The Enterprise Guide to Agentic Development

cursorworkshop.com

January 2026

Most organizations get it wrong. They deploy Cursor, Copilot, or Claude across their engineering teams. Junior developers generate code they cannot properly verify while senior engineers see marginal productivity gains. The organization lacks a shared framework for what AI should handle versus what requires human judgment.

The data is clear. Research from Jellyfish [link here] shows senior engineers are 22% faster with AI coding tools. Junior developers? Just 4% faster. The tools are identical. The gap is in how people use them.

AI multiplies existing engineering capability. A strong engineer becomes stronger. A developing engineer sees minimal improvement. The question becomes: how do you elevate everyone toward the higher end of that spectrum?

The Cost of Getting It Wrong

Organizations that adopt AI tools without methodology face compounding problems:

Quality Degradation: Junior developers accept AI suggestions without understanding the implications. Security vulnerabilities, performance issues, and architectural violations slip through. The codebase accumulates technical debt faster than before AI adoption.

Inconsistent Practices: Some team members use AI extensively. Others resist entirely. There is no shared language for discussing AI-assisted work, no agreed standards for when delegation is appropriate.

Missed Competitive Advantage: Your competitors are developing systematic approaches to AI-assisted development. Without methodology, you capture perhaps 10% of the potential productivity gain while assuming 100% of the risk.

Senior Engineer Frustration: Your most experienced engineers see AI tools as either a threat to be resisted or a toy that does not match their workflow. Neither perspective captures the actual opportunity.

The Framework: Delegate, Review, Own

Every task in AI-assisted development falls into one of three categories.

Delegate

- The AI executes. You provide clear specification, the AI produces output.
- Examples: boilerplate generation, test scaffolding, database migrations, documentation, log analysis, standard implementations.
- Senior engineers delegate more effectively because they write precise prompts with minimal ambiguity. The AI has less room for interpretation.

Review

- The AI proposes. You verify. You are the quality gate.
- Examples: evaluating AI-generated code for edge cases, security implications, architectural fit, and meaningful test coverage.
- Senior engineers review faster because they recognize correct patterns immediately. They spot problems in seconds that would require junior developers extended investigation.

Own

- The AI cannot make this decision. Human judgment is the value.
- Examples: product direction, architecture trade-offs, risk acceptance, team conventions, the standards that govern how AI operates in your codebase.
- The more effectively you own the right decisions, the more you can safely delegate everything else.

Applying the Framework Across Development Phases

Phase	**Delegate**	**Review**	**Own**
Plan	Draft specifications, map dependencies	Validate scope, verify estimates	Vision, trade-offs, approval
Design	Generate component skeletons	Check accessibility, performance	System design, architecture
Build	Infrastructure code, configurations	Audit security, cost implications	Architecture decisions
Test	Generate test suites	Reject shallow coverage	Test strategy, data approach
Review	Automated style and pattern checks	Verify intent, audit for drift	Standards, risk acceptance
Docum ent	Generate changelogs, technical docs	Verify accuracy	Documentation structure
Deploy	Execute deployments, monitor logs	Audit configurations	Release decisions, incident response

Why Self-Learning Is Not Sufficient

Engineering teams can learn AI tools independently. They watch tutorials. They experiment. They develop individual workflows.

The problem is inconsistency. Each developer develops their own approach. There is no shared vocabulary for discussing AI-assisted work. Best practices discovered by one engineer do not transfer to others. The organization captures a fraction of the potential value. Structured training provides:

I. Shared Methodology

Everyone speaks the same language. "Delegate this" and "you should own that decision" become actionable guidance rather than vague preferences.

II. Accelerated Learning

What takes months of individual experimentation takes days with structured instruction. Developers learn from patterns observed across many organizations, not just their own trial and error.

III. Senior Engineer Buy-In

Skeptical senior engineers often become advocates once they see that the methodology protects their judgment rather than replacing it. They delegate more confidently because the framework gives them clear criteria.

IV. Measurable Improvement

Organizations that implement structured AI training see 40% velocity improvements within the first month. Team-wide AI adoption rises from approximately 50% to over 95%.

Six Actions High-Performing Teams Take

I. Establish Codebase Documentation for AI

Create documentation that helps AI tools understand your codebase. This includes project conventions, testing approaches, common patterns, and known quirks. When AI tools have context, they produce more accurate output.

II. Write Implementation Plans Before Prompting

Successful teams plan before they prompt. They outline what they want to build, break it into discrete tasks, and work through each task systematically. This prevents scope creep and produces cleaner implementations.

III. Reset Early When AI Output Goes Wrong

When AI produces incorrect output, the instinct is to keep prompting corrections. High-performing teams do the opposite: they discard the failed attempt and start fresh with clearer specification. Stacking corrections creates fragile code.

IV. Decompose Work Into Small Units

AI tools perform best on focused, well-defined tasks. Asking for entire features produces sprawling, incorrect output. Asking for specific components produces accurate implementations.

V. Create Verification Checklists for Junior Developers

Junior developers cannot verify AI output by instinct. Provide explicit checks: no hardcoded credentials, error states handled, tests cover failure cases, permissions follow least-privilege principles.

VI. Protect Core Engineering Skills

Some capabilities should not be delegated: problem decomposition, debugging intuition, system design, trade-off analysis. Allow AI to handle execution while maintaining the skills that require human judgment.

What Happens When Teams Apply This

We have trained engineering teams at Fortune 100 companies, DAX 40 enterprises, and high-growth technology firms. Consistently, we observe:

- 40% velocity improvement within the first month
- Team-wide AI adoption increasing from approximately 50% to over 95%
- Cleaner codebases with enforced standards
- Fewer review cycles and reduced back-and-forth
- Faster onboarding for new team members
- Senior engineers moving from skepticism to advocacy

One financial services team reduced their average PR cycle time from 4 days to 1.5 days. A SaaS company shipped a major feature in 3 weeks that had been estimated at 8 weeks. A consulting firm standardized AI practices across 200+ engineers in under a month.

What Training Covers

This white paper provides the framework. In training, your team receives implementation.

I. Tool Proficiency

Cursor (Tab completion, Composer, multi-file context, MCP servers), Claude Code (terminal-native workflows), and Codex (background agents at scale). We cover the tools your team uses or plans to adopt.

II. Infrastructure Development

Setting up documentation that works across AI tools. Configuring integrations for your existing systems. Creating templates your team can reuse.

III. Judgment Development

When to delegate versus own. How to verify AI output efficiently. How to develop junior engineers without creating dependency.

IV. Application to Your Codebase

We work on your actual code, under NDA. Your team applies the framework to real work and leaves with internal playbooks.

V. Cross-Functional Coverage

Training extends beyond developers to include designers, QA engineers, and DevOps. AI-assisted development is a team capability, not just an individual skill.

Training Formats

- **On-Site**

We come to your office, anywhere in the world. Minimum two trainers. Custom curriculum based on your technology stack and challenges.

- **Offsite Intensive**

Multi-day bootcamp at premium locations. Maximum 12 participants. Immersive learning environment with all accommodations included.

- **Online Sessions**

Remote training for distributed teams. Same methodology, flexible scheduling.

- **Ongoing Support**

For teams that want it, we provide continued guidance as you implement what you have learned.

About Cursor Workshop

Rogier Muller, Co-founder

CTO at BlueMonks Group and founder of delta0. Cursor Ambassador and Anthropic Community Lead in Amsterdam.

Vasilis Tsolis, Co-founder

Founder of Cognitive+ with deep expertise in agentic workflows and enterprise engineering practices.

Our workshops consistently rate above 4.8/5. We have trained teams across financial services, SaaS, and professional services organizations.

Next Step

We work with a limited number of teams per month. If your organization is evaluating how to systematically improve AI-assisted development, we should talk.

We will discuss your current state, identify specific opportunities, and outline how structured training could accelerate your team.

Book a Strategy Session:

cursorworkshop.com/contact

Trusted by Fortune 100 and DAX 40 engineering teams.