# Training Enterprise Software Engineers to Use AI

## An Evidence-Graded Framework — What the Research Supports, What It Implies, and What Remains Untested

**A Hamilton Road Agility Technical White Paper**

| | |
|---|---|
| **Audience** | Engineering leaders, L&D / enablement teams, platform and DevEx orgs |
| **Author** | John Crider |
| **Organization** | Hamilton Road Agility |
| **Version / Date** | v0.4 (draft) · July 1, 2026 |
| **Classification** | Public |

> **Evidence-honesty stance.** This paper is deliberately conservative about what is *known*. Every load-bearing claim carries an evidence grade — `[Established]`, `[Implied]`, or `[Proposed]` (defined in §3) — so a reader can see exactly where the ground is firm and where we are extrapolating. The uncomfortable headline is that **the rigorous evidence base is thin**: there are, at the time of writing, only a handful of randomized trials in this space, none of which studied a *training program*. We build on the literature as it exists, not as we wish it were.

> **Disclosure & scope.** This paper assumes the reader's enterprise has some interest in engineers using AI to improve *some* aspect of software development, and that engineers use a mainstream general-purpose model (from Anthropic, Google, or OpenAI). Other models may fit the same framework; we do not single them out or compare them. One cited randomized study is first-party vendor research (Anthropic); it is flagged as such wherever used and graded down accordingly. This paper is about *pedagogy and program design*, not procurement, security architecture, or data governance — but because any training program routes real code through a third-party model, we flag the data-handling questions a director must clear with legal/privacy before launch (§9). Generative AI (Claude) was used in this paper's research and drafting under human direction; see the Declaration of AI Assistance.

---

## 1. Executive Summary

Enterprises are asking engineers to use AI. The instinct is to run training that teaches people to *use the tool* — prompt patterns, features, speed. The evidence says this instinct is aimed at the wrong target.

Three findings frame the problem. All three draw on the small body of randomized evidence, but they do not all carry equal weight — and this paper's discipline is to say which is which:

1. **How the tool is used matters more than whether it is used.** For learners, an AI that *withholds answers and gives hints* helped, while the same model *giving answers* made them

measurably worse on unaided work (a class-randomized trial — but in **high-school mathematics**, so `[Implied]` for coding). For working developers, using AI to *explain and reason* preserved understanding while *delegating* code generation eroded it (a randomized study — but **n=52, first-party vendor**, so `[Implied]`). The two point the same way — though note they test *different moments* (preventing skill *acquisition* in learners vs. preventing skill *erosion* in practitioners), so "consistent" here means in spirit, not the same hypothesis tested twice. Neither is `[Established]` for enterprise software work.

2. **Perceived productivity is an unreliable signal.** In the one field trial of experienced developers on code they knew well (**n=16**), AI made them **slower** while they believed it made them faster. `[Implied]` — *the strongest small, independent, non-vendor field RCT on this point, but n=16 and familiar code only*.
3. **The right design depends on the person and the task.** Support that helps a novice actively hinders an expert (the *expertise-reversal effect*), and AI *tends to widen the gap* between prepared and underprepared learners. `[Implied]` (strong theory + consistent observation).

Put together, these say the goal of AI training is **not** faster code generation. It is **judgment**: knowing when AI helps and when it hurts, using it in the mode that preserves skill, and *verifying* its output — which multiple studies identify as the scarce, teachable skill of the era.

**But first, a harder question the rest of this paper forces you to answer: is a training program even the right instrument — and if so, is now the right time?** The largest and most enterprise-representative trial to date (Cui et al., n=4,867) found a ~26% throughput gain from *tool access alone* — no training program attached. So the honest baseline a director is measured against is not zero; it is "what people get from access plus documentation." A program has to beat *that*. We therefore recommend treating training as an **instrumented bet, not a default**: name the specific failure mode you are buying down (skill erosion, insecure output, the widening novice gap), check whether a cheaper control would handle it, confirm you can measure against the access-only baseline — and if you cannot yet, **wait and instrument first** rather than launch blind. §7 Step 0 gives the decision test (including the "not yet" outcome); §10 gives the measurement.

This paper offers a **framework, not a curriculum**. It is organized around two axes — the learner's **expertise** and the **novelty of the task** to them — because those are the variables the evidence says determine the right design. Three enterprise cohorts (skilled new hires, veterans, and people being skilled up into engineering) appear as *profiles* within that framework, not as separate programs. Throughout, we mark what the research **establishes**, what it **implies**, and what is merely **proposed** — and we are explicit that most actionable program design is, today, implied or proposed rather than proven. The **reference blueprints in §8 are illustrative `[Proposed]` engineering** — their week-ranges and staffing ratios are design conventions to adapt, not numbers the evidence dictates.

**If you direct a learning program, here is how to use this paper.** The framework (§5) is a *decision tool*, not a reading assignment: you place each cohort on two axes and the program design follows. §7 turns that into a step-by-step **design procedure** — beginning with a *whether-to-train* decision and a prerequisite check, and including a bridge for defending a months-long setup to

leadership; §8 gives **reference blueprints** you can adapt, each with a **minimum-viable tier**; §9–§10 cover **running and measuring** the program so you can defend the spend without overclaiming; §11 lists the **anti-patterns** that reliably waste the budget. The evidence grades tell you which parts of your program are safe bets and which are experiments you must instrument.

---

## 2. Assumptions and Scope

This paper is written to three stated assumptions:

- **A1 — There is enterprise interest.** The organization wants engineers to use AI to improve some aspect of delivery. We take that as given and do not argue for or against it. (We do, in §7 and §9, address how to hold that interest without triggering the failure modes of a heavy-handed mandate.)
- **A2 — Mainstream models.** Engineers use a general-purpose frontier model (Anthropic, Google, or OpenAI). We treat these as a category. Other tools may align with this framework; we neither name nor rank them.
- **A3 — Enterprise context, general audience.** The reader is an engineering or enablement leader designing training for a population of engineers — not a single team, and not tied to any one company's stack.

**A note on tool *mode* — this matters more than model choice.** "Using AI" is not one behavior. Inline **autocomplete** (suggests as you type), **chat/explain** (a conversational assistant beside the editor), and **agentic** tools (that plan and edit across many files, or run in an autonomous loop) load a developer's judgment very differently, and the verification burden shifts from *per-line* to *per-PR* as you move toward agentic use. Most of the randomized evidence in §4 studied autocomplete or chat — with the important exception that METR's subjects worked in mixed modes on tooling (Cursor Pro) that *includes* an agent mode, closer to where enterprise tooling is heading than pure autocomplete, though not exclusively agentic (`[Proposed]`, and note the tension: the case where caution may matter most is also the one where the controlled evidence is thinnest, n=16). **§5.4 treats what "verify" means in each mode**, because a curriculum built only for autocomplete is a generation behind the tools many engineers are already being handed.

**In scope:** how to design training/enablement so engineers use AI in ways that *build* rather than erode capability, graded by what the evidence supports. **Out of scope:** which model to buy, tool configuration specifics, data-governance and security architecture (flagged where a program can't ignore them, but not solved here), and the separate architectural question of running autonomous agents in delivery (covered in a companion paper).

---

## 3. How to Read the Evidence in This Paper

The defining problem of this topic is a **volume-versus-rigor mismatch**: there is an enormous and fast-growing literature, but very little of it is causal. Position papers, vendor playbooks, and observational

studies outnumber controlled experiments by more than an order of magnitude, and the loudest claims (universal productivity gains) rest on the weakest designs (self-report, lab tasks, unfamiliar codebases) — the exact conditions later shown to *inflate* perceived benefit.

To keep the reader oriented, every load-bearing claim is graded:

| Grade | Meaning | What backs it |
|---|---|---|
| `[Established]` | Supported by a randomized or strong controlled study **in a relevant population**. | RCT / controlled experiment |
| `[Implied]` | The direction is well-supported, but applying it here requires **extrapolation** — a different domain or population, a small sample, first-party sourcing, or observational/theoretical evidence rather than a controlled test in our setting. | Analogical RCT, small or vendor RCT, observational field study, seminal theory |
| `[Proposed]` | Advocated in the literature or by us, with **no empirical validation** yet. | Position paper, practitioner essay, our own inference |

Two consequences of using this discipline honestly:

- **The strongest, most-cited advice in the industry is often `[Proposed]`.** Staged-adoption playbooks, AI-literacy curricula, "trust-but-verify" workflows, and champion programs are coherent and plausible — but, as of writing, untested by controlled study.
- **Almost nothing about a *training program* is `[Established]`.** The randomized evidence is about the *effect of AI use* on learning or performance. That constrains how you should design a program; it does not validate any program. Where we make program-level recommendations, they are `[Implied]` at best, and we say so.

**One caution about the grades themselves.** A grade reflects study design *and* how far the claim travels from where it was tested. An RCT in high-school math, an n=52 vendor study, or an n=16 field trial is not `[Established]` for enterprise software engineering, however clean its design — it is `[Implied]`, and we mark it so even when it is the best evidence we have. The strength of an `[Implied]` claim is not uniform; where it matters (a claim resting on n=16 versus n=4,867), we say which study carries it.

A short evidence appendix (§14) lists the underlying studies with their designs; full citations are in §16, and every cited source has been independently verified against its primary source.

---

## 4. What the Evidence Actually Establishes

This section is the honest floor: the small set of things that rest on controlled evidence, stated with their real limits.

### 4.1 The randomized trials

Four randomized trials anchor this field. They *look* like they conflict — and they can be **reconciled** by sorting them by *who* was studied and on *what kind of task*. Read that reconciliation carefully, though: **a 2×2 model has enough definitional freedom to sort any four data points into quadrants by construction.** What follows is a *consistency*, not a confirmation — the framework in §5 was drawn to fit these results, and its predictive value is untested (§5.1). The four also differ enormously in weight: one enrolled 4,867 developers, another 16. We keep that asymmetry visible rather than treating "four RCTs" as four equal votes.

- **AI raises throughput at enterprise scale — and, in a subgroup analysis, helped less-experienced developers most.** The largest trial to date — a pre-registered RCT across Microsoft, Accenture, and a Fortune 100 firm, **n=4,867** — found a **~26% increase in completed tasks** with an AI code-completion assistant (Cui et al., *Management Science* 2025). `[Established]` for that headline throughput effect. A **subgroup analysis** found less-experienced developers adopted more and gained more — but this was *not* the pre-registered primary endpoint, subgroup splits inflate false-positive rates, and it has not been replicated, so it is weaker than the headline number and should be leaned on lightly. *Limit: measures a tool-access intervention on task throughput — not training, and not code quality.*
- **Experienced developers are slowed on *familiar* code — and don't notice.** A field RCT (**n=16**) of experienced developers working in repositories they knew deeply — using AI in mixed modes (chat, Cursor's agent mode, autocomplete) — found AI **increased** task time by **~19%**, while the same developers believed it sped them up ~20% (METR 2025). `[Implied]` — *the strongest small, independent, non-vendor field RCT on expert slowdown (large effect, coherent theory), but n=16, open-source, and familiar mature codebases only; it says nothing about experts on unfamiliar work.*
- **Guarded beats unguarded for learners — but "guarded" is not "AI off," and it is not better than "AI off."** A class-randomized trial (**~1,000 students**) found an *unguarded* answer-giving model left students **17% worse on unaided exams**, while a *guarded* hints-only tutor built on the same model **performed about like the no-AI control** (Bastani et al., *PNAS* 2025). **Design, not presence, drove the outcome.** Read the result precisely: guarded ≈ no-AI, and both beat unguarded. The study shows guardrails *prevent harm*; it does **not** show a guarded tutor beats simply removing AI during the learning phase. `[Established]` in-domain → `[Implied]` for software (it is high-school **mathematics**).
- **Mode of use governs skill retention.** A randomized study of **52 developers** found *conceptual/explanatory* use preserved comprehension while *delegating* code generation eroded it, with **debugging hit hardest** (Shen & Tamkin / Anthropic 2026). `[Implied]` — *small n and first-*

*party vendor research; the direction is consistent with the deskilling literature (§4.3), but this single study cannot carry an `[Established]` grade.*

**The shape that matters.** The two "beneficial" results (Cui: throughput up; Bastani-guarded: learning *preserved* — harm prevented, not a gain) and the two "cautionary" ones (METR: experts slower on familiar code; Anthropic: delegation erodes skill) are **not** a contradiction — they are what an *expertise* × *task-familiarity* model predicts. AI's benefit is largest at **lower expertise on well-scoped work**; its slowdown and skill-erosion risks are largest for **experts on familiar, complex code**, or for **anyone who delegates rather than engages**. This is a genuinely useful organizing idea — but hold its epistemic status honestly: it is a **post-hoc reconciliation, not an independent confirmatory test**, built to fit four studies and not yet used to predict a new one. §5.1 states what would falsify it. What none of the four shows is a **training program** working — that remains unproven, and is the honest floor of this paper.

**And note what Cui does to the premise.** The +26% came from *access*, not a course. Any training program is therefore competing against a real, measured, cheaper baseline. That is not an argument against training — the **broader evidence in this paper** (the mode-of-use study on skill erosion, §4.2 on insecure output) shows access *alone* does not address skill erosion or insecure code — but access is the bar a program must clear, and it is why §7 opens with a decision test rather than a curriculum.

### 4.2 Strong, non-randomized findings we lean on

- **AI-generated code is frequently insecure.** Peer-reviewed repo analysis found ~25–30% of AI-generated snippets contained CWE-class weaknesses; a developer survey found most engineers do not consistently verify AI output. Treat the percentage as directional, not a fixed constant — it varies substantially by model generation, language, snippet type, and how "weakness" is operationalized. Together these establish the *need* for verification and security review, even though *training* it works is `[Proposed]`. `[Implied]`
- **AI widens the novice gap.** Observational (including eye-tracking) studies of novices found prepared students used AI as a planning tool while underprepared students formed an "illusion of competence" and fell further behind, exhibiting over-reliance patterns ("shepherding," "drifting"). `[Implied]`
- **Assessment of "write the code" is broken.** Frontier models pass standard introductory programming exams, so those instruments no longer measure a person's ability. `[Established]` (as a problem) — *the replacement assessments are `[Proposed]`.*
- **System-level delivery can degrade even as individuals feel faster.** Large-survey (DORA) and repo-mining (code-clone growth, falling refactoring) evidence associates AI adoption with rising instability and structural debt absent testing discipline. `[Implied]` (correlational).

### 4.3 The theory layer (why the above holds)

The controlled coding evidence is thin, but decades of learning science and human-automation research explain the mechanisms and travel well:

- **Expertise-reversal effect** — guidance that helps novices hinders experts; and it is *domain-specific*, not about a person's global seniority (an expert in one stack is a novice in your internal one). `[Implied]`
- **The worked-example effect and the generation effect are two phases of one sequence.** For genuine novices with no schema, studying *worked examples* builds it; *then* effortful unaided generation consolidates it. Handing a novice answers skips schema-building; but so does asking a true novice to generate from nothing — both are failure modes, at opposite ends. `[Implied]`
- **Retrieval practice (the testing effect) and spacing.** Unaided recall *strengthens* memory, independent of its use as assessment — so unaided checkpoints are learning events, not just measurements, and spacing them out compounds the effect. `[Implied]`
- **Automation bias & complacency; trust miscalibration** — over-reliance is *not* eliminated by expertise, and worsens under workload; calibrated trust comes from structured exposure to a system's *actual* failure profile, not from a lecture about bias. `[Implied]`
- **Deskilling** — making a skill effortless erodes it (evidence from aviation, navigation, and clinical decision support; the closest *coding* signal is the mode-of-use study's debugging drop). `[Implied]`
- **Metacognitive miscalibration** — people (especially the less skilled) overestimate their competence, and AI removes the error signals that would correct them; making the loop explicit — **predict** your unaided score → **attempt** → **evaluate** against the prediction — is a cheap, validated counter. `[Implied]`
- **Motivation (Self-Determination Theory)** — mandates that undermine autonomy predict amotivation and worse performance; this is the mechanism behind the observed backfire of "use-AI" quotas (§9), and it tells you what to do instead: give rationale, genuine method-choice, and visible competence support. `[Implied]`

---

## 5. The Framework: Two Axes and a Set of Constants

The evidence supports a single design pattern applied at different **settings**, not a set of separate programs.

### 5.1 The two axes that set the design

Every learner-and-task situation sits somewhere on two axes, and the research says these are the variables that matter:

- **Axis 1 — Learner expertise** (novice → expert), *in the relevant domain*. Governs how much scaffolding helps vs. hinders (expertise reversal) and how vulnerable the learner is to the illusion of competence.
- **Axis 2 — Task familiarity** (novel → familiar *to this person*). Governs where AI genuinely helps vs. where it slows or deskills. This axis is why an expert on a *familiar* codebase (METR: slowdown, deskilling risk) and the same expert on an *unfamiliar* language (AI genuinely useful) need different

guidance — and why a skilled new hire learning your stack behaves, on a *new* concept, more like a novice than like a veteran.

> **The core insight:** you do not train "novices vs. seniors." You train *people on tasks*, and you set the AI dial by where that person-and-task sits on these two axes. The same senior engineer belongs in different quadrants on different days.

**Placing a learner on the axes** is a design step, not an afterthought — the framework is only as portable as your placement is consistent, so treat the placement instrument as a build/buy item (§7 Step 0.5). A workable rule: for expertise, ask *can they already build and ship this class of work unaided?* (no → novice; yes-but-not-this-domain → intermediate; yes → expert). For task-familiarity, ask *has this person done this specific concept/system before?* Use a short **unaided diagnostic** — e.g. an explain-this-code task and a find-the-bug task scored against a rubric, not a self-rating — because self-assessment is exactly what the evidence says is unreliable.

The four quadrants, and what the evidence implies for each:

| | Task novel to the person | Task familiar to the person |
|---|---|---|
| **Low expertise** | *Highest risk of hollow learning*. Worked examples first, then protect the unaided attempt; withhold/guard AI; verify understanding. Anchored by Bastani (guarded > unguarded, `[Implied]` for coding). | *(rare)* Reinforcement; guarded practice with fading support. **Untested prediction.** |
| **High expertise** | *Predicted sweet spot*. AI genuinely accelerates (new language/API); still verify output — and note the per-PR verification burden is *highest* on novel work, where the reviewer least knows what "correct" looks like. **`[Implied]` — inferred by negating METR, not directly observed.** | *Slowdown & deskilling zone*. Be selective; measure, don't self-assess; rotate manual practice. Anchored by METR (`[Implied]`, n=16 — mixed-mode use on agent-capable tooling, not the plain autocomplete/chat baseline the blueprints assume). |

Two of these four cells rest on direct randomized evidence; two are **predictions of the model, not observations** — the low-expertise/familiar cell and the high-expertise/novel "sweet spot" have no dedicated RCT and are the framework's own testable hypotheses. Treat program designs that live mostly in those cells (notably parts of the veteran and new-hire profiles) as more speculative than the low-expertise/novel cell, where the evidence is strongest.

**What would falsify this framework — and what to do if your own data disconfirms it.** State the falsifiers so the model can be tested, not just applied: if a well-run study found (a) experts *speeding up* on genuinely familiar, complex code with AI (familiar = they can already modify it unaided; complex

= multi-file, non-trivial); or (b) novices learning *as durably* — measured by unaided performance several weeks later, on transfer tasks in the same domain — from unguarded answer-giving AI as from guarded/worked-example sequences; or (c) task-familiarity showing *no* interaction with AI benefit once expertise is controlled — the two-axis model would be wrong or incomplete. Be honest about the horizon: criterion (c) needs a large factorial (expertise × familiarity) RCT that, to our knowledge, no one is currently running, so the framework is not falsifiable on the timescale you are deploying it — which is a reason to hold it loosely, not tightly. And guard against the framework absorbing every result: if programs built on it show lift but *without* the predicted quadrant pattern (e.g. novices and experts gain equally), treat that as evidence the *model* is wrong even though the program "worked," and say so in your report rather than re-classifying cohorts until the data fits.

### 5.2 The constants (invariant principles, graded)

Regardless of quadrant, the same principles hold; only their *strength* varies:

1. **Design and mode-of-use over mere usage.** Prefer guarded tools for learners and *explain/verify* over *delegate* for practitioners. `[Implied]` (the two supporting RCTs are cross-domain (Bastani, math) and small/first-party (Anthropic, n=52); the direction is consistent but not `[Established]` for enterprise coding).
2. **Sequence support to the learner's schema: worked example → unaided attempt → AI critique.** For a *genuine novice* on a new concept, provide a worked example first (worked-example effect); once a schema exists, protect the unaided attempt before augmenting (generation effect). One caveat on the third step: a novice cannot yet evaluate an AI critique, so **the critique must be checked against the worked example or by an instructor/test suite before the learner acts on it** — otherwise "get critiques from AI" quietly reproduces the over-reliance that "get answers from AI" was meant to avoid. `[Implied]`
3. **Fade support on *competence*, not on the calendar — against a pre-specified criterion.** Introduce/loosen AI when the learner demonstrates mastery, not when a phase's weeks elapse. This is only a real mechanism if "mastery" has a number: set a **criterion-referenced threshold in advance** (a sensible default is ≥80% on an unaided task specified in the module design; teams may adjust it but must pre-specify it). Without a pre-set criterion, competence-based fading collapses back into instructor intuition — time-based fading by another name. `[Implied]`
4. **Make verification — including security review — the taught skill.** The need is `[Implied]`; that training it works is `[Proposed]`.
5. **Treat subjective speed/competence as unreliable; measure outcomes.** `[Implied]` (the sharpest single data point, METR's ~39-point perception gap, is n=16; the direction is reinforced by the metacognition literature).
6. **Calibrate trust through exposure, not exhortation** — teach automation bias and the perception gap *by* structured contact with AI's real failure modes (e.g., the stack-reality lab, §8). `[Implied]`
7. **Guard against deskilling** — deliberate AI-off practice for skills you want to keep. The *risk* is `[Implied]`; the *specific intervention* (scheduled AI-off rotations) is `[Proposed]` and, as §9 notes,

politically fraught under an adoption mandate.

8. **Shift assessment away from "produce the code"** toward explaining, evaluating, and debugging it — and treat unaided checkpoints as *learning events* (retrieval practice), not only measurements. Problem is `[Established]`; replacements are `[Proposed]`.

### 5.3 The design levers (what a program actually turns)

A concrete program sets these knobs according to the quadrant(s) it serves. A useful lens for *which way* to turn them: does making AI available here reduce **extraneous** load (searching syntax — good to offload) or **germane** load (the effortful schema-building the task exists to produce — bad to offload)? The answer flips by quadrant, and it is the reasoning behind every lever below.

- **Exposure timing** — worked-examples/withheld early vs. available from day one.
- **Tool guardrails** — a *guarded* hint/explanation tutor vs. an open assistant.
- **Mode-of-use rule** — explain/review only, vs. author-then-verify, vs. delegate-and-verify.
- **Verification & security practice** — recurring "find-the-flaw-in-AI-code" labs (see §5.4 for what "verify" means by tool mode).
- **Assessment** — unaided checkpoints (with a predict-then-check step), oral walkthroughs, critique-the-AI tasks (with the fairness caveats in §7/§9).
- **Measurement** — outcome metrics (e.g., DORA) segmented by usage via a non-surveilling method (§9/§10), never individual velocity/credits, and decoupled from performance review.
- **Anti-deskilling** — scheduled manual rotations ( `[Proposed]` ; see §9 on framing).
- **Trust-calibration content** — automation bias, confidence ≠ correctness, the perception gap.

**A transfer note.** Unaided practice builds the schema; but the job the learner returns to *has AI in it*, and transfer-appropriate-processing says training conditions should resemble use conditions. That is the tension the mode-of-use ladder resolves in stages (unaided → author-then-verify → supervised AI-assisted — in Blueprint A, Phase 2's author-then-verify *is* that supervised stage), and it is why far transfer to *new* stacks and tools needs variability of practice — a dimension the minimum-viable tiers (§8) deliberately trade away for speed, and should be understood as optimized for near transfer only.

### 5.4 A note on tool mode: what "verify" actually means

The verification skill the paper keeps calling scarce is not one skill — it changes shape by tool mode, and a curriculum should name the mode it targets:

- **Autocomplete** — verification is *per-line/per-hunk*: is this suggestion correct here? The failure mode is quiet acceptance of plausible wrong lines.
- **Chat/explain** — verification is of *claims and generated snippets*, including the classic **deprecated-API hallucination**: syntactically valid code calling a public function that changed or was removed after the model's training cutoff. API-currency checking is a distinct, teachable verification step and belongs in the labs.
- **Agentic** — verification is *per-PR* and materially harder. These failure modes are practitioner-observed rather than established in controlled study ( `[Proposed]` ), but consistent and

consequential: you review a diff that may span many files, produced by a reasoning chain **not visible in the diff**; the agent may have quietly dropped a constraint it held earlier as its context filled; and re-running the same task can produce **different** code (non-determinism). None of the "code-author intent" cues a human PR carries are present. Programs that raise generation throughput without teaching (and staffing) this per-PR review are accelerating toward the failure mode the paper exists to prevent — see the review-capacity note in §9.

The blueprints in §8 are written for autocomplete/chat as the common baseline; where a cohort is on agentic tools, the assessment and verification labs should use PR-level specimens (multi-file, constraint-drop, non-deterministic reruns), not line-level ones.

---

## 6. Applying the Framework: Three Enterprise Cohort Profiles

These are *instances* of the framework, chosen because they are common in enterprises. They are illustrative, not exhaustive — locate any cohort on the two axes and the levers follow.

### 6.1 People being skilled up into engineering (career-changers, apprentices)

**Axes:** low expertise; most tasks novel. Squarely the highest-risk quadrant — and the one with the strongest evidence.

**What the evidence says vs. implies:** - Sequence it: worked examples to build the schema, *then* unaided attempts, *then* AI critique (checked against the worked example/instructor, per Constant 2). Asking a true novice to generate from nothing is as much a failure mode as handing them answers. `[Implied]` - On guarding vs. removing AI: Bastani established that a guarded, hints-only tutor is **as good as no AI**, and both beat an unguarded answer-engine. It did **not** establish that a guarded tutor beats simply *withholding* AI during schema-building. So the choice between "build a guarded tutor" and "AI off in Phase 1" is a `[Proposed]` pedagogical preference, not an evidence-backed one — and since a guarded tutor is a 1–3 sprint engineering build (§7 Step 5), for a small career-changer cohort the cheaper, evidence-equivalent option may simply be **AI off + instructor-led worked examples**. Choose the tutor on pedagogical or scale grounds, honestly, not on the RCT. `[Implied]` that guardrails-or-absence beats unguarded; `[Proposed]` that the tutor beats absence. - Assessment must move to unaided/oral/critique formats, because "write the code" no longer measures learning. `[Established]` problem; `[Proposed]` remedies.

**Dominant levers:** exposure timing (worked-examples → guarded-or-off → unaided), assessment redesign, metacognitive predict-then-check. **Primary risk:** illusion of competence and the widening gap.

### 6.2 Skilled new hires (experienced engineers, new to your concepts/stack)

**Axes:** medium-to-high expertise, but **low task-familiarity** for the specific concepts and internal systems they're learning — so they move between quadrants concept by concept.

**What the evidence says vs. implies:** - Use AI to *explain and review*, with the human authoring, while a concept is being learned — the mode-of-use study's most transferable result (its subjects were developers). `[Implied]` (n=52, first-party vendor). - Attempt-then-augment, gated by whether *that concept* is new to *that person*, and by demonstrated competence against a pre-set criterion rather than elapsed time (§5.2 Constant 3). `[Implied]` - Note the boundary: the **METR slowdown does *not* apply here** — they are on *unfamiliar* code, not familiar code. Do not import that caution to this cohort. - Their internal stack (proprietary CI/CD, internal libraries) is absent from any model's training data, making it a natural, high-value **trust-calibration** exercise: AI will confidently hallucinate it. `[Implied]` (grounded in the "misplaced trust" harm; the specific exercise is `[Proposed]`). **Two caveats:** (i) if your assistant has internal-doc retrieval (RAG/MCP), it may *not* hallucinate your stack — the lab must be run against a model *without* that context to work as designed; (ii) this exercise has engineers paste proprietary code and internal docs into the model — clear the data-handling path first (§9).

**Dominant levers:** mode-of-use rule, per-concept competence-gating, verification on the real stack. **Primary risk:** skipping the rep on genuinely new concepts.

### 6.3 Veteran engineers (deep experience, often on familiar codebases)

**Axes:** high expertise; frequently high task-familiarity — the slowdown-and-deskilling quadrant. **A candor flag:** this profile rests more heavily than the other two on a single small trial (METR, n=16), so treat its specifics as the most speculative in the paper.

**What the evidence says vs. implies:** - Do not scaffold them like learners (expertise reversal); lead instead with the honest evidence, which calibrates trust and disarms both hype and reflexive rejection. `[Implied]` - The most directly applicable RCT: AI can slow them on familiar code while they believe the opposite → **measure outcomes, be selective by task, don't self-assess.** `[Implied]` (METR, n=16). - Delegation erodes skill (debugging worst) → conceptual-use-over-delegation and, where feasible, deliberate manual rotations to counter deskilling. `[Implied]` (mode-of-use, vendor) + `[Implied]` (deskilling); the rotation *intervention* is `[Proposed]` and needs the political framing in §9. - Adoption is best driven by respected peers demonstrating wins on the team's *actual* tasks, not by mandate — but this is observational. `[Implied]`

**Dominant levers:** trust calibration, measurement, anti-deskilling, verification as sharpened craft, champions over compliance. **Primary risk:** unnoticed slowdown and quiet skill atrophy.

---

## 7. From Framework to Program — a Director's Design Procedure

The framework (§5) answers "what should AI training look like?" with "it depends — on expertise and task-familiarity." This section makes that operational. It begins, deliberately, with the decision *whether* to build a program at all — then a prerequisite check — before any curriculum.

**Step 0 — Decide whether a program is the right instrument (and whether now is the time).** A training program is not free and not the only lever. Run this test, which has **three** honest exits, not

two:

- **What specific failure mode are you buying down?** Name it: skill erosion in veterans, insecure AI output reaching production, the widening novice gap, slow onboarding to your stack. “Increase adoption” is not a failure mode — Cui shows access alone raises adoption and throughput.
- **What is the cheaper alternative, and what exactly would it *not* catch?** Make this comparative, not rhetorical. For insecure output, a **CI security gate / SAST + review policy** likely catches the bulk of the mechanical risk at a fraction of the cost, and should probably exist regardless; a program only earns its keep on the *residual* — the judgment of when to trust and how to verify — and note that program contribution is `[Proposed]`, not proven. If the gate plausibly handles 80% of the failure mode, the honest expected-value case may favor the gate alone.
- **Can you measure lift over the access-only baseline?** Not lift over zero — lift over access-plus-documentation (the Cui +26% world). Define concretely what “access-plus-docs” means in *your* org (a README? a prompt library? an onboarding wiki?), because that is your comparison condition.
- **The three exits.** (1) *A control/gate handles it* → don’t build a program; ship the control. (2) *There is a real judgment gap (recall the program’s contribution to closing it is itself `[Proposed]`), you have the prerequisites (Step 0.5), and you can measure against the baseline* → build the blueprint (or its minimum-viable tier). (3) *You cannot yet name the failure mode with a metric, or you cannot yet measure the baseline* → **wait**: stand up the instrumentation first and revisit in a quarter or two. Running a program you cannot measure is not a bet, it is a spend.

**Step 0.5 — Check prerequisites before you promise a design.** Several blueprint elements assume organizational capabilities L&D may not have. Audit them first; each row names who actually owns the fix and the realistic timeline, because “use a proxy” is a deferral, not a plan:

| Prerequisite | Needed for | If missing — owner & realistic path |
|---|---|---|
| Delivery instrumentation (DORA) **plus** a real test suite + incident record | Veteran/outcome measurement (§10) | *Owner: platform/data eng, not L&D*. Multi-month buildout; until then use a named proxy (e.g. escaped-defect rate on a sampled set) and don't promise DORA proof. Note: DORA's failure/restore keys are meaningless without test coverage to catch regressions. |
| Per-module access-gating (LMS or tooling) | Competence-based fading (Blueprints A/B) | *Owner: L&D + tooling*. Fall back to criterion-referenced module-level gates + honor-policy attempt-first with artifact spot-checks (of work products, not usage logs). |
| Instructors with real SE depth | Oral walkthroughs, find-the-flaw labs | *Owner: eng org (a political ask, not an L&D decision)*. Negotiate protected engineering hours up front, or use peer-panel assessment under rubric. |
| Internal engineering support for tooling | The guarded tutor (Step 5) | *Owner: platform eng*. If unavailable, instructor-mediated AI — but budget it (see Step 5; it is close to a full-time per-cohort role). |
| A placement-diagnostic instrument | Axis placement (§5.1) | *Owner: L&D + a senior engineer*. Build/buy a short unaided explain+bug-find task with a rubric; without it, axis placement is inconsistent. |
| Review capacity for the throughput you'll generate | Any blueprint that raises generation throughput | *Owner: eng org*. Assess current PR cycle time + reviewer load; if already constrained, cap throughput targets or pair with a reviewer-capacity expansion — otherwise the program optimizes for the failure mode §9 warns about. |

**Step 1 — Segment by the two axes, not by job title.** For each cohort, ask: *Can they already build and ship unaided?* and *In the work this program covers, are they mostly learning new concepts, or working in familiar territory?* A single title spans cells — a senior engineer learning your unfamiliar

internal platform is “expert × novel” and behaves differently there than in their home codebase. Segment the *work*, not just the people. Place people with the unaided diagnostic from Step 0.5, not self-report.

**Step 2 — Set the eight levers from the cell.** This table is the bridge from framework to program; the three columns are the three common enterprise profiles (they map to the blueprints in §8). Treat the specifics as illustrative design conventions, not evidence-derived values.

| Lever | Skilling-up *(low expertise · novel)* | New hire *(intermediate · new concept/stack)* | Veteran *(expert · familiar code)* |
|---|---|---|---|
| **Exposure timing** | Worked examples → guarded-or-off → unaided; introduce open AI late | Attempt each concept unaided first, then AI | Available day one; calibrate by task |
| **Tool guardrails** | Guarded tutor *or* AI-off (see §6.1) | Open, under an “explain/review” rule | Open |
| **Mode-of-use rule** | Explain-only early → author-then-verify late | AI explains/critiques; human authors while learning | Conceptual use > delegation; delegate-and-verify only for well-scoped work |
| **Verification & security** | Introduced with AI; graded | Core skill: adversarial review, incl. your stack | Sharpened craft — they are the reviewers |
| **Assessment** | Unaided + oral walkthroughs + predict-then-check (fairness caveats, §9) | “Explain why the AI is right/wrong here” + unaided post-test | Outcomes, not a test |
| **Fade rule** | On a pre-set mastery criterion (e.g. ≥80% unaided), not weeks | Per concept, on the same criterion | N/A |
| **Primary metric** | Unaided competence, pre/post | Unaided concept mastery pre/post *(time-to-first-safe-PR only as a secondary signal)* | DORA segmented by usage (non-surveilling method) + quality |
| **Anti-deskilling / trust** | Confidence ≠ correctness from day one | AI hallucinates your internal/proprietary stack | Scheduled AI-off practice *(see §9)*; automation-bias + perception gap |

**Step 3 — Pick one primary success metric aligned to the cell.** Resist tracking everything; pick the one that predicts real capability, not usage. Keep performance-while-assisted metrics (time-to-first-PR) as *secondary* signals only — they measure output with AI present, not what was learned.

**Step 4 — Instrument before you launch — and budget the instrumentation as its own workstream.** Capture a baseline, define in advance what "working" means, and plan the pre/post measure. This is not a paragraph of program design; if you lack delivery instrumentation and a usable test suite today, standing them up (and segmenting by AI usage *without* individual surveillance — §9) is a multi-month data-engineering project with its own budget and timeline. Program-level effectiveness is `[Proposed]` across the whole field (§4) — so your program *is* the experiment, and Step 4 is how you avoid running it blind. See §10 for the confounds that will otherwise fool you.

**Step 5 — Decide build vs. buy — and budget the build's *maintenance*.** Buy generic fundamentals and tool mechanics; **build** the parts specific to you — the verification labs on your CI/CD, the stack-reality exercises, the assessments. Two realities the word "build" hides: - **The guarded tutor is a software project, not a toggle.** Enterprise AI contracts are tool-access deals; a hints-only, no-answers mode generally does not ship as a setting. Achieving it means a system-prompt wrapper + an access layer a motivated learner can't bypass by opening a second chat, and possibly RAG for domain context — realistically a 1–3 sprint engineering effort that must be owned, funded, and *kept current as model behavior shifts under updates*. Remember (§6.1) the evidence does not say the tutor beats simply withholding AI in Phase 1 — so if you can't staff the build, AI-off + worked examples is a legitimate, cheaper choice, not a compromise. - **The custom labs are a recurring content line, not a one-time artifact.** The stack-reality lab goes stale every time your CI/CD changes; find-the-flaw specimens decay as models improve and as learners memorize them — and failure modes reliable in one model version can shift in the next, so the trust-calibration content itself needs re-calibrating each model-release cycle. Budget a refresh cadence (at least once per program year or per major model release) and name an owner with engineering access. Do **not** buy vendor "adoption" material as if it were pedagogy; it is change-management content, not learning design.

**Step 6 — Bridge the setup period to leadership.** The default enterprise reality is that leadership has *already announced* an AI program on a ~90-day horizon, while Steps 0.5–5 imply months of prerequisite work before the first cohort. If you don't manage that gap, directors cut the prerequisites to hit the date — producing exactly the uninstrumented program this procedure exists to prevent. So plan the narrative: at kickoff, report the **prerequisite audit result and the decision** (build / minimum-viable / wait-and-instrument) as the deliverable; define credible **interim milestones** for months 1–3 that don't require outcome data (baseline captured, placement instrument built, pilot cohort defined, guarded-tutor path decided); and if three of five prerequisites are missing, say so and re-scope to a minimum-viable pilot *with* a stated decision rule, rather than launching the full program undercooked. "We are running a measured pilot and will scale on evidence" is a defensible 90-day story; "we shipped a program we can't yet evaluate" is not. And if Step 0 returned exit (3) — you cannot yet measure the baseline — "not yet" still has a 90-day *deliverable*, not silence: a named instrumentation plan (baseline defined, comparison condition chosen, placement instrument built) and a committed decision date. *"We launch on evidence, not on schedule"* is a defensible position; give leadership that, not a cohort you cannot evaluate.

---

## 8. Reference Program Blueprints

Three adaptable templates, one per common cohort. **Each is illustrative `[Proposed]` engineering — a design sketch, not a validated program.** The durations, ratios, and week-counts are conventions to adapt, not values the evidence dictates; the point of §10 is to instrument whatever you build. Each blueprint ends with a **minimum-viable (MV) tier** — the smallest instrumented version a constrained team can run. The MV tiers reduce scope but do **not** eliminate every dependency; where one is load-bearing, it is named, with a rough staffing figure so the tier is costable.

**Blueprint A — Skilling people up into engineering** *(career-changers, apprentices)* - **Format / duration:** cohort-based, instructor-led, supervised; ~10–16 weeks *(convention)*. - **Sequence:** *Phase 1* — fundamentals via **worked examples → completion problems → unaided attempts**, with **AI off or a guarded hints-only tutor** (the evidence does not favor one over the other — §6.1; pick on staffing/scale grounds). *Phase 2* — introduce AI **explicitly as a taught skill**, paired with verification. **Move Phase 1 → Phase 2 on a pre-set mastery criterion (e.g. ≥80% on an unaided task), not on the calendar.** - **Mode-of-use:** explain-only → author-then-verify. - **Assessment (as a learning event, not just a gate):** weekly **unaided** checkpoints (retrieval practice) + oral walkthroughs; **before each checkpoint, the learner records a predicted score, then compares after — and notes one thing that surprised them about the gap and what they'll do differently** (the reflection, not the number, is what improves calibration). Each checkpoint feeds revision and adapts the next module. Capstone with a mandatory unaided portion. - **Primary metric: unaided competence, pre/post.** - **Staffing:** instructors + (if built) a guarded tutor. Budget oral assessment realistically — ~20–30 min/learner, so ~1 instructor per 8–10 learners at weekly cadence. *Full-tier rough order: 1 lead instructor + ~1 assistant per 16–20 learners, plus a one-time 1–3 sprint tutor build if chosen and its per-release maintenance.* - **Fairness caveat:** oral exams measure articulation as well as understanding and can disadvantage non-native speakers and anxious/neurodivergent learners — use a structured rubric, offer format alternatives, and don't let verbal fluency stand in for competence. - **Peer element (relatedness):** after each individual unaided check, pairs compare their AI critiques (comparing *judgments*, not co-producing code) — cheap, and it addresses the isolation/anxiety that runs high in this cohort (SDT relatedness). - **Main risk & control:** illusion of competence / the widening gap → the unaided checkpoints are the control. - **MV tier (~60 days, minimal build):** worked examples + **AI-off** for the first two weeks (defers the tutor build entirely — do *not* assume a guarded tutor here), **auto-graded timed unaided exercises** in place of weekly orals (orals at the capstone only) — the one required piece, and budget it honestly: this is exercise design + an automated test suite *per exercise* (a few engineering-days per module for someone with SE depth), not merely an LMS/GitHub-Classroom license, **criterion-referenced** module-level gating (a wider gate than per-concept, but it must still be a real threshold, or it is time-based fading). *Staffing: ~1 L&D practitioner + a few borrowed senior-engineer hours/week for a cohort of ~15.* Instrument the pre/post unaided measure even if nothing else.

**Blueprint B — Skilled new hires** *(experienced engineers onboarding to your concepts and stack)* - **Format / duration:** modular, per concept (e.g., 3-tier, DI/IoC, unit testing, your CI/CD, React); ~2–6 weeks *(convention)*. - **Rule:** AI **explains and reviews**; the human authors while a concept is *new to them*; AI unrestricted on concepts they've **demonstrated** mastery of (fade per concept on the pre-set

criterion, not time). - **Signature module — "stack-reality lab":** AI confidently hallucinates your proprietary CI/CD and internal libraries; engineers verify against the real docs. Does trust-calibration and stack-onboarding in one. **Run it against a model *without* internal-doc retrieval** (RAG/MCP would defeat it), and **clear the data-handling path first** — it routes proprietary code/docs to a third-party model (§9). - **Verification lab:** recurring "find-the-flaw-in-AI-code," including a deliberate security flaw *and* a deprecated-API specimen (§5.4); refresh specimens on a cadence (Step 5). For agentic-tool cohorts, use PR-level (multi-file) specimens, not line-level. - **Assessment:** "explain why the AI's suggestion fits or violates our design," an accept/reject review artifact, **plus a per-module unaided post-test** (with a predict-then-check step — predict, compare, then note what was misjudged) before AI goes unrestricted on that concept. - **Sequencing:** where far transfer to varied production code matters, *interleave* concepts rather than teaching them in strict blocks — blocked practice speeds initial acquisition, interleaving improves transfer; choose with that trade-off explicit. - **Primary metric:** unaided concept mastery pre/post (time-to-first-safe-PR is a *secondary* signal only). - **Main risk & control:** skipping the rep on genuinely new concepts → attempt-first, *verified by the unaided post-test* (attempt-first is otherwise unenforceable at scale). Spot-checks are of **work artifacts**, not individual usage logs. - **MV tier (~60 days):** one **criterion-referenced module-level** mastery gate per concept (not per-line policing), honor-policy attempt-first with artifact spot-audits. The stack-reality lab is **the one custom piece the MV tier requires** — budget ~a few senior-engineer hours to build the first version (or substitute a generic public-CI hallucination exercise until it exists), and **clear the data-handling path (§9) before building it — a launch blocker even at the MV tier.** *Staffing: ~1 L&D practitioner + ~2 borrowed engineering hours/week per cohort.* Note: a module-level gate is a real degradation of per-concept fading (a learner can reach unguarded AI on concept 4 without mastering concept 3) — instrument the pre/post measure with extra care to catch it.

**Blueprint C — Veterans** *(deep experience; an enablement track, not a course)* - **Candor flag (read first):** this blueprint rests disproportionately on a single n=16 field RCT (METR); its specifics are the most speculative in the paper. Run it as an explicit experiment. - **Format:** (i) a half-day **evidence seminar**, then (ii) an **ongoing guild / community of practice** with office hours — not a one-shot class. - **Seminar:** the honest evidence (including that AI can slow experts on familiar code without their noticing), a **task-class map** (where AI helps vs. hurts), and the mode-of-use rule. - **Guild:** respected peers demonstrating wins **on the team's real tasks**; workshops on the team's own codebase; verification as sharpened craft; a **deliberate manual-practice cadence** to counter deskilling. *Facilitation: ~a respected senior engineer at ~10% time + periodic office hours — protect that time from sprint commitments explicitly.* - **On AI-off practice — mind the politics.** A blanket "stop using the mandated tool" period lands as compliance theater and as a velocity hit, and is unenforceable without the individual monitoring §10 forbids. Frame it as **deliberate practice** scoped to a specific lab/kata or a **code-review-only session** (review AI-suggested code without AI assistance) — a schedulable, delivery-compatible substitute — not a sprint-wide prohibition. Treat it as `[Proposed]`. - **Assessment:** none as a test — measure outcomes. - **Primary metric:** DORA four keys **segmented by AI usage** (via a non-surveilling method, §9/§10) + qualitative feedback; **never** individual velocity or AI-credit counts. In most site-licensed enterprises no non-surveilling exposure variable exists — if so, skip segmented DORA and use qualitative feedback + a named delivery-quality proxy (e.g. escaped-defect rate on a

sampled sprint), as in the MV tier. Read as association, not proof (§10). - **Main risk & control:** unnoticed slowdown and quiet skill atrophy → measurement + deliberate-practice cadence are the controls. - **Note (from prior art):** at least one large enterprise program (the DX/Indeed "AICE" case study) found **champions/train-the-trainer *alone* underperformed** while direct, structured hands-on training worked — so pair champions *with* structured sessions, don't substitute them. - **MV tier:** the evidence seminar + a **standing retrospective cadence** on AI-assisted decisions in place of a full guild — **the retrospective cadence is the one piece this tier cannot drop** (run it ~monthly); skip DORA-by-usage if you can't instrument it and use qualitative + a single named delivery-quality proxy (e.g. escaped-defect rate). *Staffing: ~1–2 days of seminar prep + ~2–4 facilitator hours/month.*

---

## 9. Running It: The Organizational Layer

- **Translate the interest/mandate into capability, not compliance.** A "use AI" mandate that lands as a usage quota or a review-time KPI backfires — Self-Determination Theory explains why (it strips autonomy, predicting amotivation), and it produces gaming and resentment while system quality quietly degrades. Frame the program's goal as *using AI well, and knowing where not to*, with real method-choice. `[Implied]` (observational + theory).
- **Resolve the measurement-vs-surveillance tension explicitly.** §10 recommends segmenting outcomes by AI usage while forbidding individual AI metrics — those pull against each other, because segmenting *requires* knowing who uses AI. Resolve it by segmenting via a **non-surveilling method**: self-reported cohort tier at enrollment (heavy/light/off), license- or tool-tier distinctions that already exist for billing, or opt-in usage diaries — **never** passive per-person tracking piped to a dashboard a manager or HR can query. If you cannot segment without individual surveillance, don't segment. One honesty caveat: self-reported tiers carry the same unreliability the paper flags for self-report generally, and under a mandate people under-report into "light/off" — which *compresses the very heavy-user segment* the equity monitoring wants to see. Treat segments as directional and corroborate against a non-surveilling signal (e.g. license-tier data) where one exists.
- **Give engineers a data-collection notice.** Distinct from consultation: tell engineers what data is collected about them, how long it is retained (no longer than the program plus a defined follow-up window — not indefinitely, and not folded into general HR records), who can see the individual-level records that feed any cohort aggregate, and that measurement participation is not a performance condition. You cannot claim psychological safety around honest reporting if people don't know what is logged; in some jurisdictions this is also a legal requirement.
- **Consult the workforce; protect psychological safety — with a mechanism, not a hope.** If engineers fear that "AI made me slower" is a career risk, your data will be dishonest and adoption performative. Involve engineers in the design, and put at least one *structural* protection in place: an anonymous program-feedback channel, an explicit written policy that AI usage is not a performance variable, or manager guidance on what they may not ask.
- **Data handling is a launch blocker, not a footnote.** Every blueprint routes real code — and, in the stack-reality lab, *proprietary* code and internal docs — to a third-party model. Before any cohort

starts, clear these questions with legal/privacy: *Does our enterprise agreement prohibit provider training on our inputs? Do retention terms meet our data-classification tier? Do we need a DPA for this use case?* And know what an unacceptable answer is: **if the provider's standard terms permit training on customer data and no no-retention enterprise tier is available, that is a blocking finding, not a caveat to manage.**

- **Equity is first-order here, because the central finding is that AI widens gaps.** Design for it explicitly: uniform tool access, *and* verify the chosen tool is accessible to engineers who use screen readers or have motor/cognitive differences (with a remediation path if not); a placement step (§5.1) that routes strugglers to more scaffolding early — communicated as a *starting-point diagnostic, revisable as competence grows*, not a durable label, and identified by the diagnostic, not by manager impression; and monitoring disaggregated by starting level (cohort averages hide divergence).
- **Ethics of restricting a mandated tool.** Blueprint A withholds AI in Phase 1 and Blueprint C schedules AI-off practice, possibly under an organization-wide AI mandate. Communicate any such restriction to *both* engineers and their managers, and exclude restricted periods from any adoption reporting, so no one is penalized for following the program.
- **Mind the review bottleneck — it is the central argument, not a footnote.** The paper's whole thesis is that *verification* is the scarce, teachable skill. It follows that AI raising code *generation* faster than review capacity pushes the system toward the exact failure it warns about: if reviewer time is the binding constraint, a program that lifts throughput without lifting review capacity accelerates quality decay at the merge point. Treat review capacity as a named prerequisite for any throughput-raising rollout.
- **Champions — necessary, not sufficient.** Peer champions on real tasks are among the best-observed adoption levers, but the case evidence is mixed on champions *alone* (§8, Blueprint C) — amplify structured training with them, don't replace it. `[Implied]`.
- **Integrate into real work.** Enablement on the team's actual codebase outperforms abstract demos. `[Implied]`.
- **Staff the program (with rough time commitments, not just titles):** a program owner (accountable; ~0.3–0.5 FTE), instructors/coaches with SE depth (per the blueprint ratios), champions embedded in teams (a few hours/week each), a **measurement owner** (§10; ~0.2 FTE of a data analyst, more during setup), a lab-content owner with engineering access (a standing per-release commitment), and an executive sponsor who protects the "capability not compliance" framing. For a 1–3 person L&D team this exceeds headcount — which is the honest case for the minimum-viable tiers.

---

## 10. Measuring It and Proving It Works

The program (not the person running it) has to earn its spend. The trap is measuring the wrong thing — and the second trap is measuring the right thing badly.

**Do not measure** (these backfire or mislead): AI usage %, commits, AI credits consumed, or **self-reported speedup** — the last is unreliable (experienced developers misjudged AI's effect on themselves by ~39 points in METR, n=16), the rest invite gaming and mistake activity for value. `[Implied]`.

**Do measure**, by cohort: - **Learners (Blueprints A/B):** *unaided* competence, pre/post (oral or live coding without AI) — the only thing that predicts real ability once assessment-by-code is broken. - **Practitioners (Blueprint C): DORA four keys segmented by AI usage** (non-surveilling method, §9) + qualitative feedback; for new hires, unaided concept mastery, with time-to-productivity as a secondary signal. - **The program itself:** completion → *downstream outcome*, not adoption rate.

**Name the confounds, or they will fool you — and your leadership.** An uncontrolled pre/post confounds the program with **maturation** (people improve on the job anyway), **selection** (volunteers differ from the mandated), **Hawthorne** effects (being measured changes behavior), and — most dangerous in training, where cohorts are often enrolled *because* they scored low — **regression to the mean** (they will trend upward on re-measurement regardless of the program; the fix is a matched control group — or, failing that, pre-registering the expected improvement magnitude so regression-sized gains can be told apart from program effects — not a stepped rollout alone). "DORA segmented by AI usage" is observational: heavy users differ from light users for reasons unrelated to your program — this is **confounding by self-selection** (non-random exposure correlated with unmeasured skill/motivation), which needs covariate adjustment or assigned exposure, not a sampling fix. Reach for the strongest design your context allows — a **waitlist/stepped rollout**, a **matched-cohort** comparison, or **difference-in-differences** against a comparable un-trained team.

**Be honest about what most enterprises can actually run.** The stronger designs are often unavailable in an already-licensed org (a waitlist means withholding a licensed tool; a matched cohort needs cross-BU data access; diff-in-diff needs segmented history) — so most teams are left with the **uncontrolled pre/post above, which yields an *association*, not causal proof.** That is still worth running: do it, say plainly it is an association (don't let "we saw improvement" imply "the program caused it"), add a matched comparison wherever you can, and frame the result as one contribution to a *pattern* — it takes consistent lift across many instrumented programs, in the predicted quadrant shape, before the field's `[Proposed]` claims upgrade toward `[Implied]`.

**Report to leadership as an instrumented bet, not a proven win — and translate it into the spend conversation.** Walk in leading with the **failure mode you are buying down**, the **access-only baseline** you are competing against, and the **pre-stated decision rule** (what result makes you scale, hold, or stop). That framing is a *sales advantage*, not a confession: it distinguishes your program from vendor claims leadership has already learned to discount, and it pre-commits you to killing the program if it doesn't work — which is what earns the budget for the version that does. And **decouple AI metrics from individual performance review**; the observational evidence (and §9's psychological-safety point) says this is a precondition for honest data.

**Minimum measurement template:** *baseline (incl. the access-only comparison) → intervention → pre/post measure (per the cohort's primary metric) → decision rule.* One page. Run it every cohort. Be clear-eyed that this template is an *uncontrolled* pre/post — strengthen it with a matched or waitlisted

comparison group *wherever you can*, and where you can't, report its results as association, not cause. This template *is* the minimal instrumented pilot from §1 and Step 0 — start here, scale on evidence.

---

## 11. Program Anti-Patterns (what wastes the budget)

- **Training when a control would do** — buying a course to fix insecure output that a CI security gate would catch more reliably and cheaply; train for judgment gaps, gate for mechanical ones. `[Implied]`
- **Launching a program you cannot measure** — with no baseline and no comparison, maturation, selection, and regression to the mean will manufacture a "result"; if you can't instrument it, wait and instrument first (§7 Step 0). `[Implied]`
- **Usage mandates and activity KPIs** (commits, credits) — backfire; measure outcomes instead. `[Implied]`
- **One training regime for everyone** — ignores the expertise-reversal effect; the same content over- or under-serves half your population. `[Implied]`
- **Unguarded answer-giving AI for learners** — the arm shown to *reduce* unaided learning; use a guarded tutor *or* withhold AI. `[Implied]` (the RCT is in math; cross-domain to coding).
- **Time-based instead of competence-based fading** — withdraws support from slow learners and delays it for fast ones; gate on a pre-set mastery criterion. `[Implied]`
- **Segmenting metrics by usage via individual surveillance** — poisons the honest-data well the measurement depends on; use a non-surveilling method (§9). `[Implied]`
- **Champions with no structured training** — under-performed in at least one large enterprise program. `[Implied]`
- **Measuring adoption instead of capability/outcomes** — high adoption of a skill-eroding habit is not a win. `[Implied]`
- **Dropping verification and security from the curriculum** — a substantial share of AI code carries vulnerabilities; verification is the scarce skill. `[Implied]`
- **Believing self-reported speedups** — they are systematically unreliable. `[Implied]`
- **Buying vendor "adoption" content as pedagogy** — it is change management, not learning design.

---

## 12. Limits of This Paper

This is a reasoned synthesis, not a proof. The controlled base is a **small and lopsided set of randomized trials** — one large, positive enterprise trial (n=4,867; developers on well-scoped completion; a weaker subgroup split on experience), one small cautionary trial (n=16; experts slowed on familiar code), one cross-domain learning-design trial (guarded beats unguarded, in math), and one small first-party mode-of-use study (n=52). The two-axis framework *reconciles* these, but as §4.1 and

§5.1 state plainly, that reconciliation is a **post-hoc explanation, not a controlled demonstration** — a 2×2 can absorb any four points — and two of the framework's four quadrants are predictions no study has yet tested. The sample asymmetry (thousands vs. dozens) means our confidence is not uniform: recommendations resting mainly on the small trials (much of the veteran profile) are more speculative than those resting on the large one. A deeper risk than "thin evidence" is that **the decomposition itself could be wrong** — a future study could show expertise × task-familiarity is not the right carve-up, which would invalidate blueprint *designs*, not merely narrow their confidence intervals. Almost every **program-level** recommendation here is `[Implied]` or `[Proposed]`; the blueprints are design sketches, not validated curricula.

A broader risk the paper does not solve: an over-corrected program that withholds AI too long, or a workforce trend of AI displacing junior work, could **hollow the early-career pipeline** the industry depends on to produce future seniors — a system-level effect no single program controls but every director should watch. The literature is moving fast and specifics may be overturned. We have deliberately foregrounded caution over enthusiasm — which will read as less confident than typical vendor material, by design. The remedy is in §10: treat your program as an instrumented experiment and let your own data — pooled with others' — upgrade these claims over time.

---

## 13. Declaration of AI Assistance

Generative AI (Claude, via Claude Code) assisted the research, source-gathering, and drafting of this paper under human direction. The author directed the work, made all editorial judgments, and is responsible for its content. The evidence grades, the reconciliation of conflicting studies, and the honest accounting of how thin the controlled base is were the explicit design goals. **Every source cited in this paper has been independently verified against its primary source** — the empirical studies through two independent web-verification passes (checking existence, citation, and each load-bearing figure), and the learning-science and prior-art citations through a separate attribution check; full citations are in §16. This draft was revised across multiple rounds of independent expert-persona and adversarial review (learning science, evidence/measurement, responsible-AI, distinguished-engineer, and L&D perspectives; plus steelman critiques of the premise, the evidence, and operational feasibility). One residual bias worth naming: AI was used to help review claims that include AI-vendor sources — mitigated, not eliminated, by human review and primary-source checks.

---

## 14. Evidence Appendix — studies by design & grade

The load-bearing studies, with the design that sets their grade and how far each travels to enterprise software work. "Grade here" is the grade *as used in this paper* — an in-domain `[Established]` result becomes `[Implied]` when we extrapolate it to enterprise coding, and small/vendor RCTs are `[Implied]` per §3.

| Study | Design & population | Key result | Grade here |
|---|---|---|---|
| **Cui, Demirer, Jaffe et al.**, *Management Science* 2025 | Pre-registered RCT, **n=4,867**, Microsoft / Accenture / Fortune-100 developers; AI code completion | **~26% more completed tasks**; weaker (non-primary, unreplicated) subgroup: less-experienced gained more | `[Established]` (throughput); subgroup much weaker |
| **METR (Becker et al.)** 2025 | Field RCT, **n=16**, experienced OSS devs on **familiar** mature repos; mixed tool modes (chat, Cursor agent mode, autocomplete) | AI **+19% task time** (slower); devs believed ~20% faster (~39-pt perception gap) | `[Implied]` — strongest small independent field RCT on this, but n=16 |
| **Bastani et al.**, *PNAS* 2025 | Class-randomized, **~1,000** high-school **math** students | Unguarded AI **−17%** on unaided exams; guarded hints-only ≈ no-AI control (parity, not superiority) | `[Established]` in-domain → `[Implied]` for coding |
| **Shen & Tamkin / Anthropic** 2026 | Randomized, **n=52** developers; mode of use | Conceptual/explain use preserved comprehension; delegation eroded it (debugging worst) | `[Implied]` — small n, first-party vendor |
| **Fu et al.** (repo analysis) + developer survey | Observational; AI-generated code corpora | ~25–30% of AI snippets carry CWE-class weaknesses (context-dependent); most devs don't consistently verify | `[Implied]` |
| **Novice over-reliance studies** (incl. eye-tracking) | Observational | Prepared novices plan with AI; underprepared form "illusion of competence," gap widens | `[Implied]` |
| **DORA 2024/2025** + repo-mining (GitClear) | Large survey / telemetry | AI adoption associated with rising instability and structural debt absent testing discipline (the specific −1.5%/−7.2% instability figures are from DORA 2024) | `[Implied]` (correlational) |
| **Frontier-model exam performance** | Benchmark | Models pass standard intro programming exams → "write the code" no longer measures ability | `[Established]` (as a problem); replacements `[Proposed]` |

| Study | Design & population | Key result | Grade here |
|---|---|---|---|
| **Learning-science & human-automation theory** — expertise-reversal (Kalyuga et al.), worked-example (Sweller & Cooper) and generation (Slamecka & Graf) effects, desirable difficulties (Bjork), retrieval practice/spacing (Roediger & Karpicke; Cepeda et al.), automation bias/trust calibration (Parasuraman; Lee & See), deskilling (Bainbridge), metacognition (Kruger & Dunning), mastery learning (Bloom), transfer (Morris et al.), Self-Determination Theory (Ryan & Deci) | Seminal theory + cross-domain evidence | Mechanisms behind the framework; travel well but not tested on AI-assisted coding | `[Implied]` |

*Full citations are in §16.*

---

## 15. Glossary

Terms used in this paper, grouped by domain. Definitions are scoped to how the term is used here, not exhaustive.

### Learning science & pedagogy

- **Expertise-reversal effect** — instructional support that helps a novice can *hinder* an expert on the same material; guidance should fade as competence grows.
- **Worked-example effect** — for genuine novices, studying a fully worked solution builds schema faster than unguided problem-solving.
- **Generation effect** — information a learner produces themselves is retained better than information they are handed; requires enough prior knowledge to attempt.
- **Desirable difficulty** — a deliberate obstacle (e.g. withholding the answer) that slows performance now but improves durable learning.
- **Retrieval practice (testing effect)** — the act of recalling from memory *strengthens* it; an unaided check is a learning event, not only a measurement.

- **Spacing / interleaving** — distributing practice over time (spacing) and mixing concepts rather than blocking them (interleaving) improves retention and transfer.
- **Cognitive load** — working-memory demand, split into *intrinsic* (inherent task difficulty), *extraneous* (avoidable, from poor design — safe to offload), and *germane* (the effortful schema-building the task exists to produce — harmful to offload).
- **Scaffolding / fading** — temporary support (hints, worked examples, a guarded tool) that is progressively withdrawn as the learner masters the material.
- **Mastery learning / criterion-referenced** — advancing a learner only when they clear a *pre-specified* performance threshold (e.g. ≥80% on an unaided task), not after a fixed amount of time.
- **Formative vs. summative assessment** — *formative* feeds back into learning (attempt → feedback → revise); *summative* measures final attainment.
- **Metacognition / calibration** — a learner's awareness of what they do and don't know; *miscalibration* (over-confidence) is what AI can mask by removing error signals.
- **Transfer (near / far); transfer-appropriate processing** — applying a skill to new situations; *near* transfer is to similar problems, *far* to novel ones. Learning conditions should resemble the conditions of eventual use.
- **Automation bias / complacency** — over-trusting an automated aid: accepting its output without checking, or reducing one's own vigilance.
- **Trust calibration** — matching one's trust in a tool to its *actual* reliability; built through exposure to real failure modes, not exhortation.
- **Deskilling** — erosion of a skill through disuse when a tool makes it effortless.
- **Self-Determination Theory (SDT)** — motivation theory holding that autonomy, competence, and relatedness drive engagement; mandates that strip autonomy predict amotivation.

**Statistics & research method**

- **Randomized controlled trial (RCT)** — participants randomly assigned to conditions, so a measured difference can be attributed to the intervention rather than to who was in each group.
- **Pre-registration / primary endpoint** — declaring the hypothesis and the main outcome measure *before* running the study; the pre-registered primary result is stronger evidence than a finding discovered afterward.
- **Subgroup analysis** — examining an effect within a slice of the sample (e.g. less-experienced developers); more prone to false positives than the primary result.
- **n** — sample size (number of participants). Larger n generally means more reliable estimates.
- **Confound** — a variable that offers a rival explanation for a result. Common ones here: *maturation* (people improve over time anyway), *selection* (volunteers differ from the mandated), *Hawthorne effect* (being observed changes behavior).
- **Regression to the mean** — extreme initial scores tend to move toward the average on re-measurement; a low-scoring cohort improves on retest regardless of any intervention.
- **Confounding by self-selection** — when who receives a treatment (e.g. heavy AI users) is non-random and correlated with unmeasured traits (skill, motivation); differences are associations, not

proof of cause.

- **Association vs. causation** — a correlation between two things does not establish that one caused the other.
- **Waitlist / stepped-rollout design** — giving the intervention to groups in sequence so an untreated group serves as a temporary control.
- **Matched cohort** — a comparison group chosen to resemble the treated group on key traits, to approximate a control.
- **Difference-in-differences** — comparing the *change* in a treated group against the change in an untreated group over the same period.
- **Effect size** — the magnitude of a result (e.g. "+26%"), distinct from whether it is statistically significant.

**DevOps & engineering**

- **DORA metrics (four keys)** — deployment frequency, change lead time, change failure rate, and time to restore service; a standard measure of software-delivery performance.
- **CI/CD** — continuous integration / continuous delivery: the automated pipeline that builds, tests, and ships code.
- **CI security gate / SAST** — an automated check in the pipeline (e.g. static application security testing) that blocks insecure code before merge.
- **PR (pull request)** — a proposed change submitted for review before it is merged into the codebase.
- **CWE (Common Weakness Enumeration)** — a standard catalog of software security-weakness types.
- **Escaped-defect rate** — the share of defects that reach production rather than being caught earlier; a delivery-quality proxy.
- **Tool modes — autocomplete / chat / agentic** — *autocomplete* suggests code inline as you type; *chat* is a conversational assistant beside the editor; *agentic* tools plan and edit across many files or run autonomously. Verification shifts from per-line to per-PR as you move toward agentic use.
- **Context window** — the amount of text (code + conversation) a model can consider at once; exceeding it can cause an agent to silently drop earlier constraints.
- **RAG (retrieval-augmented generation)** — supplying a model with retrieved documents (e.g. internal docs) at query time so its answers reflect them.
- **MCP (Model Context Protocol)** — a standard for connecting AI assistants to external tools and data sources, including internal documentation.

---

## 16. References

Every source cited in this paper was independently verified against its primary source: the empirical studies through two independent web-verification passes (existence, citation, and each load-bearing figure), and the learning-science and prior-art citations through a separate attribution check. Where a

claim is graded `[Implied]` or `[Proposed]`, that reflects the *strength or transferability* of the evidence, not doubt about the source's existence.

**Randomized controlled trials**


- Cui, Z. (Kevin), Demirer, M., Jaffe, S., Musolff, L., Peng, S., & Salz, T. (2025). The effects of generative AI on high-skilled work: Evidence from three field experiments with software developers. *Management Science* (published online Feb 2026). https://doi.org/10.1287/mnsc.2025.00535 — *pre-registered RCT, n=4,867; +26.08% completed tasks (SE 10.3%); the "less-experienced gain more" split is a non-primary, unreplicated subgroup finding. Tool tested: GPT-3.5-era GitHub Copilot.*
- Becker, J., Rush, N., Barnes, E., & Rein, D. (2025). Measuring the impact of early-2025 AI on experienced open-source developer productivity. METR. arXiv:2507.09089. https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/ — *field RCT, n=16, familiar repos; AI +19% task time while developers believed ~20% faster (~39-pt gap); mixed tool modes (chat, Cursor agent mode, autocomplete).*
- Bastani, H., Bastani, O., Sungu, A., Ge, H., Kabakcı, Ö., & Mariman, R. (2025). Generative AI without guardrails can harm learning: Evidence from high school mathematics. *PNAS, 122*(26). https://doi.org/10.1073/pnas.2422633122 — *class-randomized, ~1,000 students; unguarded AI −17% on unaided exams; a guarded hints-only tutor ≈ the no-AI control (parity, not superiority).*
- Shen, J. H., & Tamkin, A. (2026). How AI impacts skill formation. Anthropic. arXiv:2601.20245. https://www.anthropic.com/research/AI-assistance-coding-skills — *randomized, n=52; conceptual/explanatory use preserved comprehension while delegation eroded it (debugging worst); first-party vendor research.*


**Observational & industry evidence**


- Fu et al. (2025). Security weaknesses of Copilot-generated code in GitHub projects: An empirical study. *ACM Transactions on Software Engineering and Methodology*. https://doi.org/10.1145/3716848 (arXiv:2310.02059) — *~29.6% of generated snippets carried CWE-class weaknesses (29.5% Python, 24.2% JavaScript).*
- Sonar (2026). State of Code: Developer Survey 2026 (1,100+ developers). https://www.sonarsource.com/ — *only 48% of developers report always verifying AI-generated code before committing. Vendor-commissioned.*
- DORA / Google Cloud. Accelerate State of DevOps Report (2024); State of AI-Assisted Software Development (2025, ~5,000 respondents). https://dora.dev/ — *AI as an "amplifier/mirror"; adoption associated with reduced delivery stability absent strong testing discipline. The specific −1.5% throughput / −7.2% stability figures are from the 2024 report.*
- GitClear (2025). AI Copilot code quality: 2025 research (211M changed lines, 2020–2024). https://www.gitclear.com/ai_assistant_code_quality_2025_research — *cloned lines rose 8.3% → 12.3%; refactoring fell from ~25% (2021) to <10% (2024).*
- Finnie-Ansley, J., Denny, P., Becker, B. A., Luxton-Reilly, A., & Prather, J. (2022). The robots are coming: Exploring the implications of OpenAI Codex on introductory programming. *ACE 2022*.

https://doi.org/10.1145/3511861.3511863 — *Codex placed 17th of 71 on CS1 exam problems, outscoring most students; "write the code" no longer isolates ability.*

- Prather, J., Reeves, B., Denny, P., Becker, B. A., Leinonen, J., Luxton-Reilly, A., Powell, G., Finnie-Ansley, J., & Santos, E. A. (2024). "It's weird that it knows what I want": Usability and interactions with Copilot for novice programmers. *ACM Transactions on Computer-Human Interaction, 31*(1). https://doi.org/10.1145/3617367 — *think-aloud + eye-tracking; names the "shepherding" and "drifting" over-reliance patterns.*
- Prather, J., et al. (2024). The widening gap: The benefits and harms of generative AI for novice programmers. *ICER 2024*. https://doi.org/10.1145/3632620.3671116 — *eye-tracking; underprepared novices finish with an "illusion of competence" as the performance gap widens.*

**Learning science & human-automation theory**

- Bainbridge, L. (1983). Ironies of automation. *Automatica, 19*(6), 775–779. — *deskilling*.
- Bjork, R. A. (1994). Memory and metamemory considerations in the training of human beings. In J. Metcalfe & A. Shimamura (Eds.), *Metacognition: Knowing about knowing* (pp. 185–205). MIT Press. — *desirable difficulties.*
- Block, J. H., & Burns, R. B. (1976). Mastery learning. *Review of Research in Education, 4*, 3–49; and Bloom, B. S. (1968). Learning for mastery. *Evaluation Comment, 1*(2), UCLA (report). — *mastery learning; the ≥80% criterion.*
- Cepeda, N. J., Pashler, H., Vul, E., Wixted, J. T., & Rohrer, D. (2006). Distributed practice in verbal recall tasks: A review and quantitative synthesis. *Psychological Bulletin, 132*(3), 354–380. — *spacing effect* (originating work: Ebbinghaus, 1885).
- Kalyuga, S., Ayres, P., Chandler, P., & Sweller, J. (2003). The expertise reversal effect. *Educational Psychologist, 38*(1), 23–31. — *expertise-reversal.*
- Kruger, J., & Dunning, D. (1999). Unskilled and unaware of it: How difficulties in recognizing one's own incompetence lead to inflated self-assessments. *Journal of Personality and Social Psychology, 77*(6), 1121–1134. — *metacognitive miscalibration.*
- Lee, J. D., & See, K. A. (2004). Trust in automation: Designing for appropriate reliance. *Human Factors, 46*(1), 50–80. — *trust calibration.*
- Morris, C. D., Bransford, J. D., & Franks, J. J. (1977). Levels of processing versus transfer appropriate processing. *Journal of Verbal Learning and Verbal Behavior, 16*(5), 519–533. — *transfer-appropriate processing*.
- Parasuraman, R., & Riley, V. (1997). Humans and automation: Use, misuse, disuse, abuse. *Human Factors, 39*(2), 230–253; and Parasuraman, R., & Manzey, D. H. (2010). Complacency and bias in human use of automation: An attentional integration. *Human Factors, 52*(3), 381–410. — *automation bias & complacency*.
- Roediger, H. L., & Karpicke, J. D. (2006). The power of testing memory: Basic research and implications for educational practice. *Perspectives on Psychological Science, 1*(3), 181–210 (companion experimental paper: *Psychological Science, 17*(3), 249–255). — *retrieval practice / testing effect*.

- Ryan, R. M., & Deci, E. L. (2000). Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American Psychologist, 55*(1), 68–78. — *Self-Determination Theory*.
- Slamecka, N. J., & Graf, P. (1978). The generation effect: Delineation of a phenomenon. *Journal of Experimental Psychology: Human Learning and Memory, 4*(6), 592–604. — *generation effect*.
- Sweller, J., & Cooper, G. A. (1985). The use of worked examples as a substitute for problem solving in learning algebra. *Cognition and Instruction, 2*(1), 59–89. — *worked-example effect*.
- Sweller, J., van Merriënboer, J. J. G., & Paas, F. G. W. C. (1998). Cognitive architecture and instructional design. *Educational Psychology Review, 10*(3), 251–296. — *cognitive load types (intrinsic / extraneous / germane)*.


**Prior art / case studies**


- Noda, A. (host) (2026, June 29). 2× the power users: How structured AI training scaled developer productivity [DX "Engineering Enablement" newsletter, featuring M. Redding & J. Davis, Indeed]. https://newsletter.getdx.com/p/2x-the-power-users-how-structured — *Indeed's champions-only "AI Coding Ambassadors" program saw teammate adoption decline after it ended; a direct structured program (AICE) cut coding time ~35% for participants — the basis for "pair champions with structured training, don't substitute them."*