# Engineering Effectively with AI Tools

Six days of material, offered as three independent two-day courses. Each is written for a different level of working experience with agentic coding tools, and a short placement questionnaire routes each engineer to the right one.

## Why three courses rather than one

Any room assembled by job title will contain a five-year spread of relevant experience, and that range is not a problem of pace — it is a problem of *subject*. An engineer three weeks into working this way needs an accurate mental model of what the machine is actually doing. An engineer eighteen months in has built that model from experience and needs something else entirely: how to widen autonomy without widening blast radius, how to run more than one agent without losing track of what changed, and how to turn a personal practice into something a team can rely on.

Delivered to one room, the first group is over-extended and the second is told things it learned the hard way last year — which is the most common failure of training in this subject, and it does not look like failure at the time. The feedback it produces is *"this was good, and it would have helped more a while ago."*

Three courses let each open at the altitude its audience is standing at.

**Two days each, four half-day sessions per course.** The tracks are cumulative in subject and independently deliverable: a team can run one, two, or all three, in any order that matches who needs what. Sessions are designed for online delivery with the exercises solo and the discussion in chat, and translate to a room without change.

## Everyone works in the same codebase

Every exercise in all three tracks runs against a repository we supply, not against the participant's own work. This is a deliberate reversal of the obvious approach, and it is worth stating plainly because it is what makes the exercises reliable.

Asking engineers to bring real work fails in five predictable ways. Many will not bring anything. Some are not permitted to — the code is restricted, or the client is. Those who do bring something frequently cannot get it running inside a session, because the build needs credentials, a VPN, or twenty minutes of dependency resolution. Those who get it running often cannot discuss what they found, so the debrief —

where most of the learning is — collapses into generalities. And the moments the course is built around stop being reliable: a planted defect produces the same realization in every room, while a participant's own repository produces whatever it happens to contain.

So the codebase is provided, it is small, it installs with one command and no external services, and it is seeded with defects and design decisions chosen to produce specific realizations at specific moments. Nothing in it is labeled as an exercise. Participants apply the same techniques to their own work between sessions, where the stakes are real and the discussion is theirs.

**Two repositories across the three tracks.** Tracks A and B use a scholarly-collections service — small, readable, conventionally built, with no tooling around it. Track C uses a second, larger repository of the same fictional product: several packages sharing a common library, with the continuous-integration configuration, written conventions, agent guidance and operational runbook that the first one deliberately lacks. Track C's exercises are about process and governance, so its repository is shallow in code and deep in process surface — the inverse of the first.

---

## Track A · Foundations

2 DAYS | FOUR HALF-DAYS | NO PREREQUISITES

**For engineers whose working model of these tools is either "magic" or "autocomplete."** Both are wrong in ways that produce predictable, expensive mistakes, and neither corrects itself with casual use.

**What participants leave able to do.** Predict where an agent will be reliable and where it will not, before running it. Write an instruction that survives contact with a real codebase. Keep a session recoverable. Review generated code without mistaking fluency for correctness. And judge honestly whether their own capability is being built or quietly spent.

### THE SESSIONS

**1 · How the Machine Sees.** How retrieval actually works, and why it finds things that share no words with the question. An honest capability map: genuine strengths, genuine weaknesses, and why hallucination is the mechanism working as designed rather than a defect awaiting a patch. The asymmetry that governs everything downstream — a hedge is a signal, and its absence is not.

*Exercises: Find the Neighbors · The Calibration Bet.*

**2 · The Loop and the Window.** What wraps the model and turns it into an agent: the observe-act loop, why it self-corrects, and why it drifts. Context as an economy rather than a budget. What a tool chain can touch, and what bounds it. Includes a practical grounding in starting and running a session cleanly — permissions, scope, and what the tool can reach.

*Exercises: Trace Autopsy · Where Did the Tokens Go · Prompt Rewrite Ladder.*

**3 · Saying What You Mean.** The step up from prompting to specification. The four kinds of constraint that live in a team's head rather than its code, and why a codebase under-determines its own conventions. Capturing that knowledge somewhere the agent reads. Checkpointing, and making abandonment cheap enough to actually do.

*Exercises: Adversarial Ambiguity Hunt · Build the Starter Kit.*

**4 · Staying an Engineer.** Reviewing generated code when the usual tells no longer fire. Verification habits that do not depend on trusting the output. Then an extended block of work with no agent and no autocomplete, measured against a baseline taken on the first morning — because the erosion this course warns about feels exactly like productivity while it is happening, and introspection is unreliable about it.

*Exercises: The PR review pass · The Unplugged Hour.*

---

## Track B · Practitioner

2 DAYS | FOUR HALF-DAYS | DAILY WORKING USE ASSUMED

**For engineers who use these tools every day and have been burned at least once.** They do not need convincing that the tools work, and they do not need convincing that the tools fail. They need the discipline that turns both facts into reliable practice.

**What participants leave able to do.** Decide what to delegate based on the terrain they are working in rather than on how the task feels. Write a specification an agent cannot quietly reinterpret. Recognize a session going wrong early enough to stop it. Review at volume without the review becoming theater. And tell the difference between a control that is enforced and a control that is evidence.

### THE SESSIONS

**1 · Foundations, Compressed.** The machine model, the agent loop, and context economy in one session, pitched for people with working familiarity. Recalibration against measured results rather than folklore — most engineers' model of how these tools fail was formed two or three years ago and nothing has forced an update since.

*Exercise: The Calibration Bet.*

**2 · Naming the Work.** Not one codebase — heritage, greenfield, acquisitions and modern platforms are different terrains with different hazards, and how much can safely be delegated is a property of the codebase's feedback signals as much as of the engineer's judgment. Using an agent for archaeology on unfamiliar systems. Specification as the skill the tools make visible. Decomposition, testability, and what makes acceptance criteria bite.

*Exercises: Two-Tool Bake-Off · Spec, Then Check — both against a supplied feature brief rather than work brought from outside.*

**3 · Directing and Controlling.** Making tacit constraints explicit. The ambiguity you are structurally unable to see in your own prose. Workflow patterns for real work rather than single prompts. Checkpointing, scoping, and intervening in a session that has gone wrong — including the two symmetrical failures of intervening too early and too late.

*Exercises: The Intervention Drill · Build the Starter Kit.*

**4 · Review and Judgment.** The supervision paradox in its current form: the finding got cheap and the deciding did not. Why the old heuristics fail when generated code reads well by construction. The gate ladder, and the difference between enforcement strength and independence of evidence — a required merge gate whose tests came from the same process as the code is a mirror, not a control. Named failure modes and the earliest point each is detectable.

Includes a short block on **why a passing build is sometimes not one**. An agent reports what the shell tells it, and the shell is frequently reporting something other than what was asked: a pipeline's exit status is the last command's, so a check piped into anything readable reports the reader's success rather than the check's. Related habits of the same shape — blocks that read as guarded and are not — are the most common way a confident "everything passes" turns out to be false. Both are cheap to check and almost nobody in the room will have checked.

*Exercises: The PR Gauntlet · Failure Mode Bingo · The Unplugged Hour.*

---

## Track C · Autonomy and Orchestration

2 DAYS | FOUR HALF-DAYS | GATED — SEE PLACEMENT

**For engineers and leads who already have review discipline and want to widen what they hand over.** This course does not relitigate whether these tools hallucinate. It starts from the position that the participant is already delegating successfully and asks the harder question: on what evidence should they delegate *more*, and what has to be true of the environment first.

**What participants leave able to do.** State the conditions under which a class of work earns more autonomy, and the controls that make that safe. Convert the rules their team currently writes down into mechanisms that cannot be skipped. Run more than one agent on related work without losing accountability for what changed. Reconstruct, after the fact, the intent behind a line in production. Write governance their team will actually follow rather than route around. And answer whether their organization would know if its engineers' judgment were eroding.

### THE SESSIONS

**1 · Earning Autonomy and the Shape of a Gate.** Autonomy as something granted against evidence rather than assumed or withheld. Two independent axes — how hard a control is to bypass, and how independent its evidence is from the thing being verified — and why the most dangerous position is strong enforcement with no independence. Blast radius and reversibility. Capability boundaries,

attenuation, and isolated execution, including the strongest form of a boundary: one where a restricted principal does not merely get refused but *cannot see* the action at all, so the attempt is never made and the refusal never has to be explained.

Then the discipline that makes any of it real — **knowing what your gate does not cover.** The recurring shape of nearly every failure in this area is a gate that silently covers less than it appears to: a suite that tests one layer and is blind to the one below it, a version pin that quietly ceilings a team out of the current definition of "done" while still reporting green. Worked through with real examples in which the instrument was wrong for years.

Closes on the promotion path: naming in advance the evidence that would move a class of work from *approve the plan* to *just do it*, which is what separates governance from a ratchet that only tightens.

*Exercise: Place the repository's seven real controls on the grid — a CI workflow, a linter, a test suite, a review rule — then find the ones that are not in the quadrant they appear to be in. Applying the same pass to your own team's controls is the take-home.*

**2 · Mechanism Over Intention.** The organizing principle of the track: **a mechanically-checkable rule belongs in a hook, a procedure belongs in a documented skill, and a role belongs in an agent definition.** Anything left in prose is advisory, and it will be skipped — not through carelessness, but because momentum is exactly what makes an agent productive, and prose does not survive momentum three hours into a task.

Expressing "done" as an executable check rather than a paragraph, and what happens when you do: conversions of this kind routinely surface defects that shipped for years. Gates that return a trustworthy exit status, and the specific ways a shell reports success that was never achieved. Durable state that lives outside any context window, because a session's own report of what it did is not evidence.

And the constraint that governs guardrail design: **a guardrail the rightful owner has to evade is a failed guardrail.** A control that blocks legitimate work does not fail safe — it teaches evasion, and an agent is extremely good at evasion. Fence the narrow destructive act rather than the neighborhood, never fence reads, fail open on uncertainty, and tell the agent why the wall is there.

*Exercise: The repository's contributing guide states eight conventions in prose and enforces none of them. Convert three into mechanisms, and identify the ones that cannot be converted — including the rule that looks mechanizable and whose mechanism would give false assurance.*

**3 · Orchestration.** Running multiple agents honestly. Decomposition for parallelism, and the work that resists it. Ownership as a durable record rather than an intention, and what it takes to hold concurrent sessions accountable to each other: exclusive claims on a working tree, first-writer- wins with expiry, surgical rather than wholesale edits, and identity attached to a claim. The limits of any such scheme — in particular what happens when two workers share an identity — because a fence you believe covers more than it does is worse than none.

Work products that outlive the session that produced them, and why the transcript of an expensive piece of work is an archive rather than a cache. The mental cost of supervising concurrent work, which is real and is the reason most orchestration quietly fails. Explicit treatment of when *not* to orchestrate.

*Exercise: One feature brief against a multi-package repository, decomposed for three agents working at once. Find the collisions before dispatching — they are planted, and at least one of them is the choice between duplicating a shared helper and modifying it underneath somebody else.*

**4 · Governance, Measurement and the People.** Capability and trust as separate questions: what the agent can demonstrably do, and what it should therefore be allowed to do. Delegation, safety, accountability and ownership as the four things a team charter has to answer, written against a repository the participants actually work in and then stress-tested against the people who would have to live with it.

Provenance: getting from a line in production back to the intent, the plan and the person who approved it. Treated concretely — when a commit, a pull request, a review finding, a work item and a human annotation share one identity scheme, questions that no single-purpose tool can answer become a single query, and the reviewer arrives with context already assembled instead of spending the first ten minutes building it. This is the most direct available answer to reviewing at volume: **review gets faster by removing things from the reviewer, not by reading faster.**

Then measurement — output versus capability, and instruments that do not rely on self-report — and the documentation failure this creates, where plausible claims survive retelling because nobody re-read the source.

Closes on the part that is nobody else's job: how people learn this trade now. Onboarding that preserves productive struggle, explain-back as an acceptance gate, fading scaffolds, and rotation with a named schedule rather than good intentions. The organizational form of the competence question rather than the personal one.

*Exercises: The Team Agent Charter, written against the supplied repository — which ships with agent guidance that is stale and self-contradictory, and a runbook naming one person for everything · Design the first ninety days for an engineer joining that codebase, using its out-of-date onboarding document and the gaps in its decision records.*

> **A note on what this displaces.** The long-range material — how the economics shift as the cost of producing a change keeps falling, and which of today's bets stay sound — is delivered as a closing argument rather than a section of its own. It is the most interesting content in the track and the least actionable on Monday, and this audience is paying for the latter.

---

## Placement

Route on **demonstrated practice, not job title or tenure.** Experience with agentic tools correlates only loosely with seniority: engineers with twenty years behind them are frequently new to this way of working, and engineers three years in are sometimes the most advanced practitioners in the building. Gating on grade will misplace people in both directions, and the resulting mismatch is what produces the *"this would have been more useful earlier"* response.

Four questions, answered by the participant at registration.

| | Question |
|---|---|
| **1** | Is there a rules or context file — `CLAUDE.md`, `AGENTS.md`, or equivalent — in a repository you own? |
| **2** | Have you ever abandoned a session because the accumulated context had itself become the problem? |
| **3** | Do you routinely run more than one agent against related work? |
| **4** | Are you accountable for how other people on your team use these tools? |

**Routing.**

| Answers | Track |
|---|---|
| No to 1 | **A · Foundations** |
| Yes to 1, no to 2 | **A · Foundations**, or B with a caveat |
| Yes to 1 and 2, no to 3 and 4 | **B · Practitioner** |
| Yes to 3 **or** 4 | **C · Autonomy and Orchestration** |

**Question 4 overrides the others.** Someone accountable for a team's practice who answers no to 1 through 3 is a lead who needs governance, onboarding design and measurement — not calibration. They belong in Track C even though their personal hands-on practice may be lighter than a Track B participant's.

**Question 1 is the honest floor for Track C.** Anyone who cannot answer yes to it has not yet had to make their working conventions explicit, and the governance material will land as abstraction.

### SEQUENCING

The tracks are cumulative in subject but not strictly prerequisite in time. A Track A participant is well served by Track B six to twelve months later, once they have accumulated the experience that makes the discipline meaningful. Track B to Track C is the same interval. Running the full sequence back to back is not recommended and will not produce three courses' worth of change.

### A NOTE ON MEASUREMENT ACROSS TRACKS

Each course opens with a short, unframed diagnostic and returns to it at the end. In Tracks A and B this measures the participant's own judgment, and its value is personal and preventive. In Track C the same instrument is turned on the organization: not *could I still do this*, but *would we know if our people could not, and is there anything in our current tooling that could tell us*. Very few teams can answer the second half, and the question is the most durable thing the course can leave with them.

## Practical requirements

**Tool-agnostic throughout.** The material is about how these systems behave and how to work with them, not about one vendor's interface. Participants use whichever agentic coding tool they already have, and the exercises are written to produce comparable results across them. One exercise in Track B deliberately has everyone work in a tool that is *not* their default, because what transfers between tools and what does not is itself part of the subject.

**What participants need.** A machine they can install and run code on, a current agentic coding tool, and Node 20 or later. Nothing else — the supplied repositories have no external services, no credentials and no network dependencies, and install with a single command. No prior familiarity with the codebase is assumed or useful.

**What is supplied.** The exercise repositories; a participant handout for each session carrying the exercises, worked material and space to write; the diagnostic instrument and its answer sheets; and a facilitator key for each session with measured expectations, so a normal result can be told from a broken one while it is happening.

**Format.** Designed for online delivery and equally deliverable in a room. Exercises are solo, with discussion in chat or open floor; nothing depends on breakout rooms or pairing, which makes the sessions robust to partial attendance and to participants joining from different sites. Each half-day runs about four hours including two breaks.

## From participants

> "This was the best AI/SDD training offered to me to date."

> "Brian is a great communicator and after the class I understood aspects of interacting with AI agents that I didn't know that I didn't know."

> "The practical examples and hands-on approach were particularly helpful."

**Next steps.** This syllabus is a starting point. Courses are assembled per client — the emphasis, the examples and the length all move to fit the team in the room. To request more detail, or to schedule a discussion about running this course, get in touch at bosatsu.net/contact.html.

Brian Sletten · Bosatsu Consulting, Inc.

AI & LLMs · agentic coding · specification · code review · guardrails · orchestration · governance