that manual constraint**. An AI coding agent can inspect thousands of lines of source code, generate changes across multiple files, construct database scripts, propose test cases, and update documentation within minutes. As generation becomes faster, generation itself becomes less scarce.

> *What becomes scarce instead is correct understanding, sound engineering judgment, sufficient context, effective direction, architectural consistency, continuous validation, and organizational accountability.*

This shifts the developer's primary identity from being a producer of source code to an engineer and orchestrator of software production. Industry guidance from major platforms similarly emphasizes planning, acting, evaluation, supervision, governance, and traceability rather than mere syntax completion. DEOVD makes that paradigm shift explicit.

2. Traditional Software Development vs. The AI-Era Shift

A simplified traditional lifecycle (Requirements → Design → Development → Testing → Deployment) relied on iterative human coding. DEOVD preserves core engineering rigor while shifting the operational focus across all phases.

DEOVD is iterative, not sequential. Work may move backward or forward between phases as new evidence changes understanding, design, implementation, or readiness.

Phase / Focus	Traditional Paradigm	DEOVD AI-Era Paradigm
Problem Scoping	Gather requirements	Discover actual problem & operating context
System Design	Design the technical solution	Engineer solution & execution context
Implementation	Developers write code manually	Orchestrate humans, AI agents, & tools
Quality Assurance	QA tests completed work	Validate continuously & independently
Release	Deploy compiled code	Deliver working, supportable capability
Primary Artifact	Source code	Context, rules, code, evidence, & knowledge
Core Metric	Coding output & velocity	Verified business outcomes & stability

> ***The Fundamental Transition:*** *Traditional development organizes people who produce software. DEOVD organizes people, AI, tools, and evidence that collectively produce software.*

3. Discover: Problem & Context Definition

The primary failure mode of AI-assisted development is building the wrong thing—at unprecedented speed. Discover establishes the business problem, desired outcome, current system behavior, stakeholders, constraints, dependencies, and risks before implementation decisions are made.

Discovery inspects support incidents, source code, production logs, database schemas, user behavior, and operational evidence. Crucially, Discover recognizes that an incoming request is mere evidence, not the absolute requirement.

Raw Discovery Input	Structured Engineering Result
Business request	Defined business objective & bounded success criteria
System symptoms	Root problem definition
Existing code & docs	Current-state operational understanding
Logs & telemetry	Technical findings & confirmed data facts

DISCOVER GOVERNANCE QUESTION

Do we understand enough about the real problem to engineer a solution without making blind architectural assumptions?

4. Engineer: Specification & Context Engineering

When AI agents can generate thousands of lines of code rapidly, poor architecture propagates rapidly. Engineer is the phase where intent becomes structured implementation context: architecture, integration contracts, security constraints, data models, acceptance criteria, and explicit agent rules.

In AI-assisted development, **context itself becomes a primary engineering artifact**. Prompt engineering is insufficient; the larger discipline is *context engineering*. The engineer determines what the AI needs to know, what it may modify, what it must preserve, which constraints are authoritative, and what constitutes verified completion.

Traditional Artifact	AI-Era Context Extension
Requirements	Machine-usable structured context
Architecture	Implementation boundaries & repository constraints
Business Rules	Explicit rule catalogs & policy specs
Acceptance Criteria	Verifiable completion conditions & assertions

ENGINEER GOVERNANCE QUESTION

Could a capable team—or autonomous AI agent—execute this work without inventing critical requirements or architectural decisions?

5. Orchestrate: Supervised Execution

Orchestrate replaces the traditional word "Develop". It recognizes that software is now produced by a hybrid combination of AI coding agents, developers, specialized models, CI/CD pipelines, and automated tools.

The senior developer acts as a technical conductor, determining work delegation, task granularity, context boundary, human-in-the-loop (HITL) review gates, and branch integration. AI agents are managed through controlled iterative execution:

> ***Direct → Execute → Inspect → Correct → Integrate → Repeat***

Instead of asking *"What code should I type next?"*, the developer asks *"What should happen next, who or what should perform it, what context is required, and how will I know that the result is correct?"*

ORCHESTRATE GOVERNANCE QUESTION

Is execution occurring within defined architectural, security, scope, and human-oversight boundaries?

6. Validate: Continuous Verification & Evidence

Generative systems produce syntactically coherent code that may harbor subtle defects, security flaws, or architectural drift. **The marginal cost of generating code falls dramatically**, but the cost of trusting incorrect code remains high. DORA research shows that AI can amplify weaknesses in the surrounding software delivery system, making strong engineering and validation practices increasingly important.

Validation surrounds orchestration as a continuous control mechanism. It includes functional testing, compilation, static code analysis, security auditing, performance testing, and alignment with original criteria. Importantly, validation establishes explicit recursive feedback loops:

- **Validate → Orchestrate:** Implementation defect; code or tests require re-execution.
- **Validate → Engineer:** Architectural defect; execution context or specifications were flawed.
- **Validate → Discover:** Intent defect; original understanding of the problem was incorrect.

> ***Core Principle:*** *Generated is not completed. Implemented is not proven. Something becomes trustworthy only when sufficient evidence demonstrates that it behaves as intended.*

VALIDATE GOVERNANCE QUESTION

Do we have sufficient independent evidence that the system behaves as intended and meets its acceptance criteria?

7. Deliver: Operational Capability & Ownership

Deliver converts a validated implementation into an operational business capability. Deployment is a technical event; delivery is a business outcome. This phase covers release automation, production verification, documentation, monitoring setup, support handoffs, and formal operational ownership.

DELIVER GOVERNANCE QUESTION

Is the capability operationally ready, owned, observable, supportable, and safe to place into use?

8. Phase Accountability Matrix

While AI capability expands across all phases, human accountability remains mandatory at organizational boundaries:

Phase	Human Role (Accountable)	AI Role (Automated / Assisted)
Discover	Accountable for problem scope & business priorities	Analyze telemetry, logs, & summarize context
Engineer	Accountable for architecture & system constraints	Generate design options & machine-readable specs
Orchestrate	Supervise execution & enforce boundary controls	Perform multi-file generation & execution tasks
Validate	Accountable for risk acceptance & safety gates	Generate test cases, inspect diffs, & analyze failure
Deliver	Accountable for operational ownership & release	Automate release scripts, docs, & monitoring setup

9. DEOVD as an Enterprise Operating Model

DEOVD is not intended merely as a naming convention for development phases. It is an operating model for allocating responsibility in AI-assisted software delivery. Each phase identifies the dominant purpose of the work, the artifacts required to proceed, the role of human judgment, the role of AI automation, and the evidence necessary to move forward.

Software development is entering an era where generating code is no longer the primary constraint. Competitive advantage will belong to organizations that master problem discovery, context engineering, AI orchestration, continuous validation, and reliable delivery. DEOVD provides the governance framework to transition engineering teams from manual code writing to controlled, high-throughput intelligent execution.

10. Key References

1. **DevOps Research and Assessment (DORA).** (2025). *State of AI-assisted Software Development*. Google Cloud.
2. **Tabassi, E.** (2023). *Artificial Intelligence Risk Management Framework (AI RMF 1.0)* (NIST AI 100-1). National Institute of Standards and Technology, U.S. Department of Commerce. DOI: 10.6028/NIST.AI.100-1.
3. **Microsoft.** (n.d.). *Foundations of Agentic AI in GitHub*. Microsoft Learn.
4. **Gartner.** (2024). *Top Strategic Technology Trends for 2025: Agentic AI*. Published October 21, 2024.