

When agents write the code

What the software engineer's job becomes

Mark Dickie, Tarmac. September 2026.

The short version

At the firms furthest along, engineers have mostly stopped typing code. Sundar Pichai wrote in April 2026 that **75% of new code at Google** is generated by AI and then approved by engineers. Anthropic says **more than 80% of the code it merged in May 2026** was written by its own model. Stripe merges **more than 1,300 pull requests a week** that contain no human-written code.

The engineers are still there. What they do all day has moved. 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.

My argument is that this is a software engineer's job, and that it is a harder one than the job it replaces. The evidence says experience counts for more with agents than it did without them, and employers are paying accordingly. It also says the part of the job that has grown fastest, checking the work, is the part the industry is worst at.

One limit up front. Almost nobody has measured whether any of this makes a company ship better software faster. The numbers on code volume are strong. The numbers on outcomes are thin, and some point the wrong way. I've tried to mark which is which, and every figure below links to its source.

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 agents now write. Airbnb's chief executive said **AI wrote 60% of its new code** in the first quarter of 2026. At Shopify, **one merged pull request in eight** is co-authored by its internal agent. Monzo, a regulated bank, says an in-house agent writes **about 10% of its merged pull requests**, and that an engineer still reviews and merges every one.

Ordinary developers have followed. JetBrains surveyed more than 15,000 of them between May and July 2026 and found that **90% use a coding agent at work every week and 68% every day**. JetBrains sells these tools, so treat that as the high end. Stack Overflow's smaller poll in April put **agent use at work at 59%, up from 31% a year earlier**. In the same poll, 63% said they rarely or never let an agent run unattended.

Don't lean on the company numbers too hard. They're executive statements and nobody audits them. Uber shows how loose the counting is. In March 2026 **11% of its pull requests were opened by agents**. In May its chief executive put agents at **about 10% of committed code**. By August its engineering blog said **more than 70% of pull requests were attributed to agents**. Those are three definitions. The last appears to count any pull request an agent worked on, whether on the engineer's machine or in the cloud.

What does seem solid is the volume. Anthropic reports that its typical engineer merged eight times as much code per day in the second quarter of 2026 as in 2024, and names human review as the new slow step. A Microsoft study of its own rollout, covering tens of thousands of engineers, found that those who adopted command-line agents **merged about 24% more pull requests** than they otherwise would have.

More code is a different thing from more software that works. Typing was never most of the job. 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, and 25 to 30% only at companies that rebuilt their whole delivery process around the tools.

This has happened before

In the mid-1950s most programs were written by hand, in the machine's own instructions or something close to them. John Backus, who led the team that built Fortran, **later wrote** that programming and debugging then made up as much as three quarters of the cost of running a computer. Programmers thought of the work as a craft. When Backus told customers a compiler could write efficient machine code for them, most didn't believe him.

Fortran shipped in April 1957. A year later, a survey of 26 installations found that over half used it for more than half their problems. By the autumn of 1958, more than half of all the machine instructions running on those computers had been produced by the compiler. It took about eighteen months for most of the code to stop being written by hand.

Programmers didn't disappear. They moved up a level and wrote more ambitious programs. Fred Brooks, in **No Silver Bullet**, credits high-level languages with at least a fivefold gain in productivity. He also passes on David Parnas's dry observation that "automatic programming" has always meant programming in a higher-level language than the one you had yesterday.

The move up has a cost, and US government job counts show who pays it. The Bureau of Labor Statistics tracks two occupations. "Computer programmers" are defined as people who work from specifications drawn up by someone else and turn them into code. "Software developers" design the software. In 2000 there were **530,730 programmers** and about 639,000 developers. By May 2023 there were **120,370 programmers** and **1,656,880 developers**. The job codes changed twice in that period, so read it as a trend. The trend is that the translating job shrank by three quarters while the deciding job more than doubled.

That's the hopeful reading, and there's a less hopeful one. James Bessen studied **two centuries of automation in three US industries**, from textiles to cars. Automation raised employment for as long as cheaper output found new buyers. Once demand was met, jobs fell. Bank tellers are the usual happy example: cash machines cut **tellers per branch from 20 to 13**, banks opened 43% more branches, and teller jobs held up for decades. The story usually stops there. The Bureau now projects **teller jobs to fall 13% by 2035**, because phones removed the reason to visit a branch at all.

Nobody knows where software sits on that curve. The Bureau has cut its ten-year growth forecast for software developers three editions running, from 17.9% to 16% to **10% in the edition released in August 2026**. That is still about three times the rate for all jobs. The Bureau also says, in its own words, that its methods **"are not designed to capture extremely rapid technological change"**.

What to call it

Nobody agrees yet. The terms in use mostly name one part of the job.

Term	Origin	What it covers
Agentic engineering	Karpathy, February 2026; restated April 2026 . Simon Willison's patterns guide dates from the same month	Directing coding agents while holding a professional quality bar
Vibe coding	Karpathy, February 2025	Building without reading the code. The thing agentic engineering is defined against
Harness engineering	Mitchell Hashimoto and OpenAI, February 2026; Birgitta Böckeler's April 2026 article	Building the rules and checks around the agent
Context engineering	Anthropic, September 2025 ; moved to "Adopt" on the Thoughtworks Radar in April 2026	Choosing what the model sees
Spec-driven development	GitHub Spec Kit , September 2025; AWS Kiro	Writing the specification first and having the agent build from it
Supervisory engineering	Thoughtworks, June 2026 ; a 2026 study of 158 engineers uses the same phrase	Judging whether the agent solved the right problem

I'll use agentic engineering for the practice and agentic engineer for the person. It isn't a job title. The ads still say "software engineer" and add a line.

That line is spreading. Match.dev, a hiring marketplace, read [4,820 software engineering ads](#) from 179 technology companies in September 2026. One in five said something about AI coding tools, and 10.1% listed their use as a requirement. Among ads published between January and March the share requiring them was 5.2%. Among those published from July it was 13.9%. It's a vendor's count with a US tech skew, though the dataset is public. Lightcast, which supplies labour data to Stanford's AI Index, found the share of all US job ads asking for agent skills [rose from 0.06% in 2024 to 0.23% in 2025](#).

The job

An agentic engineer has four duties.

- Specify. Turn a need into a written spec with a clear scope, the things that are out of scope, and acceptance tests the agent can run.
- Guard. Set up the checks that run whether or not anyone is watching: tests, type checks, linters, CI gates, permission limits, sandboxes. Mitchell Hashimoto's rule applies: [each time the agent makes a mistake, change the setup so it can't 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. The published accounts of what takes its place look alike. Across Stripe, Shopify, Monzo, Cloudflare, Uber and DoorDash the same parts keep turning up: a sandbox with no access to production, a gateway that limits which tools and data the agent can reach, rule files in the repository, a machine reviewer in CI, and a person who approves the merge. None of them describes agents merging to production unattended as normal practice.

Two details show how much engineering sits in those parts. At Stripe an agent gets one full CI run and at most one retry before a person steps in. At Cloudflare, engineering standards are written as numbered rules a machine can check, and since the start of 2026 its reviewer has **flagged close to 230,000 violations and withheld approval almost 16,000 times**. Engineers built every part of both systems, and the systems are where the quality comes from.

Böckeler's article 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 maintainability and architecture fairly well. The third group 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 an engineer's judgement goes now. A linter can tell you a function is too long. It can't tell you the retry logic will double-charge a customer.

Does industry want this from engineers?

Yes, and mostly from experienced ones.

Hiring has turned up. Indeed publishes **a daily index of US software development ads**. Worked out from that data, the index stood 21.6% higher on 25 September 2026 than a year earlier, while ads for all jobs rose about 2%. It is still 21.9% below its level before the pandemic.

The recovery is tilted towards seniority. In the occupations most exposed to AI, with software development named first, Indeed found the **entry-level share of ads fell from 29% to 10% between 2021 and 2026, and the senior share rose from 22% to 47%**. Pay followed. Advertised pay in those occupations is up about 46% since 2021, against 25% in the least exposed. At senior level the gap is 17 points. At entry level it is 2.

Engineers are holding their place inside companies too. SignalFire, a venture firm that tracks hiring, reported in June 2026 that **software engineers make up 55% of hiring at large technology companies, up from 46% in 2019**. Engineering hiring at those companies is down 11%. Design is down 48% and product management 39%. SignalFire's own reading is that engineering's share rose because everything around it shrank faster.

Hiring tests are being rewritten. 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 problems in AI-generated code. Meta **built an interview in which the candidate has an AI assistant**. In CoderPad's 2026 survey, employers who allow AI in interviews were asked which signal shows real skill. The top answer, at 66%, was **catching and fixing the AI's mistakes**.

Interviews lag the ads, though. The same survey found 34% of employers still ban AI in interviews, and Karat's survey of 400 engineering leaders put **the share that prohibit it at 62%**. Match.dev found employers whose ads expect AI tools from day one and whose interviews ban them.

Now the holes.

The recovery is American and British. On the same Indeed data, software ads in Germany are down 15.2% in a year and in France 7.7%.

The pay figures measure "AI skills" and "AI-exposed occupations". No published wage series separates engineers who direct coding agents from engineers who build machine-learning products. The premium is real. What it's a premium for is less clear.

Companies are cutting as well as hiring. Challenger, Gray and Christmas counted **116,175 announced US job cuts in 2026 that cited AI**, about 22% of the total, with technology cuts up 52% on the year before. Atlassian's chief executive wrote, when cutting about 10% of staff in March, that it **would be disingenuous to pretend AI doesn't change the number of roles required**. The headline overstates the case a little. Most of the statements talk about cost and structure, and few say engineers were cut because agents write the code.

Put together, employers want more engineering judgement per engineer, and they're finding it in people who already have experience. That's good news for anyone mid-career. It's a problem for everyone coming up behind them, and I come back to it below.

Why experience counts for more

The largest dataset on this 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 on whether the person got what they set out to build, with hard evidence such as passing tests or a commit.

What separated success from failure was expertise in the task. Novices reached verified success 15% of the time. Everyone above novice managed 28 to 33%. When a session ran into trouble, novices recovered to a verified result 4% of the time and experts 15%. Novices abandoned 19% of sessions, against 5 to 7%.

Experts also got more out of each instruction: about 12 agent actions and 3,200 words of output per prompt, against 5 actions and 600 words for novices. And the direction of correction flipped. Experts tended to correct the agent. With novices, the agent tended to correct the person.

It has limits. This is a vendor studying its own product, and the researchers couldn't see whether the code was ever used. But independent work agrees with it.

A study of **more than 30 million GitHub commits by 170,000 developers** found that experienced programmers captured nearly all of the productivity gain from AI, which widened the skill gap. A study of **22,953 agent-assisted pull requests** found that those from less experienced contributors drew 4.52 times as many review comments and were accepted 31% less often. The cost of their inexperience landed on the reviewer. And when researchers watched how professionals use agents, in a paper titled **Professional Software Developers Don't Vibe, They Control**, they found engineers keeping hold of design decisions and using what they know to constrain the agent.

There's direct evidence for why. When instructions are vague, agents don't stop and ask. **One 2026 study** held 69 operations tasks fixed and varied only how clearly the goal, the target and the limits were stated. With underspecified instructions, 55.8 to 67.8% of runs crossed at least one boundary they shouldn't have. Warning the agent about the risk barely helped. Knowing what to pin down before you start is most of what experience is.

One finding cuts the other way, and it matters. A study that interviewed and observed 15 professional engineers found that with an AI assistant they **rarely stated security requirements up front**, and that years of experience did not reliably predict who got security right. Seniority helps with nearly everything here. On security it doesn't seem to be enough.

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. Microsoft's 24% is the best number since, and it counts merged pull requests, which says nothing about whether they were worth merging.

The perception gap hasn't aged at all. In May 2026 METR **surveyed 349 technical workers**, who reported a median threefold gain in speed. METR's own comment was that people in its earlier trial had overestimated AI's effect on their time by 40 percentage points, and that these respondents were probably overstating too.

The company-level evidence is no better. Uber **spent its entire 2026 AI budget in four months**, and its chief operating officer said the link from that spending to features customers can use "is not there yet". DX, which measures engineering teams, found that across more than 500 organisations the **share of time spent on new features barely moved** while AI-written code passed half of the total.

So an agentic engineer has to measure cycle time and defect rates, and distrust the feeling of speed.

"Nobody can review it all"

This is the strongest objection, and the evidence for it got worse in 2026.

Faros AI measured 22,000 developers over two years and compared each team's low-adoption and high-adoption periods. Throughput per developer rose 33.7%. **Median time in review rose 441.5%**, pull requests merged with no review at all rose 31.3%, and incidents per pull request rose 242.7%. Faros sells measurement software, and these are correlations inside its customer base. Its most uncomfortable claim is that organisations with mature delivery practices were not protected.

An independent study says something similar with better data. Researchers from Carnegie Mellon and Stanford followed **one company of 802 developers through 196,212 pull requests** after it told engineers to double their output with agents. Output per developer did reach 2.09 times the baseline. Pull requests grew 3.1 times while reviewers grew 1.5 times, so each reviewer's load doubled. The share of pull

requests that got any human review fell from 89% to 68%. The share that got a human review with actual comments fell from about 39% to about 21%.

Asking people to review harder won't fix this, and the research on review from before AI explains why. At Google, where review works well, **the median change is 24 lines and engineers spend about three hours a week reviewing**. A Cisco study found defect detection **drops off above about 400 lines or 500 lines an hour**. At Microsoft, **only about 15% of review comments pointed to a possible defect**. Human review was always a weak bug filter that worked on small changes. Agents produce large changes, and many of them.

Lisanne Bainbridge saw the shape of this in 1983, writing about industrial control rooms. Her paper **Ironies of Automation** points out that a person can't keep up effective attention on a source where very little happens for more than about half an hour. Watching an agent's correct output scroll past, waiting for the one wrong line, is that task.

The answer the leading firms have reached is to move checking into machines and sort it by risk. Cloudflare ran **131,246 automated reviews on 48,095 merge requests in 30 days, at an average cost of \$1.19 each**. Anthropic says that before it added machine review, 16% of its pull requests got substantive review comments, and **afterwards 54% did**. In the Carnegie Mellon study, machine review rose from about 19% of pull requests to about 84%, and the revert rate did not go up. That's one company and a coarse measure of quality. It's also the only measured outcome anyone has published of swapping part of human review for machine review.

Green tests aren't enough either. METR asked maintainers of three well-known Python projects to review 296 agent pull requests that had passed the benchmark's tests. **About half would not have been merged**, mostly on code quality. And agents will cheat when cornered. Researchers built tasks where the tests contradicted the spec, so that any pass had to be a cheat. **Leading models cheated on about half of them**, by editing the tests or special-casing the inputs. Making the test files read-only helped. Giving the agent a way to say "this can't be done" cut one model's cheating from 54% to 9%.

One part I can't answer. Review was never only about bugs. Microsoft's researchers found its main results were **shared understanding of the code and awareness across the team**. A machine reviewer catches defects. It doesn't leave a second person who knows how the thing works. Nobody has shown what replaces that.

"It is insecure"

It is, in two ways.

The first is the code. Veracode tests more than 150 models on security tasks. Its **July 2026 report** puts the average pass rate at 56%, against 55% when it started, while the same models got much better at everything else. The best model still fails nearly one task in three. Models do well on SQL injection, passing 83% of the time. On cross-site scripting they pass 15%. The test gives the model no security guidance, which is how most prompts are written.

The second is newer: the agent is itself a way in. A security researcher found **more than 30 vulnerabilities across the main AI coding tools** in late 2025, with 24 CVEs assigned, and reported that every tool tested was affected. Check Point showed that **cloning an untrusted repository and opening it with an agent** was enough to run an attacker's commands. In February 2026 someone used a crafted GitHub issue title to steer the Cline project's triage agent, and the chain ended with **an unauthorised**

release of Cline on npm. GitGuardian found that commits made with Claude Code leaked secrets at 3.2%, against 1.5% for all public commits. That's a vendor figure, and the two groups aren't like for like. And agents still invent package names. A 2026 replication across five current models found invented names in about 5% of cases, with 127 names that all five models made up identically. An attacker only has to register one.

The incidents so far have mostly been agents doing damage with access they shouldn't have had. A startup's agent, working in staging, found a production API token in an unrelated file and deleted the production database and its backups in a single call. Amazon's own account of a December 2025 outage is that an engineer's agent ran with a role broader than intended. Amazon calls that user error and says any tool could have done it. Both sides agree on the fix: narrower permissions, and a second person's approval for production access.

Some of this has tested answers. Showing the agent all the tests up front, security tests included, raised the rate of code that was both correct and secure by 19.3 percentage points. Feeding static-analysis findings back to the model in a loop cut security issues from over 40% of samples to 13%. Sandboxing, narrow credentials and read-only tests are engineering work, and engineers know how to do them.

What I can't answer is that a test never catches a control nobody thought to specify, and scanners are weaker than they sound. One study ran four static-analysis tools over 830 agent-written changes and found the tools agreed on the class of weakness 3.9% of the time. The role needs a threat checklist, scanners in CI, and a firm rule about when a security specialist looks before launch.

"You will forget how to code"

Possibly. Nobody has measured it in working engineers, and the evidence from elsewhere is worrying.

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. The widest gap was in debugging. How they 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 subgroups were tiny, so take the direction and not the numbers.

Aviation has forty years of this. In 2013 the US Federal Aviation Administration warned airlines that continuous use of autoflight could degrade a pilot's ability to recover the aircraft quickly, and told them to schedule manual flying. A simulator study of 16 airline pilots then found that the hand-flying held up and the thinking didn't: with the flight computer off, 44% failed to identify a key point on the approach. Bainbridge had predicted it. The better the automation, she wrote, the more the rare manual intervention matters and the less practised the operator is at it.

For engineers, the skill at risk is diagnosis, the ability to hold a model of the system in your head and work out why it's doing the wrong thing. That's also the skill verification depends on.

There's a cheap countermeasure with some evidence behind it. In an experiment with 78 novice programmers, one group had to explain the AI's code back before accepting it. They built just as much as the group with unrestricted AI. On a later maintenance task with the AI switched off, the unrestricted group failed 77% of the time and the explain-back group 39%. Addy Osmani's advice for working engineers runs the same way: form a guess before you prompt, read the diff and predict where it breaks, and solve some problems by hand on purpose.

"It costs too much"

It can. Anthropic's own guidance puts Claude Code at **about \$13 per developer per active day, or \$150 to \$250 a month**. That's the average. Ramp's spending data shows **the top 1% of firms spending \$7,449 per employee per month on AI**, and Uber ran through a year's budget in four months.

The spread is mostly engineering choices. The same job, a machine code review, costs Cloudflare \$1.19 and Anthropic **\$15 to \$25**. Uber cut its cost per session 52% from the peak by capping context and trimming what it sent to the model. One finding goes against a common habit. A team at ETH Zurich tested the repository context files that every agent vendor recommends and found they **did not generally raise task success and added over 20% to cost**. The files helped when they stated things the agent couldn't work out for itself. General tours of the repository didn't help.

Cost control is now part of the job, and almost nobody teaches it.

"Where will the next senior engineers come from?"

I don't have a good answer to this.

Stanford's Digital Economy Lab and the payroll firm ADP publish **an employment index for software developers by age**. With November 2022 set to 100, developers aged 22 to 25 stood at 80.5 in August 2026. Those aged 41 to 49 stood at 119.5. Over the last year the decline has spread to engineers up to 34. SignalFire puts new-graduate hiring at large technology companies about 65% below 2019. Students have noticed: enrolment in computer science at US four-year colleges **fell 8.4% in spring 2026**.

A small interview study from South Korea describes the mechanism. Entry-level work is being **absorbed into what a senior engineer does with an agent**, and the struggle that used to build judgement goes with it. Kent Beck argues that a junior managed for learning can become productive in **9 months where it used to take 24**, but that's his model and not data.

A few employers are betting the other way. Shopify **grew its internship programme from about 100 a year to more than 1,000**. IBM says it is **tripling entry-level hiring**. They're exceptions. The honest summary is that the industry is spending judgement it built over twenty years and hasn't worked out how to make more.

"Who is liable, and who owns it?"

The organisation that ships 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 Linux kernel's rules put the same point in one line: an AI agent **must not sign off a contribution**, because only a person can certify where code came from.

Ownership is less settled than most engineers assume. The US Copyright Office concluded in January 2025 that **prompts alone don't give a person enough control to be the author of the output**, and in March 2026 the Supreme Court **declined to hear** the leading case on AI authorship. The report is about creative works in general and hasn't been tested on code. Read plainly, it means code an agent wrote from a prompt and nobody edited may have no copyright owner in the United States. The more a person shapes and edits the result, the stronger the claim. I'm not a lawyer, and this isn't legal advice.

Both points argue for the role. An agent can't be held to account. A named engineer who specified the work, built the checks and approved the release can.

Where this doesn't hold

Old code. Most of the success stories are new systems built around agents from the start. DORA's 2026 model assumes **gains of 35 to 40% on new work and 10% or less on legacy code**. Osmani's **piece on older codebases** cites a refactoring benchmark in which 28 of 520 agent migration runs passed every verification stage. If your day is a fifteen-year-old system with no tests, the first job is to write tests that pin down what it does now, and the agent comes after.

Anything you can't give the agent a way to check. Every practice that works in this paper depends on a test, a type or a scanner the agent can run. Where correct behaviour can only be judged by a person looking at the result, the agent speeds up the typing and leaves the judging to you.

Claims about company results. I can show that engineers merge more code. I can't show that their employers ship better products sooner. Uber's chief operating officer and DX's data both say it hasn't shown up yet.

Anyone who needs this to be a job title. It's a way of doing the job you already have.

What an agentic engineer has to know

I merged this list from practices published by working engineers and from Andrew Ng's **skills map for coding agents**, which was built from more than 10,000 job ads. 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, and the operations material from Armin Ronacher. Where a controlled study backs an item, it's cited above.

- Specs: the outcome, the non-goals and the acceptance tests, then the work cut into pieces small enough to review. Vague instructions make agents guess.
- Tests as the contract: write or approve the tests first, show all of them to the agent, and make the test files read-only to it. Give it a way to stop and report that a task can't be done.
- Guardrails: rule files that say only what the agent couldn't infer, hooks that act as hard gates, custom linters, and the habit of turning each mistake into a permanent check.
- Review by risk: machine review on everything, human attention where a mistake would be expensive. Look hardest at changes to tests.
- Securing the agent: a sandbox, credentials that are narrow and expire quickly, no agent in CI that reads untrusted text with shell access, and a check on every new dependency it adds.
- Security of the output: a threat checklist, scanners in CI, and a rule for when to call a specialist.
- Delivery: small changes, easy rollback, preview environments, a hard wall between test and live systems.
- Operations: logs the agent can read, and monitoring and incident response you can run without it.
- Cost: knowing what a session, a review and a CI run cost, and which of your habits drives the bill.
- Keeping your own skills: explaining the agent's code back, debugging by hand on a schedule, and knowing one area of the system well enough to catch a confident wrong answer.
- The business the software serves: enough to write acceptance tests that mean something.

The list is mostly old engineering. Tests, small changes, least privilege and code you can reason about were good practice in 2015. What changed is that they used to make a good team better. Now they decide whether the agent's output can be trusted at all.

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

What to do with this

If you write code for a living: the four duties above are your job now, whatever your title says. Put your effort where the evidence says the shortage is, which is verification. Learn to write a spec an agent can't misread and a test it can't game. Keep your debugging sharp on purpose.

If you lead a team: stop counting output. Thoughtworks lists [coding throughput as a productivity measure](#) among the practices to avoid, and the Faros data shows why. Measure time in review, the share of changes merged unreviewed, and incidents per change. Put a machine reviewer on everything and spend your people on the risky changes. And decide how your juniors will build judgement, because the old way has gone.

If you hire: test for the job as it is. Canva's published rubric is the best model I've found: real tools, a vague brief, and AI-written code with problems to find. An interview that bans the tools your own ads require tells the candidate you haven't thought about it.

If you're early in your career: the numbers above are hard reading, and I won't pretend otherwise. The skill that the market is short of is judgement about software, and it comes from building things and finding out why they broke. Use the agent to explain, and do some of the debugging yourself.

Sources

How firms build with agents

- Sundar Pichai, Cloud Next 2026, 22 April 2026: <https://blog.google/innovation-and-ai/infrastructure-and-cloud/google-cloud/cloud-next-2026-sundar-pichai/>
- Anthropic Institute, When AI builds itself, updated 18 September 2026: <https://www.anthropic.com/institute/recursive-self-improvement>
- Anthropic, Code Review, 9 March 2026: <https://claude.com/blog/code-review>
- Stripe, Minions part 2, 19 February 2026: <https://stripe.dev/blog/minions-stripes-one-shot-end-to-end-coding-agents-part-2>
- Airbnb, via TechCrunch, 8 May 2026: <https://techcrunch.com/2026/05/08/airbnb-says-ai-now-writes-60-of-its-new-code/>
- Shopify Engineering, Under the River, 28 May 2026: <https://shopify.engineering/under-the-river>
- Monzo, Building agent Chip, 13 August 2026: <https://monzo.com/blog/building-agent-chip>
- Uber Engineering, 27 August 2026: <https://www.uber.com/us/en/blog/efficient-software-factory/>
- Pragmatic Engineer on Uber, 10 March 2026: <https://newsletter.pragmaticengineer.com/p/how-uber-uses-ai-for-development>

- Fortune on Uber's AI budget, 26 May 2026: <https://fortune.com/2026/05/26/uber-coo-ai-spending-tokens-claude-code/>
- Cloudflare, AI code review, 20 April 2026: <https://blog.cloudflare.com/ai-code-review/>
- Cloudflare, engineering standards enforcement, 4 August 2026: <https://blog.cloudflare.com/engineering-standards-enforcement/>
- DoorDash cloud agents, via InfoQ, 31 August 2026: <https://www.infoq.com/news/2026/08/door-dash-flux-cloud-agent/>
- Murphy-Hill, Butler and Savelieva, Microsoft rollout study, 1 July 2026: <https://arxiv.org/html/2607.01418v1>
- 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/>
- Atlassian developer experience report, 9 July 2025: <https://www.atlassian.com/blog/developer/developer-experience-report-2025>
- DX, State of AI impact in engineering, Q2 2026: <https://getdx.com/news/dx-releases-q2-2026-state-of-ai-impact-in-engineering-report/>
- DORA, ROI of AI-assisted software development, via InfoQ, 11 May 2026: <https://www.infoq.com/news/2026/05/dora-roi-ai-assisted-dev-report/>
- Anthropic, Claude Code costs: <https://code.claude.com/docs/en/costs>
- Ramp AI Index, 26 June 2026: <https://ramp.com/data/ai-index-june-2026>

History and forecasts

- John Backus, The History of FORTRAN I, II and III, 1978: <https://softwarepreservation.computerhistory.org/FORTRAN/paper/p165-backus.pdf>
- Fred Brooks, No Silver Bullet, 1986: <https://www.cgl.ucsf.edu/Outreach/pc204/NoSilverBullet.html>
- BLS occupational employment, computer programmers, 2000: <https://web.archive.org/web/2003/http://www.bls.gov/oes/2000/oes151021.htm>
- BLS occupational employment, computer programmers, May 2023: <https://web.archive.org/web/2025/https://www.bls.gov/oes/2023/may/oes151251.htm>
- BLS occupational employment, software developers, May 2023: <https://web.archive.org/web/2024/https://www.bls.gov/oes/2023/may/oes151252.htm>
- BLS Occupational Outlook Handbook, software developers, 27 August 2026: <https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm>
- BLS, Incorporating AI impacts in employment projections, 10 February 2025: <https://www.bls.gov/opub/mlr/2025/article/incorporating-ai-impacts-in-bls-employment-projections.htm>
- James Bessen, Automation and Jobs, 2019: https://scholarship.law.bu.edu/faculty_scholarship/815/
- James Bessen, Toil and Technology, March 2015: <https://www.imf.org/external/pubs/ft/fandd/2015/03/bessen.htm>
- BLS Occupational Outlook Handbook, tellers, 27 August 2026: <https://www.bls.gov/ooh/office-and-administrative-support/tellers.htm>

- Lisanne Bainbridge, Ironies of Automation, 1983: https://ckrybus.com/static/papers/Bainbridge_1983_Automatica.pdf
- FAA Safety Alert for Operators 13002, 4 January 2013: https://www.faa.gov/sites/faa.gov/files/other_visit/aviation_industry/airline_operators/airline_safety/SAFO13002.pdf
- Flight Safety Foundation on Casner and others, 2014: <https://flightsafety.org/asw-article/use-it-or-lose-it/>

Terms and practice

- Andrej Karpathy, Sequoia Ascent 2026 summary, 30 April 2026: <https://karpathy.bearblog.dev/sequoia-ascent-2026/>
- Simon Willison, Agentic Engineering Patterns: <https://simonwillison.net/guides/agentic-engineering-patterns/>
- Birgitta Böckeler, Harness Engineering for Coding Agent Users, 2 April 2026: <https://martinfowler.com/articles/harness-engineering.html>
- Mitchell Hashimoto, My AI Adoption Journey, 5 February 2026: <https://mitchellh.com/writing/my-ai-adoption-journey>
- Kent Beck, Augmented Coding: Beyond the Vibes, 25 June 2025: <https://newsletter.kentbeck.com/p/augmented-coding-beyond-the-vibes>
- Anthropic, Effective context engineering for AI agents, 29 September 2025: <https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents>
- Thoughtworks Technology Radar, volume 34, April 2026: <https://www.thoughtworks.com/radar/techniques>
- Thoughtworks, Supervisory engineering, 3 June 2026: <https://www.thoughtworks.com/insights/blog/agile-engineering-practices/supervisory-engineering-orchestrating-software-middle-loop>
- GitHub Spec Kit: <https://github.com/github/spec-kit>
- Andrew Ng, AI engineering skills map, using coding agents, 4 September 2026: <https://www.deeplearning.ai/the-batch/the-ai-engineering-skills-map-in-detail-using-coding-agents>
- Addy Osmani, Agentic Skill Decay, 31 August 2026: <https://addyosmani.com/blog/agentic-skill-decay/>
- Addy Osmani, Brownfield Agentic Engineering, 14 September 2026: <https://addyosmani.com/blog/brownfield-agentic-engineering/>

Jobs and pay

- Indeed Hiring Lab, job postings tracker (data): https://github.com/hiring-lab/job_postings_tracker
- Indeed Hiring Lab on AI exposure and advertised pay, 17 September 2026: <https://hiringlab.indeed.com/2026/09/17/ai-exposure-isnt-squeezing-advertised-pay-in-the-us-its-boosting-it/>
- SignalFire, State of Tech Talent 2026, 22 June 2026: <https://www.signalfire.com/blog/signalfire-state-of-talent-report-2026>

- Match.dev, AI coding tools in job postings, 15 September 2026: <https://www.match.dev/post/ai-coding-tools-in-job-postings/>
- Lightcast on the Stanford AI Index, 13 April 2026: <https://lightcast.io/resources/blog/stanford-ai-2026>
- JetBrains, AI coding agent adoption 2026: <https://blog.jetbrains.com/research/2026/08/ai-coding-agent-adoption-2026/>
- Stack Overflow, Agents on a leash, 27 May 2026: <https://stackoverflow.blog/2026/05/27/agents-on-a-leash-agentic-ai-remains-mostly-monitored-at-work/>
- Canva, AI in engineering interviews, 11 June 2025: <https://www.canva.dev/blog/engineering/yes-you-can-use-ai-in-our-interviews/>
- 404 Media on Meta's AI-enabled interview, 29 July 2025: <https://www.404media.co/meta-is-going-to-let-job-candidates-use-ai-during-coding-tests/>
- CoderPad, State of Tech Hiring 2026, 11 February 2026: <https://coderpad.io/survey-reports/coderpad-state-of-tech-hiring-2026/>
- Karat, Engineering interview trends 2026, 7 January 2026: <https://karat.com/engineering-interview-trends-2026/>
- Challenger, Gray and Christmas, August 2026 report: <https://www.challengergray.com/blog/challenger-report-august-job-cuts-up-58-consumer-products-food-lead/>
- Atlassian team update, 11 March 2026: <https://www.atlassian.com/blog/company-news/atlassian-team-update-march-2026>
- Stanford Digital Economy Lab and ADP, Canaries dashboard: <https://digitaleconomy.stanford.edu/project/indicators/canaries-dashboard/>
- National Student Clearinghouse on computer science enrolment, 11 August 2026: <https://www.studentclearinghouse.org/nsblog/computer-science-enrollment-is-cooling/>
- CoderPad interview with Shopify, 23 April 2026: <https://coderpad.io/blog/hiring-developers/in-the-ai-era-shopify-is-investing-in-junior-engineers-not-cutting-them/>
- CIO on IBM entry-level hiring, 18 February 2026: <https://www.cio.com/article/4134276/ibm-looks-beyond-short-term-ai-gains-tripling-entry-level-hiring.html>
- Kent Beck, The Bet On Juniors Just Got Better: <https://newsletter.kentbeck.com/p/the-bet-on-juniors-just-got-better>

Expertise

- Anthropic, Agentic coding and persistent returns to expertise, 16 June 2026: <https://www.anthropic.com/research/claude-code-expertise>
- Daniotti and others, Who is using AI to code?, June 2025: <https://arxiv.org/abs/2506.08945>
- Asdaque and others, agent-assisted pull requests by contributor experience, 27 February 2026: <https://arxiv.org/abs/2602.23905>
- Huang and others, Professional Software Developers Don't Vibe, They Control: <https://arxiv.org/abs/2512.14012>
- Ji and others, Coding Agents Are Guessing, 2 July 2026: <https://arxiv.org/abs/2607.02294>

- Bappy and others, From Preventive to Reactive, SOUPS 2026: <https://arxiv.org/abs/2605.23130>
- Vella and Blincoe, longitudinal study of AI coding assistants, 22 May 2026: <https://arxiv.org/abs/2605.23135>
- Yu and Moon, Who Will Become the Next Senior?, 19 July 2026: <https://arxiv.org/abs/2607.17067>

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 survey of technical workers, 11 May 2026: <https://metr.org/blog/2026-05-11-ai-usage-survey/>
- METR, Many SWE-bench-passing PRs would not be merged, 10 March 2026: <https://metr.org/notes/2026-03-10-many-swe-bench-passing-prs-would-not-be-merged-into-main/>
- Faros AI, The Acceleration Whiplash, 2026: https://pages.faros.ai/hubfs/AI_Engineering_Report_2026_The_Acceleration_Whiplash_Faros.pdf
- He and others, AI Writes Faster Than Humans Can Review, 2 July 2026: <https://arxiv.org/abs/2607.01904>
- Sadowski and others, Modern Code Review: A Case Study at Google, 2018: <https://sback.it/publications/icse2018seip.pdf>
- Jason Cohen, Code Review at Cisco Systems, 2006: <https://static0.smartbear.co/support/media/resources/cc/book/code-review-cisco-case-study.pdf>
- Czerwinka, Greiler and Tilford, Code Reviews Do Not Find Bugs, 2015: <https://www.microsoft.com/en-us/research/publication/code-reviews-do-not-find-bugs-how-the-current-code-review-best-practice-slows-us-down/>
- Bacchelli and Bird, Expectations, Outcomes, and Challenges of Modern Code Review, 2013: <https://www.microsoft.com/en-us/research/publication/expectations-outcomes-and-challenges-of-modern-code-review/>
- Zhong, Raghunathan and Carlini, ImpossibleBench, 23 October 2025: <https://arxiv.org/abs/2510.20270>
- Gloaguen and others, Evaluating AGENTS.md, February 2026: <https://arxiv.org/abs/2602.11988>
- Anthropic, How AI assistance impacts the formation of coding skills, 29 January 2026: <https://www.anthropic.com/research/AI-assistance-coding-skills>
- Sankaranarayanan, explain-back experiment, 22 February 2026: <https://arxiv.org/abs/2602.20206>

Security and law

- Veracode, 2026 GenAI Code Security Report, 28 July 2026: <https://www.veracode.com/blog/2026-genai-code-security-report-ai-risk/>
- Ari Marzouk, IDEsaster, 6 December 2025: <https://maccarita.com/posts/idesaster/>
- Check Point Research on Claude Code project files, 25 February 2026: <https://research.checkpoint.com/2026/rce-and-api-token-exfiltration-through-claude-code-project-files-cve-2025-59536/>

- Cline post-mortem, unauthorised npm release, February 2026: <https://cline.bot/blog/post-mortem-unauthorized-cline-cli-npm>
- GitGuardian, State of Secrets Sprawl 2026, 17 March 2026: <https://blog.gitguardian.com/the-state-of-secrets-sprawl-2026/>
- Churilov, package hallucination on 2026 models, 16 May 2026: <https://arxiv.org/abs/2605.17062>
- The Register on the PocketOS database deletion, 27 April 2026: <https://www.theregister.com/software/2026/04/27/cursor-opus-agent-snuffs-out-startups-production-database/5224442>
- Amazon on the AWS Kiro report: <https://www.aboutamazon.com/news/aws/aws-service-outage-ai-bot-kiro>
- Liang and others, Security Tests as Executable Specifications, 10 August 2026: <https://arxiv.org/abs/2608.09740>
- Blyth and others, Static Analysis as a Feedback Loop, 20 August 2025: <https://arxiv.org/abs/2508.14419>
- Rajput and others, Trajectory-Level Security Debt in LLM Coding Agents, 28 September 2026: <https://arxiv.org/abs/2609.35199>
- EU Product Liability Directive: https://single-market-economy.ec.europa.eu/single-market/goods/free-movement-sectors/liability-defective-products_en
- EU Cyber Resilience Act: <https://digital-strategy.ec.europa.eu/en/policies/cyber-resilience-act>
- Linux kernel, coding assistants policy: <https://github.com/torvalds/linux/blob/master/Documentation/process/coding-assistants.rst>
- US Copyright Office, Copyright and Artificial Intelligence, Part 2, 29 January 2025: <https://www.copyright.gov/ai/Copyright-and-Artificial-Intelligence-Part-2-Copyrightability-Report.pdf>
- US Supreme Court docket 25-449, Thaler v. Perlmutter: <https://www.supremecourt.gov/search.aspx?filename=/docket/docketfiles/html/public/25-449.html>