The agentic engineer

Who should direct the machines that write the code

Mark Dickie, Tarmac. September 2026.

The short version

Writing code used to be the expensive part of building software. At the firms furthest along, it is now the cheap part. Sundar Pichai wrote in April 2026 that 75% of new code at Google is generated by AI and then approved by engineers. Stripe merges more than a thousand pull requests a week that contain no human-written code.

The expensive part is now everything around the typing. Someone has to decide what to build, say it precisely enough for a machine to act on, prove the result is right, and answer for it when it breaks. Andrej Karpathy gave that practice a name in February 2026: agentic engineering. The person who does it still has no settled job title.

My argument is that for most business software, the best person for this job is someone who already knows the field the software serves, and who has learned to specify and check what an AI agent builds and to put guardrails around it. The evidence turned out stronger than I expected, and thinner than the hype suggests. Every figure below links to its source.

One limit up front. For systems that hold customer or regulated data and serve many users, I found nothing that supports a domain expert shipping alone. That work still needs an engineer to check security and data handling independently. The section near the end on where this is the wrong choice draws the line.

I run Tarmac, which sells interview preparation to software engineers. Weigh what follows with that in mind.

What changed

Start with how much of the code machines now write. Google's figure was 50% in autumn 2025 and 75% by April 2026. Anthropic reported that more than 80% of the code it merged in May 2026 was written by its own model, and that review had become the slow step. Spotify said in February 2026 that its most productive developers had not written a line of code since December.

Don't lean on those numbers too hard. They're executive statements. Nobody audits them, and no firm says how it counts. Every one of these companies still has a person approve the change.

The more useful number is how little of the job typing ever was. Bain's 2025 technology report puts writing and testing code at 25 to 35% of the time from idea to launch. Atlassian's 2025 survey found that developers spend 16% of their week coding. Bain measured gains of 10 to 15% from coding assistants alone. The gain reached 25 to 30% only at companies that rebuilt their whole delivery process around the tools. Cheap code is worth something only when the rest of the process can keep up with it.

Fred Brooks drew the line in 1986. In No Silver Bullet he split software work in two: the accidental difficulty of expressing a design in code, and the essential difficulty of deciding what the design should be. Coding agents go after the first. The second is mostly knowledge of the business the software serves.

The oldest project data points the same way. The 1994 Standish survey (365 respondents, 8,380 applications) ranked user involvement as the top reason projects succeeded, at 15.9% of responses. Incomplete requirements topped the reasons projects were cancelled, at 13.1%. Technology incompetence was blamed for 7.0% of troubled projects. It's an opinion survey and it's thirty years old, but I haven't seen anything overturn its shape.

What to call it

Nobody agrees yet. The terms in use fall into a few families, and they don't line up.

Term	Origin	What it covers	Includes non-programmers?
Agentic engineering	Karpathy, February 2026; restated April 2026. Simon Willison's patterns guide dates from the same month	The practice of directing coding agents while holding a professional quality bar	Not as defined. Both authors describe a professional engineer
Vibe coding	Karpathy, February 2025; Collins Word of the Year 2025	Building without reading the code	Yes, but quality is outside its definition
Forward deployed engineer	Palantir; spread to AI firms after a16z's June 2025 essay	An engineer embedded with a customer	Mostly no
Builder titles: LinkedIn's Full Stack Builder, Walmart's Agent Builder, Meta's AI Builder	LinkedIn, December 2025; others by March 2026	One person doing product and design work as well as code	Yes
Domain-prefixed engineer: GTM engineer, legal engineer	Clay, 2023; law firms from about 2017	A domain expert who builds systems for that domain	Yes, by design
Harness engineering, context engineering, spec-driven development	2025 and 2026	Sub-skills of the practice	Not addressed

Only one of these has hard hiring data behind it: forward deployed engineer. Indeed postings for it in April 2026 stood 729% above a year earlier. One tracker counted 1,333 live postings at 565 companies on 12 September 2026, with a median base salary of $193,000. That's fast growth from a small base. It also runs the opposite way to my argument, because those jobs go to engineers who then learn the customer's field.

The domain-prefixed titles are closer to what I mean. An analysis of 1,000 GTM engineer ads found postings up 205% in a year and a median salary of $127,500, with SQL and Python each named in 38% of ads. Most people in the job used to work in sales or revenue operations and taught themselves. Harvey, the legal AI company, hires legal engineers at $220,000 to $320,000, It asks for a law degree and at least three years in practice, and no technical background at all.

I'll use agentic engineering for the practice and agentic engineer for the person. Type the second phrase into a job board today and you won't get much back. The role is advertised under the builder titles and the domain-prefixed ones.

The job

An agentic engineer has four duties.

- Specify. Turn a business need into a written spec with a clear scope and acceptance tests drawn from real cases in the field.
- Guard. Set up the checks that run whether or not anyone is watching: tests, type checks, linters, CI gates, agent rule files, permission limits, sandboxes. Mitchell Hashimoto's rule applies: each time the agent makes a mistake, change the setup so it cannot make that mistake again.
- Verify. Make the agent prove its work with evidence. Read the diff for the known warning signs, which Kent Beck lists as tests deleted or disabled, and features nobody asked for. Then use the thing by hand.
- Own. Deploy it, watch the logs, handle the incident at 2am, and answer for both the security and the bill.

Typing the code isn't on that list. Stripe shows what takes its place. Its agents work on isolated machines with no access to production or the internet. Selected lints run on every push and finish in under five seconds. The test suite holds more than three million tests, and an agent gets two rounds of CI before a person steps in. Engineers built every part of that control system, and the control system is where the quality comes from.

Birgitta Böckeler's April 2026 article on martinfowler.com has the clearest words for the guard duty. Guides steer the agent before it acts. Sensors check the result afterwards, either by computation (tests, linters, type checkers) or by inference, where a second model reviews the first. She sorts these controls by what they protect. Existing tools already cover two of her groups well. The third is behaviour, meaning whether the software does the right thing, and by her own account its controls are the least developed.

That gap is where domain knowledge comes in. A linter can tell you a function is too long. It takes someone who knows the field to notice that a refund landed in the wrong tax period.

Does industry want this?

Four kinds of evidence say yes.

Hiring tests have been rewritten around it. Canva has required candidates to use AI tools in engineering interviews since June 2025, and grades them on breaking down vague requirements and on finding and

fixing problems in AI-generated code. Zapier requires AI fluency of every new hire, and the 2026 version of its rubric adds accountability for the output. Karpathy's advice to employers in April 2026: drop the small puzzles, hand the candidate a large project, and see whether they can make it good and secure.

The skills carry a wage premium. PwC's 2026 AI Jobs Barometer, built on more than a billion job ads in 27 countries, puts the premium for AI skills at 62%, up from 57% a year earlier. In Australia the count of ads asking for AI skills went from about 20,000 in 2024 to 41,000 in 2025. Lightcast found that 51% of ads asking for AI skills sit outside IT.

Firms are already doing it, mostly without a name for it. Gergely Orosz reported on 15 September 2026 that OpenAI's non-engineering teams, among them finance and legal, went from roughly zero to 90% use of its coding agent in four months, and that OpenAI now places subject experts inside engineering teams to set quality standards. Shopify bought 1,500 Cursor licences, then 1,500 more, and found its fastest-growing users were in Support and Revenue. Replit says Zillow staff built more than 7,000 apps on about 600 seats in a year. In Retool's February 2026 survey of 817 of its own users, 36% were engineers, 51% had shipped production software their teams use, and 60% had built something outside IT oversight. The last three are vendor figures, so take them as a direction rather than a measurement.

People are paying for the tools. Lovable reached $500 million in annualised revenue in June 2026 with about 146 staff, and says 80% of its builders describe themselves as non-technical.

Now the holes, which matter if you're planning a career or a hiring budget around this.

Nobody publishes a count of job ads that ask for agentic coding skills. The title at the top of LinkedIn's 2026 Jobs on the Rise list, AI Engineer, describes someone who builds products on top of language models. That's a different job.

For now, employers buy this judgement by hiring senior engineers. Indeed's Hiring Lab found that in the first quarter of 2026, 69.3% of US software development postings were senior and 4.5% were entry-level.

The evidence for domain experts is evidence about people building inside the job they already hold. I found no data on employers hiring non-programmers into engineering roles on the strength of their skill with agents.

Put together, employers clearly want the skills, and they aren't yet hiring for the title.

Why start from the domain expert

The largest dataset on this question came out in June 2026. Anthropic studied about 400,000 Claude Code sessions from about 235,000 people, October 2025 to April 2026, and scored each session on whether the person got what they set out to build, with hard evidence such as passing tests or a commit.

People in software jobs reached verified success in 34% of coding sessions. Everyone else reached it in 29%. All ten of the largest occupations landed within seven points of software engineers.

What separated success from failure was expertise in the task itself. The study's example is an accountant who can't write Python but can state reconciliation rules precisely. On that task, the accountant is the expert, and a senior engineer on day one in a new language counts as a novice. Task novices succeeded 15% of the time, against 28 to 33% for everyone above novice. They also abandoned 19% of sessions,

against 5 to 7%. The authors found that a working grasp of the domain captured most of the benefit, with deep specialisation adding a little more.

It has limits. This is a vendor studying its own product. The researchers couldn't see whether the code was ever used. And engineers still led by five points.

A security study fills in the other half. Research by the security firm Tenzai, reported by CSO Online, found coding agents weakest in two areas: business logic and authorisation. Business logic is where a domain expert is strongest. Authorisation is where an untrained one is blind. So the domain expert brings the first and has to learn the second.

The market already pays for domain-first hybrids. Harvey pays lawyers $220,000 and up to be legal engineers. One recruiter's 2026 profile of GTM engineers puts the median age at 25, with 53% self-taught, and the median salary in the ad analysis above is $127,500.

The strongest counter-example is the forward deployed engineer. The AI companies with the most funding fill their customer-facing hybrid role from the engineering side: Python appears in 66% of those ads, and 21% name a customer industry. Nobody has run the two staffing models against each other on software that shipped and was maintained. Nobody has measured which direction is faster to train, either.

So here's the claim I can defend. For internal tools, workflow software and the first version of a product for a specific industry, a trained domain expert will often beat a generalist engineer who has to learn the field first. That holds when the platform has safe defaults and an engineer independently checks security, data handling and operations. The most valuable person of all can verify both the domain answer and the software. I can't show that the domain expert always wins, and I don't claim it.

The objections

"AI makes experienced developers slower"

This is METR's July 2025 randomised trial: 16 experienced open-source developers, 246 tasks. Tasks took 19% longer with AI. The developers believed afterwards that they had been 20% faster.

The slowdown has aged badly. The trial used early-2025 tools. When METR reran it in February 2026 with 57 developers, so many refused to work without AI, or held back the tasks where AI helps most, that METR judged its own data compromised. It now thinks developers are probably faster with AI and says its evidence for the size of the gain is very weak.

The perception gap hasn't aged at all. People can't feel whether these tools are helping them. An agentic engineer has to measure cycle time and defect rates, and distrust the feeling of speed.

"More code, worse code, and nobody can check it all"

The data here is real. CodeRabbit compared 470 pull requests and found 10.83 issues per AI-assisted change against 6.45 for human ones. GitClear's January 2026 analysis of 623 million code changes found duplicated blocks up 81% and refactoring down from 21% of changed lines in 2022 to 3.8%. Faros AI measured 10,000 developers in 2025 and found merged pull requests up 98%, review time up 91%, bugs up 9%, and no improvement at company level.

These studies measure AI adoption with the old process left in place. Google's 2025 DORA report, with about 5,000 respondents, found that AI amplifies whatever a team already has: strong testing and version control get stronger results, weak ones get more instability.

Two controlled experiments point to process as the variable. A randomised trial with 151 professional developers found no significant difference in how easily other developers could later extend code written with AI help. A 2026 paper on test-driven agents found that telling the agent which tests a change affects cut regressions from 6.08% to 1.82%. Instructing it to follow test-driven development, with no further support, made regressions worse, at 9.94%. A specific guardrail helped, and a slogan on its own made things worse.

One part of the objection stands: teams are producing code faster than they can review it. Faros's 2026 figures, which I have seen only second-hand, suggest CI alone did not protect teams from the flood. OpenAI and Stripe are responding with review tiered by risk and with reviewing agents, and that work is young.

"It is insecure"

This is the hardest objection, and I can only partly answer it.

Veracode tests more than 150 models on security tasks. Its spring 2026 report puts the pass rate at 55%, flat for two years while the same models got much better at everything else. The flaws appear when the prompt gives no security guidance, which is what an untrained person provides by default.

The incidents match. In February 2026 Wiz found that Moltbook, a social network whose founder said "I didn't write a single line of code", had exposed 1.5 million API tokens and 35,000 email addresses because its database had no row-level access rules. A May 2026 scan by RedAccess found 380,000 publicly reachable assets built on AI app platforms, about 5,000 of them corporate and more than 2,000 holding sensitive data.

The answer I have is that the failures cluster in a short list: missing access rules, keys left in client code, hosting that is public by default, no separation between test data and live data. A short list can be taught, and most of it can be checked by machine. After its agent deleted a customer's live database in July 2025, Replit's fix was structural: development and production databases were separated so the mistake could not recur.

What I can't answer is that no test catches a control nobody thought to specify. So the role needs a threat checklist, scanners wired into CI, and a firm rule about when a specialist reviews the system before launch.

"You cannot judge what you cannot write"

Four respected voices make versions of this case. Addy Osmani's 70% problem describes non-engineers stalling on the last 30% because they cannot reason about architecture. Simon Willison holds that these tools amplify the expertise you already have. Andrew Ng wrote in August 2026 that a novice without software fundamentals lets the agent make bad trade-offs on reliability and cost, among others. And Anthropic's January 2026 randomised trial of 52 mostly junior engineers found that those who learned a new library with AI help scored 50% on a comprehension quiz against 67% for those who coded by hand, with the widest gap in debugging.

I accept all four. Look at what each one names as the missing ingredient, though. None of them names typing fluency. Ng's list has five entries, running from building full-stack applications and managing data through to operating systems in production. That is knowledge of how systems behave. My belief, which nobody has yet tested, is that it can be taught directly and faster than the years of syntax practice that used to carry it.

The Anthropic trial also found that how people used the AI decided how much they kept. Participants who asked it conceptual questions scored 65% or more. Those who handed it the writing scored under 40%. The habits follow from that: ask the agent to explain its work, walk through its code, and practise debugging with it switched off.

The large Anthropic dataset helps here too. The jump in success was from novice to intermediate, 15% to about 30%, with little gained beyond that. The bar is a working grasp, and that's reachable.

Charity Majors has moved on this too. She argued in 2024 that it takes seven-plus years to forge a competent engineer. By August 2026 she was asking what it would take to ship code that no person has read, and ranking line-by-line human review low among the things people add.

"We have heard this before"

COBOL was meant to let managers write programs. So were fourth-generation languages, CASE tools and low-code platforms. Each hit a ceiling where the problem outgrew the tool, and the work went back to programmers.

Two things differ this time. Agents write ordinary code in ordinary languages, so there is no platform ceiling, and an engineer can take over the codebase on any day. The second difference is pace: METR measures the length of task that frontier models can complete, and that length has been doubling roughly every three months since 2024. METR is clear that its tasks are self-contained and do not predict whole jobs.

The precedent that should worry us is the spreadsheet. It did move a specialist skill to domain experts, at enormous scale, and Ray Panko's research has found for decades that spreadsheet errors are common and material, and that organisations are overconfident about them. Spreadsheets spread with no training and no independent checking. Anyone in this role should have both from day one.

"Where will future experts come from?"

Stanford's Digital Economy Lab updated its payroll study in August 2026. Employment of 22 to 25 year olds in AI-exposed occupations is 19% below that of less-exposed peers, up from 15% a year earlier. The gap comes from reduced hiring, and the authors stress it is a description and not proof of cause.

I don't have a good answer to this. Kent Beck argues that a junior managed for learning can become productive in 9 months where it used to take 24, but that is his model and not data. The role described here opens a second door into software, through a field someone already knows. It does nothing to reopen the first.

"Who is liable?"

The EU's revised Product Liability Directive applies from 9 December 2026 and treats software as a product under strict liability. Reporting duties under the Cyber Resilience Act began on 11 September 2026. Neither asks who typed the code. The organisation that ships is liable.

That's an argument for the role. An agent can't be held to account. A named person who specified the work, built the checks and approved the release can.

Where this is the wrong choice

Multi-user systems that hold customer or regulated data. I found no independently audited case of a non-programmer running such a system reliably alone. Every published example I found of a product built with zero hand-written code, including OpenAI's, was run by engineers. Pair the domain expert with an engineer, or find a person with both skills.

Software where the domain is software. Databases, compilers, embedded firmware, high-scale infrastructure. Here the engineer is the domain expert.

Anyone who needs a job title to apply for this quarter. The ads use other names, and most hiring for this judgement still goes to senior engineers. The near-term opportunity for a domain expert is to build inside the job or business they already have.

What an agentic engineer has to know

I merged this list from practices published by working engineers and two research programmes. Simon Willison and Addy Osmani supplied most of the testing and review habits. Kent Beck supplied the warning signs. The guardrail material comes from Birgitta Böckeler and Mitchell Hashimoto, the operations material from Armin Ronacher, and the remainder from the engineering teams at Anthropic and OpenAI plus the DORA research.

- Framing and specs: stating the outcome, the non-goals and the acceptance tests, then cutting the work into small pieces that can each be reviewed.
- How software fits together: how a web request reaches a database, how logins and sessions work, and enough code literacy to follow a walkthrough of a diff.
- Testing and verification: test-first loops, always giving the agent a check it can run, manual exploration, spotting an agent that cheats.
- Guardrails: short context files, hooks that act as hard gates, custom linters, permission limits and sandboxes. Above all, the habit of turning each mistake into a permanent rule.
- Security and privacy: access rules, secrets, dependency risk, prompt injection against agents. Scanners in CI, and a rule for when to call a specialist.
- Data and structure: modelling the data, changing it safely with migrations, backing it up and testing the restore. Plain, stable technology choices, with scheduled clean-up of what the agents leave behind.
- Version control and delivery: small commits, easy rollback, CI, preview environments, a hard wall between test and live systems.
- Operations: logs the agent can read, monitoring, incident response, and cost control for both hosting and model usage.
- Orchestration: running agents in parallel, using one agent to review another, managing long-running jobs.

- Domain modelling: writing what you know about your field into worked examples and tests that an agent can be checked against.

The whole argument rests on that last item. Domain knowledge is worth nothing in this role until it's written down as something a machine can be checked against.

Two areas are close to absent from what practitioners have published: cost control, and the legal ground of licensing and privacy law. Anyone in the role will have to learn them with little published help.

What to do with this

If you hire: your domain experts are probably building already. Retool's 60% figure for tools made outside IT oversight suggests they are doing it without guardrails. Find them, train them, give them a platform with safe defaults, and put an engineer's security review between them and anything that touches customer data. Change your interview to match Canva's published rubric: real tools, a vague brief, and AI-written code with problems to find.

If you're a domain expert: the evidence says your knowledge of the field is the scarce part, and that a working grasp of software gets you most of the way. It also says novices succeed half as often as everyone else and give up three times as often. Close that gap deliberately, starting with testing and security, and build for the field you already know.

If you're an engineer: the four duties above are your job now, whatever your title says. The best long-term bet is to pick a field and learn it as deeply as you once learned a framework.

Sources

Role names and hiring data

- Andrej Karpathy, Sequoia Ascent 2026 summary, 30 April 2026: https://karpathy.bearblog.dev/sequoia-ascent-2026/
- Simon Willison, Agentic Engineering Patterns, from 23 February 2026: https://simonwillison.net/guides/agentic-engineering-patterns/
- Simon Willison, vibe coding and agentic engineering, 6 May 2026: https://simonwillison.net/2026/May/6/vibe-coding-and-agentic-engineering/
- a16z, services-led growth and the forward deployed engineer, 4 June 2025: https://a16z.com/services-led-growth/
- Gergely Orosz on forward deployed engineers: https://newsletter.pragmaticengineer.com/p/forward-deployed-engineers
- Indeed data on forward deployed engineer postings, via Business Insider, 18 May 2026: https://www.aol.com/articles/job-postings-tech-role-grown-185134480.html
- Plank, forward deployed engineer job market, 12 September 2026: https://joinplank.com/fde-job-market
- Bloomberry, 1,000 GTM engineer ads: https://bloomberry.com/blog/i-analyzed-1000-gtm-engineering-jobs-here-is-what-i-learned/

- Bloomberry, 1,000 forward deployed engineer ads: https://bloomberry.com/blog/i-analyzed-1000-forward-deployed-engineer-jobs-what-i-learned/
- Harvey, legal engineer role: https://www.harvey.ai/blog/day-in-the-life-legal-engineer
- Lenny's Newsletter on LinkedIn's Full Stack Builder, 4 December 2025: https://www.lennysnewsletter.com/p/why-linkedin-is-replacing-pms
- San Francisco Standard on builder titles, 5 March 2026: https://sfstandard.com/2026/03/05/engineer-2025-ai-land-everyone-s-builder-now/
- LinkedIn Jobs on the Rise 2026: https://www.linkedin.com/pulse/linkedin-jobs-rise-2026-25-fastest-growing-roles-us-linkedin-news-dlb1c
- Indeed Hiring Lab on seniority, 23 July 2026: https://hiringlab.indeed.com/2026/07/23/the-labor-market-is-tilting-toward-seniority/
- PwC 2026 Global AI Jobs Barometer: https://www.prnewswire.com/news-releases/ai-reshapes-global-labour-market-into-two-distinct-paths-rewarding-human-skills-pwc-2026-global-ai-jobs-barometer-302798987.html
- Lightcast, Beyond the Buzz, 23 July 2025: https://lightcast.io/resources/blog/beyond-the-buzz-press-release-2025-07-23

How leading firms build

- Sundar Pichai, Cloud Next 2026, 22 April 2026: https://blog.google/innovation-and-ai/infrastructure-and-cloud/google-cloud/cloud-next-2026-sundar-pichai/
- Stripe, Minions, 9 February 2026: https://stripe.dev/blog/minions-stripes-one-shot-end-to-end-coding-agents
- Gergely Orosz, Inside OpenAI's agentic software factory, 15 September 2026: https://newsletter.pragmaticengineer.com/p/openai-software-factory
- OpenAI, harness engineering, 11 February 2026: https://openai.com/index/harness-engineering/
- Birgitta Böckeler, Harness Engineering for Coding Agent Users, 2 April 2026: https://martinfowler.com/articles/harness-engineering.html
- Canva, AI in engineering interviews, 11 June 2025: https://www.canva.dev/blog/engineering/yes-you-can-use-ai-in-our-interviews/
- Bain, From Pilots to Payoff, 23 September 2025: https://www.bain.com/insights/from-pilots-to-payoff-generative-ai-in-software-development-technology-report-2025/
- DORA 2025 report, 24 September 2025: https://cloud.google.com/blog/products/ai-machine-learning/announcing-the-2025-dora-report
- Retool, build versus buy report, 17 February 2026: https://retool.com/blog/ai-build-vs-buy-report-2026

Domain expertise

- Anthropic, Agentic coding and persistent returns to expertise, 16 June 2026: https://www.anthropic.com/research/claude-code-expertise
- Fred Brooks, No Silver Bullet, 1986: https://www.cgl.ucsf.edu/Outreach/pc204/NoSilverBullet.html
- Standish Group, CHAOS report, 1994: https://personal.utdallas.edu/~chung/SYSM6309/chaos_report.pdf

- Andrew Ng, AI engineering skills map, software fundamentals, 28 August 2026: https://www.deeplearning.ai/the-batch/the-ai-engineering-skills-map-in-detail-software-engineering-fundamentals

Critics and counter-evidence

- METR trial, 10 July 2025: https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/
- METR update, 24 February 2026: https://metr.org/blog/2026-02-24-uplift-update/
- METR time horizons, 29 January 2026: https://metr.org/blog/2026-1-29-time-horizon-1-1/
- Faros AI, 23 July 2025: https://www.faros.ai/blog/ai-software-engineering
- CodeRabbit, 17 December 2025: https://www.coderabbit.ai/blog/state-of-ai-vs-human-code-generation-report
- GitClear, January 2026: https://www.gitclear.com/the_ai_code_quality_maintainability_gap
- Veracode, spring 2026: https://www.veracode.com/blog/spring-2026-genai-code-security/
- Wiz on Moltbook, 2 February 2026: https://www.wiz.io/blog/exposed-moltbook-database-reveals-millions-of-api-keys
- RedAccess scan, May 2026: https://securityboulevard.com/2026/05/thousands-of-vibe-coded-apps-exposing-corporate-personal-data-redaccess/
- Anthropic, How AI assistance impacts the formation of coding skills, 29 January 2026: https://www.anthropic.com/research/AI-assistance-coding-skills
- Addy Osmani, the 70% problem: https://addyo.substack.com/p/the-70-problem-hard-truths-about
- Borg, Farley et al., randomised trial on maintainability: https://arxiv.org/abs/2507.00788
- Test-driven agentic development paper, 2026: https://arxiv.org/abs/2603.17973
- Ray Panko on spreadsheet errors: https://arxiv.org/abs/1602.02601
- Stanford Digital Economy Lab, Canaries update, 12 August 2026: https://digitaleconomy.stanford.edu/news/canariesaug26/