The Agentic Coding Cheat Sheet

How teams actually ship faster with AI

CURSOR WORKSHOP

cursorworkshop.com

Contents

Why Most Teams Fail with AI Coding Tools

Organizations adopt AI coding assistants expecting immediate productivity gains. The reality is more complex. Without a framework for when and how to use these tools, teams often see marginal improvements at best and quality regressions at worst.

The data tells a clear story

Jellyfish analyzed engineering velocity across thousands of developers using AI coding tools. Senior engineers completed tasks 22% faster. Junior engineers? Just 4% faster. Same tools, same prompts, dramatically different outcomes. The gap has nothing to do with the AI and everything to do with the human using it.

AI amplifies existing capability

Think of AI coding tools as a multiplier, not a replacement. A senior engineer rated at "10" becomes a "15" with proper AI assistance. A junior rated at "3" becomes a "3.2" at best. The tool accelerates what you already know how to do. It cannot substitute for judgment you have not developed.

This creates a paradox: the engineers who benefit most from AI assistance are those who need it least. Meanwhile, junior developers generate plausible-looking code they lack the experience to verify. Technical debt accumulates. Review cycles lengthen. The promised efficiency gains evaporate.

Three failure modes we observe repeatedly

Over-delegation. Teams hand complex architectural decisions to AI tools expecting coherent solutions. The AI produces code that compiles but makes poor trade-offs invisible to reviewers unfamiliar with the domain.

Under-delegation. Experienced engineers refuse to use AI for routine tasks where it excels, manually writing boilerplate and losing hours to work that carries no learning value.

No verification protocol. Generated code enters the codebase without systematic review. Edge cases go unhandled. Security vulnerabilities slip through. The codebase degrades incrementally until a major incident forces a reckoning.

The solution is a framework

High-performing teams share one characteristic: they have explicit rules about what AI handles, what humans handle, and how handoffs work. The following pages describe that framework.

Three Zones: Delegate, Review, Own

Every task in software development falls into one of three categories based on the appropriate level of AI involvement. Misclassifying tasks is the primary source of AI-related quality problems.

Zone 1: Delegate to AI

The AI executes the task independently. You provide a specification, the AI delivers a result, you integrate it with minimal modification.

Characteristics: Well-defined inputs and outputs, low ambiguity, established patterns, reversible if wrong.

Examples:

- Generating boilerplate code from templates
- Writing test scaffolds for existing functions
- Creating database migrations from schema changes
- Converting documentation between formats
- Analyzing logs for known error patterns

Why seniors excel: They write prompts with precise specifications that leave no room for interpretation. Ambiguity in prompts produces ambiguity in output.

Zone 2: Review AI Output

The AI proposes solutions. You evaluate, modify, and approve. Human judgment is the quality gate that determines what ships.

Characteristics: Multiple valid approaches, context-dependent trade-offs, consequences for errors, requires domain knowledge to evaluate.

Examples:

- Evaluating generated code for edge cases
- Checking security implications of suggested changes
- Assessing architectural fit of proposed solutions
- Verifying performance characteristics
- Validating business logic correctness

Why seniors excel: They have mental models for what "correct" looks like. They spot deviations quickly because they have seen the patterns before.

Zone 3: Human Ownership Required

AI cannot make these decisions because they require judgment that emerges from experience, organizational context, or accountability that cannot be delegated.

Characteristics: High stakes, irreversible consequences, requires understanding of business context, involves trade-offs without objectively correct answers.

Examples: Product vision and roadmap priorities. Architecture decisions with long-term implications. Security risk acceptance. Team process and culture. The AGENTS.md file that governs how AI operates in your repository. Hiring and performance evaluations.

The governance paradox

You must own the rules that govern AI behavior. If AI writes its own rules, you have abdicated responsibility for outcomes you will still be accountable for when things go wrong.

Seven Phases of the Development Lifecycle

The three-zone framework applies differently at each stage of development. This matrix provides concrete guidance for where AI adds value and where human judgment remains essential.

PHASE	DELEGATE TO AI	REVIEW AI OUTPUT	HUMAN OWNERSHIP
Plan	Draft planning documents, map dependencies between tasks, slice user stories into implementable units, estimate complexity	Validate that scope is realistic, check that estimates account for unknowns, verify dependency ordering	Product vision, prioritization decisions, resource allocation, timeline commitments to stakeholders
Design	Generate component structures, suggest implementation patterns, create interface skeletons, draft design documentation	Evaluate accessibility compliance, assess performance implications, verify consistency with design system	Design system decisions, CI/CD pipeline architecture, tool selection, coding standards
Build	Generate infrastructure-as-code, create Dockerfiles and container configs, scaffold CI/CD workflows, write deployment scripts	Audit for exposed secrets, verify IAM permissions follow least-privilege, review cost implications of resource choices	Infrastructure architecture, security policies, disaster recovery strategy, vendor selection
Test	Generate test cases from specifications, create Playwright/Vitest scaffolds, produce mock data, suggest edge cases to cover	Reject tests that over-mock and test implementation details, verify tests actually assert meaningful behavior	Testing strategy, test data management, coverage requirements, performance benchmarks
Review	Automated code review via AGENTS.md rules, style checking, complexity analysis, dependency auditing	Verify that changes match stated intent, check for drift from architectural decisions, assess impact on other systems	Review process design, AGENTS.md rule creation and maintenance, merge authority, exception handling
Document	Generate API documentation, create changelog entries, draft README updates, produce inline code comments	Verify documentation accuracy against implementation, ensure examples work, check for outdated information	Documentation structure, update policies, public API contracts, deprecation procedures
Deploy	Execute infrastructure-as-code, monitor deployment logs, run smoke tests, generate rollback procedures	Final secrets audit, billing impact review, verify feature flags are correctly configured	Go/no-go decisions, on-call responsibility, incident response, feature flag ownership

04 IMPLEMENTATION

Four Things You Can Implement Today

These practices require no organizational approval, no budget, and no process changes. Any developer can start using them immediately. Each addresses a specific failure mode we observe in teams adopting AI tools.

1. Create an AGENTS.md file in your repository

AGENTS.md is the emerging standard for providing AI agents with context about your codebase. It functions as a README specifically for AI tools, explaining project structure, coding conventions, and common pitfalls.

Over 60,000 open-source repositories now include this file. Major AI coding tools have adopted it as a convention: Claude Code, OpenAI Codex, Cursor, GitHub Copilot, Devin, Aider, Jules, and Zed all read AGENTS.md when present. Write the file once and every AI tool follows the same rules.

A minimal AGENTS.md includes: project purpose in one paragraph, directory structure explanation, build and test commands, coding conventions (naming, formatting, patterns to follow), and common mistakes to avoid.

2. Maintain a CHANGELOG.md for agent context persistence

AI agents lose context between sessions. Every new conversation starts from zero. This creates inefficiency as you re-explain project state and recent decisions.

A CHANGELOG.md solves this. After each significant change, document: what changed, why it changed, which files were modified, and any follow-up tasks. When you start a new AI session, the agent reads the changelog and resumes with full context. This eliminates the "cold start" problem that wastes the first 10 minutes of every session.

3. Decompose work into agent-compatible units

AI tools perform best on well-scoped tasks with clear success criteria. Large, ambiguous requests produce inconsistent results.

> INEFFECTIVE PROMPT
> "Create a user authentication system with OAuth support"
>
> EFFECTIVE PROMPT
> "Create a JWT validation middleware that extracts user_id from the Authorization header, validates the signature against our JWKS endpoint, and returns 401 with error code AUTH_INVALID_TOKEN for malformed or expired tokens"

The second prompt succeeds because it specifies inputs, outputs, error handling, and success criteria. There is no room for interpretation.

4. Give junior developers verification checklists

Junior engineers generate code confidently but lack heuristics for spotting problems. Provide explicit checklists they must complete before marking AI-generated code as ready for review:

- No hardcoded secrets, API keys, or credentials anywhere in the diff
- All error states have explicit handling with user-facing messages
- Tests cover at least one failure case, not just the happy path
- Permissions follow least-privilege principle
- Any new dependencies are necessary and from trusted sources

Measured Results from Enterprise Teams

These figures come from teams we have trained directly, measured before and after implementing the framework described in this document. Results vary by team composition and starting point, but the patterns are consistent.

40%	50%	4.8/5
INCREASE IN DEVELOPMENT VELOCITY	REDUCTION IN REVIEW CYCLES	AVERAGE WORKSHOP RATING

Velocity improvement analysis

The 40% velocity increase comes primarily from three sources. First, delegation of routine tasks that previously consumed 2-3 hours daily. Second, faster context-switching enabled by AGENTS.md and CHANGELOG.md reducing ramp-up time. Third, reduced back-and-forth in code review as AI-generated code arrives with better initial quality.

Teams see the largest gains in the first month as they eliminate the most obvious inefficiencies. Improvements continue but flatten as teams approach their efficiency frontier.

Review cycle reduction

The 50% reduction in review cycles reflects two changes. Junior developers submit higher-quality initial code because checklists catch common issues before review. Senior developers spend less time explaining basic problems because AGENTS.md enforces conventions automatically.

The experience gap narrows but does not close

After training, junior engineers show the largest percentage improvement in AI tool effectiveness. The gap between senior and junior performance narrows from 18 percentage points to approximately 8 percentage points. However, seniors still outperform because judgment and pattern recognition take years to develop regardless of tooling.

What we observe in teams that struggle

Not every team sees these results. Common patterns in teams that fail to improve:

- No designated owner for AGENTS.md maintenance, leading to drift
- Checklists exist but are not enforced in the review process
- Management expects AI to reduce headcount rather than increase throughput
- Engineers view AI as threatening rather than augmenting their work

Tooling changes fail without cultural alignment. Teams must agree on what AI is for before adoption can succeed.

Training Programs and Curriculum

This cheat sheet covers the conceptual framework. Our training programs go deeper, with hands-on exercises on your actual codebase and direct coaching from practitioners who use these tools daily.

Tool mastery

We cover the major AI coding tools in depth, with emphasis on features most teams underutilize:

- **Cursor:** Tab completion, Composer for multi-file edits, context management, MCP server configuration
- **Claude Code:** Terminal-native workflows, agentic task execution, context window optimization
- **Codex:** Background agents for parallel task execution, orchestration patterns

Most teams use 20% of available features. Training focuses on the capabilities that produce the largest efficiency gains for your specific workflow.

Infrastructure development

Teams leave with working infrastructure, not just knowledge:

- An AGENTS.md file tailored to your codebase and conventions
- MCP server configurations for your tools (GitHub, databases, cloud providers, design tools)
- Prompt templates your team can reuse for common tasks
- Verification checklists calibrated to your risk tolerance

Judgment development

The hardest skill to teach is knowing when to trust AI output and when to be skeptical. We use case studies from real incidents to develop this intuition:

- Recognizing hallucination patterns in generated code
- Identifying security vulnerabilities AI tools commonly introduce
- Spotting architectural drift before it becomes technical debt
- Training junior engineers without creating AI dependency

Program formats

On-site training (1-2 days): Intensive workshops at your office with your team and your codebase. Best for teams of 8-20 engineers who want immediate, practical results.

Executive offsite (2-3 days): Strategic sessions for engineering leadership covering organizational adoption, change management, and metric design alongside hands-on technical content.

Enterprise program (ongoing): Multi-month engagement including training, coaching, and regular check-ins. For organizations rolling out AI tools across multiple teams.

07 ABOUT

Cursor Workshop

We train engineering teams to use AI coding tools effectively. Our clients range from Fortune 100 companies and top 50 EU enterprises to high-growth startups and smaller engineering teams. Every sector, domain, and team size can benefit from systematic AI adoption. Workshop ratings consistently exceed 4.8 out of 5.

Our approach

We do not teach theory divorced from practice. Every concept in our curriculum is demonstrated on real code, tested in production environments, and refined based on feedback from hundreds of engineers.

We stay current. AI coding tools evolve weekly. Our curriculum updates continuously to reflect new capabilities, changed best practices, and lessons learned from recent client engagements.

We measure results. Before and after metrics are part of every engagement. If velocity does not improve, we want to know why and adjust our approach.

Founders

Rogier Muller

Founder of delta0 (B2B ML agency). CTO at BlueMonks Group (fintech/regtech). 15 years in production ML. Ships daily with agentic tooling.

Vasilis Tsolis

Founder of Cognitiv+ (AI document intelligence). Scaled engineering orgs in fintech and legaltech. Deploys agentic systems in production environments.

Explore Training Programs

On-site workshops. Executive offsites. Enterprise programs.

cursorworkshop.com/training

CONTACT

info@cursorworkshop.com