

2026 Vibe Coding Meta Study

Your employees are building the company's software.

What the evidence says about finishing it safely.

A meta-study of 34 research papers, security scans, vendor reports, and public arguments about enterprise vibe coding.

Version 7.0 | August 2026 | JetBridge

Abstract

Employees have begun building their own production software by describing what they want to an artificial intelligence in plain language, a practice the industry calls vibe coding. It moved from hobbyist novelty to enterprise use in roughly sixteen months. This review asks what the published evidence actually shows about that practice: the value it creates, the defects it leaves behind, and the work required to finish what it starts.

We examined **34** sources published between February 2025 and August 2026, drawn from six categories: academic research, standards-body notes, security scans of live applications, industry surveys, platform vendor publications, and public argument. For each source we recorded the date, the method and sample size, the unit of analysis, the population studied, the access model, the publisher's commercial interest, and whether the source prescribed a remedy or only named a risk.

The field turns out to be unusually consistent on diagnosis and almost silent on remedy. Four independent teams, using different methods on different samples, found that most vibe-coded applications carry security defects, and those defects concentrate in a single class: access control at the data layer, which is the one property the person building the application cannot observe. The only randomised controlled trial in the field found participants working **19%** slower while believing themselves **20%** faster, a result that disqualifies self-reported productivity as evidence for anything.

Two gaps limit every conclusion here. No rigorous study samples the population that actually generates the enterprise risk, which is the employee who is not a developer, and no study measures build speed and total cost of ownership on the same population, so no honest return on investment can yet be calculated for this practice.

We conclude that employee-built software is durable and worth encouraging, that the completion work is real, itemisable and largely non-optional, and that a standards-level control framework for the non-developer builder does not yet exist as off the shelf software, and that mid-market and enterprises still need human expertise to organize data flows and create a secure perimeter around vibe coded apps. Section 7 converts the findings into a twelve-item specification of that completion work. Section 8 examines the delivery model the industry has adopted to meet it.

DISCLOSURE

Meta study conducted by JetBridge, a software engineering company and makers of the enterprise vibe coding tool AirCode. A JetBridge principal scientist also conducted AI research for METR, whose study is cited in this paper.

The Cheat Sheet

What to do, what to fear, what the numbers say.

1. The checklist — what to do, and what to demand from a vendor

Require these in the platform

Control
☐ Access control enforced at the data layer, not on the screen, and applied by default to every new app.
☐ Company single sign-on inherited automatically, so leavers lose access to every app at once.
☐ Connectors that carry the user's own permissions.
☐ Secrets held by the platform. The builder never sees, pastes or stores a key.
☐ Audit logging on by default, with retention and export, so an incident can be reconstructed.
☐ An automatic inventory of every app, each with a named owner.
☐ A gate at launch, not at build: control applies when an app touches real data or gains a second user.
☐ Dependency verification in the pipeline.
☐ Versioning, tests and a one-click path back to the last version that worked.

Ask these four separately in the contract — vendors answer one and let you assume the rest

Question
☐ Residency. Which country, and whose hardware, does our data sit on?
☐ Ownership. Who owns the model weights, and can we keep operating if you withdraw the model?
☐ Control. Who can switch it off, and who can read the logs?
☐ Training rights. Do our prompts and our data improve a model our competitors then use?

Run these in your own programme

Action
☐ Find what already exists — check expense reports and identity logs for build-platform activity.
☐ Declare an amnesty. No penalty for registering an existing app.
☐ Tier apps by data sensitivity and apply heavy controls only to the top tier.
☐ Budget expert time to finish apps.
☐ Measure delivered outcomes — defect rate, time to fix, apps with no owner.

2. What is new — the threats vibe coding introduces

Most of these evade the controls you already run, either because they appear after code review, or because nothing about them looks wrong.

Threat	Why your existing controls miss it
Broken access control at the data layer	The most common failure by a wide margin. A public key with no row-level policy behind it. The app works perfectly for the builder and equally well for a stranger, with no visible symptom.
Authentication in the browser	The password is checked on the user's own machine, where anyone can remove the check.
Hardcoded secrets in shipped code	Keys and credentials sit inside files the browser downloads. Nobody reviewed the code, so nobody saw them.
Unauthenticated internal apps	Dashboards and staging systems published straight to the open internet, with no login at all.
No audit trail	Apps ship without logging, so when something goes wrong there is no record of who saw which record, who changed what, or when.
Slopsquatting	Roughly a fifth of AI package references are invented. Attackers register the invented names and fill them with malware.
Compromised legitimate dependencies	Trusted packages, not fake ones. Shai-Hulud 2.0 executed during installation and harvested cloud credentials before the app had ever been run.
Prompt and IP leakage	The instruction describing how you price, triage or qualify is more valuable than the code. On public platforms it goes to a model vendor under terms nobody negotiated.
Orphaned applications	The builder leaves; the app keeps running on live credentials, with a business process depending on it and no owner to call.
Defect persistence	Faults introduced by AI are frequently never removed. They become a permanent maintenance liability rather than a bug queue.

3. The numbers that matter

Drawn from the **34**-source corpus. Full method, sample sizes and commercial interests are in Appendix A.

Number	What it measures	Source
98%	of 1,072 live vibe-coded apps had at least one vulnerability; 16% had a critical one	Symbiotic Security, Jun 2026
0 of 5,600	production apps had CSRF protection, security headers or properly scoped access policies	Cloud Security Alliance, Jun 2026
45%	of AI-generated code samples introduce an OWASP Top 10 vulnerability, with no improvement over time	Veracode, 100+ models
1 in 5	organisations building on these platforms is exposed	Wiz Research
3–4× / 10×	faster delivery, at ten times the vulnerability rate	Cloud Security Alliance
22.7%	of the defects AI introduces are still present in the current version — never fixed	302,600 commits, Mar 2026
19% / 20%	measured slower, believed faster: the only randomised trial in the field	METR RCT, Jul 2025
66%	of developers say AI output is “almost right, but not quite”	Stack Overflow, n= 49,009
76%	of developers reject vibe coding as a practice, while 78.5% use AI tools monthly	Stack Overflow, n= 49,009
1,275,105	leaked AI service keys in 2025, an increase of 81% in one year	GitGuardian
~20%	of AI package references point to software that does not exist	Cloud Security Alliance
~800	legitimate npm packages compromised, yielding 775 GitHub tokens and 788 cloud credentials	Shai-Hulud 2.0 , Nov 2025
75% / 78%	of employees use AI tools their employer never approved, and brought their own	Microsoft WorkLab
16 months	from fringe technique to broad enterprise adoption	Cloud Security Alliance
\$4.63M	average cost of a breach involving shadow AI	IBM Cost of a Data Breach

60 Second Overview

Your employees have started building their own software, describing what they want in plain English and letting an AI assemble it. The industry calls this vibe coding, and it took roughly **16** months to travel from hobbyist novelty to something running inside real companies.

They are building the right things, largely because the person with the problem writes a better specification than any analyst sent to interview them. That part of this is working well.

What they are not doing is finishing. The AI builds the screens and then stops, leaving out everything that keeps data private, auditors satisfied and the application alive a year from now.

98%

of 1,072 live apps built this way had at least one security flaw

Symbiotic Security scan, June 2026

0%

of 5,600 production apps had properly scoped access controls

Cloud Security Alliance, June 2026

The flaw is almost always the same one: the application never checks whether the person using it is allowed to see the data. Your employee has no way of catching this, because they can see that their app works but not that it works equally well for a stranger.

They also ship without audit logs, so when something does go wrong there is no record of who saw which record, who changed what, or when.

THE BOTTOM LINE

Let your employees keep building, but assume someone with expert skills has to finish what they start. Section 7 sets out what finishing involves, and Section 8 explains who has been doing it.

How to Read This

If you run the business — 10 minutes

Read the previous page, then read only the shaded IN PLAIN ENGLISH box that opens each section and the bold BOTTOM LINE bar that closes it. Both are written to stand on their own, so you can skip the technical text in between without losing the thread.

If you run the technology — 25 minutes

Read all of it. The document is built to replace **34** separate papers, scans and reports: where they agree we say so, where they conflict we explain why, and where nobody has measured anything at all we say that too.

- **Part One (Sections 1–6).** What is happening, what it costs, and what is at stake.
- **Part Two (Sections 7–9).** What to do about it, including the specification worth photocopying.
- **Part Three (Sections 10–15).** How we know, what the evidence cannot tell you, and our method.
- **Section 7.** The twelve-item specification, which is the section worth photocopying.
- **Section 8.** Who does that work, and why the delivery model changed.
- **Section 9.** What to do on Monday.
- **Section 13.** Where this evidence is weak, ours included.
- **Appendix A.** Every source, with its date, its sample size, and what its publisher sells.

Contents

The Cheat Sheet

The checklist, the new threats, and the numbers — all on three pages

PART ONE · What Is Happening, and What It Costs

1. Your Best Requirements Are Walking Around the Building

Finding 1 — why employees build the right thing, and why bans fail

2. Almost Every App Has the Same Hole

Finding 2 — four studies, four methods

3. The Bill Did Not Vanish.

Finding 3 — writing got cheap, checking got expensive

4. Nobody Can Feel Their Own Productivity

Finding 4 — 19% slower, and certain they were faster

5. The Sentence Your Employee Types Is the Asset

Finding 5 — who owns the instruction, and the four questions inside "sovereign AI"

6. Everyone Names the Disease. Nobody Writes the Prescription.

Finding 6 — no framework covers your employees

PART TWO · What to Do About It

7. The Twelve Things AI Will Not Do

The specification, and the most useful page in this document

8. The Rise of the Forward Deployed Engineer

Who does that work

9. What to Do on Monday

Classify, move the controls into the platform

PART THREE · How We Know, and What We Cannot Know

10. A Weekend Hobby Went Into Production

Where the term came from, and why the definition still matters

11. Both Sides Are Right. They Are Timing Different Races.

Finding 7 — why vendor and research numbers disagree

12. The Studies Are Watching the Wrong People

Finding 8 — the largest hole in the evidence

13. What This Evidence Cannot Tell You

Four weaknesses

14. Who Wrote This, and What We Sell

Our commercial interest, and where we are the wrong answer

15. How We Read the Field

The method: what we included, how we scored it

Appendix A — The Sources

All 34, with dates, sample sizes and commercial interests

Appendix B — Glossary

Plain definitions of every technical term used

Appendix C — Method Notes

How sources were selected

How This Whitepaper Was Drafted

Authorship, tools used, and how to reach us

PART ONE

What Is Happening, and What It Costs

FINDING 1

1. Your Best Requirements Are Walking Around the Building

The employee with the problem writes a better specification than anyone you could hire to write it. This is why banning the practice fails.

IN PLAIN ENGLISH

Your employees are building small tools to fix the problems they personally run into every day, the ones that were always too small to reach the top of IT's list. They tend to build the right thing, because they are the person with the problem. You also cannot really stop them, since the tool is a browser tab and a credit card. Ban it and they will carry on from personal email, and all you will have achieved is losing sight of it.

Adoption was unusually fast. The Cloud Security Alliance clocked the move from fringe technique to broad enterprise use at roughly **16** months, which is not a speed enterprise practices normally manage.

A vendor survey of **50** executives found that half of enterprises already have non-engineering teams building internal applications. The sample is small, but it points the same way as the shadow-AI figures: Microsoft found **75%** of employees using AI tools their employer never approved, and **78%** of workplace AI users bringing their own.

What they build

The applications share a profile. They are small, they serve one department, and they automate something a spreadsheet and an email chain used to handle between them. None of them was ever important enough to earn a place on the IT backlog.

- **Dashboards** that pull live numbers out of an operational system.
- **Workflows** with a form and an approval step, such as client intake, vendor onboarding or expenses.
- **Calculators** for quoting, pricing, or configuration.
- **Portals and trackers** that let clients check status without telephoning anybody.
- **Report generators** that end a monthly copy-and-paste ritual.
- **Glue** between two systems that ought to talk to each other and do not.

Why the specification is better

The person with the problem knows the exceptions, and knows which step genuinely wastes an afternoon and which one only looks wasteful. Write all that down for somebody else and most of it evaporates; route it through an analyst who then briefs a developer and more goes still.

When the employee builds it themselves the specification never has to travel, which removes the largest single cause of failure in internal software: the wrong thing, built beautifully.

Vendor case studies claim ten semi-technical builders shipping eight production applications in **30** days with no developer involved, and a process falling from five days to twelve hours. Treat that precision sceptically, since Section 11 explains what such figures do and do not measure, but the direction of travel is not in dispute.

Why a ban does not work

A prohibition produces one of three outcomes. Staff ignore it; or they move to personal accounts and personal email, at which point you lose visibility altogether; or an executive overrules you, because the value is real and somebody senior has noticed. None of the three is what you wanted, and since the tool is a browser tab and a credit card, blocking it was never really available to you.

Yet these apps often end up in the same place — company data sitting in an application nobody in IT has ever looked at, which is both a leak waiting to happen and an open door into whatever it connects to.

THE BOTTOM LINE

The question is not whether your employees build software, because they already do. The question is whether you can see it.

FINDING 2

2. Almost Every App Has the Same Hole

Four teams, four methods, four samples — same failure.

IN PLAIN ENGLISH

Four separate research teams, using four different methods, found the same thing: almost every app built this way has a security hole. One study of **1,072** live apps found problems in **98%** of them, and another examined **5,600** apps without finding a single one with proper access controls. It is nearly always the same hole, too — the app never checks whether the person using it is allowed to see the data. Your employee cannot catch this, because they can see their app works but not that it also works for a stranger.

Independent researchers using different techniques, with different commercial motives and no coordination between them, arrived at the same conclusion. In a field **18** months old that almost never happens, which is why this is the strongest result in the corpus.

Four independent studies, one conclusion

Who and when	What they did	What they found
Symbiotic Security 2 June 2026	Crawled 65,643 addresses, then fully scanned 1,072 live apps built on Lovable, v0, Bolt, Replit and Windsurf.	98% had at least one vulnerability and 16% had a critical one, averaging 5.9 flaws per app and 6,185 in total.
Cloud Security Alliance 2 June 2026	Reviewed a scan of 5,600 production applications.	Not one had cross-site request forgery protection, security headers, or properly scoped access policies.
Veracode reported April 2026	Tested more than 100 language models generating code.	45% of samples introduced a vulnerability from the OWASP Top 10, with no improvement over time.
Wiz Research 2025–2026	Fingerprinted live applications and tested their configuration.	One organisation in five is exposed, through four failure types that repeat endlessly.

It is always the same four failures

The defect rate is less interesting than the pattern behind it. Nobody is finding exotic or clever vulnerabilities; they keep finding the same small handful.

- 1. The database says yes when it should say no.** In the Symbiotic sample, **172** applications let any stranger delete records, **172** let anyone modify them, and **39** exposed entire tables to be read. The cause in each case was a public key with no access policy behind it.

- 2. Passwords and keys left in the open.** Credentials sit in code that the browser downloads, which is how one count reached **1,275,105** leaked AI service keys in 2025, up **81%** in a year.
- 3. The lock is on the wrong side of the door.** The application checks your password inside your own browser, where anyone can remove the check.
- 4. No lock at all.** Internal dashboards and test systems published straight to the open internet.

An autonomous penetration test reported on 22 July 2026 found the same thing: **434** exploitable problems across three applications, in scans of **30** minutes each. The biggest groups were authorisation failures (**88**) and access-boundary failures (**54**), and of the **23** critical findings, **11** were hardcoded secrets and **6** were remote code execution through a debug mode somebody left switched on.

Why this failure and not some other

The mechanism behind all this matters rather more than any of the percentages above.

Your employee can verify that the application works, because they open it, do the task, and watch the task get done. What they cannot verify is that it refuses to work for somebody without permission, since access control has no visible symptom at all. A safe application and a dangerous one look identical from the builder's chair.

THE BOTTOM LINE

The flaws cluster in access control because access control is the one thing the builder physically cannot see.

FINDING 3

3. The Bill Did Not Vanish.

Writing code got cheap. Checking and fixing it is still expensive.

IN PLAIN ENGLISH

AI did not remove the cost of software so much as move it. Writing became cheap, while checking, fixing and maintaining became expensive. The largest study of this found that almost one in four of the faults AI introduces is never fixed at all and simply stays in the product. Developers describe the same thing differently: two thirds of them say the AI gives answers that are "almost right, but not quite", and that is the most expensive kind of wrong there is.

The largest study in the corpus examined **302,600** verified AI-written commits across **6,299** public repositories and five different AI assistants, running static analysis before and after every change and then following each problem forward through later versions.

It found **484,366** distinct problems, with more than **15%** of commits from every single assistant introducing at least one. But one number matters considerably more than the rest.

22.7%

of the defects AI introduces are still there in the current version. They are never fixed.

Debt Behind the AI Boom, 302,600 commits, March 2026

That is a measured liability rather than an opinion, and it means nearly one problem in four becomes permanent furniture.

Where the work went

The Stack Overflow survey of **49,009** developers shows exactly where the effort relocated to.

- **66%** say the tool produces "solutions that were almost right, but not quite".
- **45%** say debugging AI-written code takes longer than debugging code a human wrote.
- **40%** say it handles genuinely complex work badly.
- **75%** go and find a human when they do not trust an AI answer.
- **62%** want a human involved in anything touching security or ethics.

"Almost right, but not quite" is the most expensive category of wrong in software. Obviously wrong costs nothing because you throw it away, whereas almost right sails through review and then fails in front of a customer six weeks later.

Fast and broken at the same time

The Cloud Security Alliance states the trade directly: developers using AI ship code three to four times faster, and introduce vulnerabilities at ten times the rate. Both figures are true at once, and together they describe a straightforward problem, which is that output accelerated and checking did not.

A brand new way to get attacked

One consequence of all this has no real precedent. Roughly **20%** of AI-generated code refers to software packages that do not exist, so attackers now register those invented names and fill them with malware, and the next person to run the code installs it.

The industry calls this slopsquatting. No traditional review process catches it, because nothing about it looks wrong.

THE BOTTOM LINE

AI did not remove the cost of software. It moved that cost downstream, where it is harder to see and harder to budget for.

FINDING 4

4. Nobody Can Feel Their Own Productivity

They were **19%** slower. They were certain they had been **20%** faster. This is the most important number in this document.

IN PLAIN ENGLISH

This is the most important number in the document. In the one properly controlled experiment ever run, developers using AI took **19%** longer to finish their work, and then reported that it had made them **20%** faster. They were wrong about their own experience, in the same direction, by a wide margin. So when somebody tells you their team is faster with AI, that is a feeling rather than a measurement, and you should ask for the defect rate and the maintenance load instead.

METR ran the only randomised controlled trial in the field and published it in July 2025.

DISCLOSURE

A principal of JetBridge, the publisher of this document, is a researcher at METR. We had no role in the design, conduct, analysis or publication of the trial described below, and we quote it exactly as published. We flag the connection because this section carries more weight in our argument than any other, and you should be able to discount it accordingly. The METR result is independently reproducible from the published paper (arXiv 2507.09089), and nothing in Sections 2, **3**, **7** or **12** depends on it.

The method was strict. Sixteen experienced developers worked on **246** real issues in their own repositories, projects carrying more than **22,000** stars each, with every issue randomly assigned either to allow AI tools or to forbid them. Sessions were recorded, tasks averaged two hours, and the developers were paid **\$150** an hour.

39%

the gap between what was measured and what was believed

METR randomised controlled trial, July 2025

They took **19%** longer with AI. Beforehand they had expected a **24%** speed-up, and afterwards, having personally lived through the slowdown, they estimated they had been **20%** faster.

What this does and does not prove

The authors are careful about this and we will be too. They make no claim that the result holds for all developers or all software work, sixteen people is a small sample, and the tasks were complex changes to large existing codebases, which is a different activity from building something new.

Strip away those caveats and one thing survives: a person using these tools cannot tell what the tools are doing to them.

What that means for governance

Nearly every productivity claim in this market is self-reported. A vendor survey proudly reports that **72%** of enterprises measured a return on their AI investment, which is a self-report, and METR tells you what a self-report is worth.

You therefore cannot govern any of this with satisfaction surveys or enthusiastic team updates. You have to count things instead: defect rate, time to fix, applications in production, and applications with nobody's name on them.

THE BOTTOM LINE

When somebody tells you their team is faster with AI, they are reporting a feeling. Ask for the defect rate instead.

FINDING 5

5. The Sentence Your Employee Types Is the Asset

The research examines whether the app leaks IP to frontier labs who may empower your competitors with your process improvements and ideas.

IN PLAIN ENGLISH

When your employee types "build me a tool that prices a job the way we price jobs", that sentence is your company's know-how, and it is worth more than the app it produces. On most platforms that sentence goes to an AI vendor, and whether it trains their model is decided by a contract nobody in your company read. Almost none of the research covers this, because the researchers sell security scanners and a scanner cannot read a contract. Ask four separate questions: where is my data, who owns the model, who can switch it off, and do my instructions train it?

Your employee types a sentence in plain English, and that sentence describes how your company prices a job, triages a claim, or decides which client to take on. The description is the valuable part and the code is merely its output, which is why a competitor who obtained the sentence would learn far more than one who obtained the software.

On most public platforms that sentence goes straight to a model vendor, and whether it becomes training data is settled by a contract term your employee never read and your company never negotiated.

What 34 sources cover — and what they do not

Question	How well the field covers it
Vulnerabilities in the deployed app	Extensively. Several independent studies.
Leaked credentials and secrets	Extensively. Counted at scale.
Defects that survive into later versions	One large study.
Where the app's data physically lives	Thinly, and mostly in vendor marketing.
Whether your instructions train someone's model	Almost nothing.
Who owns the model weights	Absent from the security literature entirely.

"Sovereign AI" is four different questions

Vendors routinely answer one of them and let you assume the rest, so ask them separately.

1. Residency. Which country, and whose hardware, does the data sit on?

2. Ownership. Who owns the model weights, and can you keep operating if the vendor withdraws the model?

3. Control. Who can switch it off, and who can read the logs?

4. Training rights. Do your instructions and your data improve a model that your competitors then use?

A private-cloud arrangement answers the first question and part of the third, while saying nothing whatever about the second or the fourth. This is not a technicality. In August 2026 a major cloud provider and a platform vendor announced exactly such an arrangement, with the reassurance that data never leaves, and that statement is entirely accurate about residency without being a statement about training rights at all.

Why the researchers missed this

The explanation is structural rather than sinister. The security researchers who produced the best evidence in this review sell scanning products, and while a scanner can find an exposed database in seconds, no scanner has ever read a contract.

THE BOTTOM LINE

The best-equipped people in this field are pointed at the wrong half of the problem.

FINDING 6

6. Everyone Names the Disease. Nobody Writes the Prescription.

IN PLAIN ENGLISH

Nearly every report tells you the problem and almost none tells you the fix. The advice that does exist says things like "check the security settings", which only works if the person can read code, and if they could they would not need the tool. The Cloud Security Alliance has now said this officially: none of the major security frameworks covers the ordinary employee who builds an app.

What the source offers	Share of the corpus	Comment
A description of risks, with data	Most of it	The diagnosis is complete and consistent
Advice for the individual developer	A small handful	Checklists and prompt rules, written for people who read code
A control architecture for a platform	Almost none	This is the gap
Anything written for a non-programmer	None found	See below

A standards body has now said this out loud

The Cloud Security Alliance examined the major AI security frameworks and reported that none contains dedicated guidance for the citizen developer, naming the NIST AI Risk Management Framework, the OWASP Top 10 for LLM applications, CSA MAESTRO, and the CSA AI Controls Matrix. That is a formal statement, from a neutral body, that the rulebook for this does not yet exist.

Why advice to the builder can never work

Most published guidance tells the builder to do something: review the generated code, check the access policy, strip out the secrets, add a test. It carries a fatal flaw, which is that it requires precisely the skill the builder does not have. If your operations manager could audit an access policy she would be a developer, and she would not need the tool in the first place.

There is a second and more arithmetic reason. Nobody reviews **400** applications built by non-engineers, and review will never scale to that number.

THE BOTTOM LINE

The control has to live in the platform rather than in an instruction to the employee, so that the safe outcome becomes the default outcome with no action required from the person building.

PART TWO

What to Do About It

7. The Twelve Things AI Will Not Do

IN PLAIN ENGLISH

This is the practical heart of the document. The AI builds the screens and the buttons, and twelve things remain, which are the twelve that decide whether the app is safe, legal and still working next year. Three of them are not really programming at all, but questions like "which of our three customer lists is the real one?" No AI can answer those. Only your people can, and usually only one or two of them, which is why this stage needs a human expert and why a better AI model will not remove the need.

The AI builds the visible part, meaning the screens, the forms, the basic logic and the report at the end. Everything below is what it leaves behind, and each item names the finding that proves it is necessary.

The last mile, itemised

#	What an expert has to do	Why — the evidence
1	Set access control at the data layer. Decide who may read, create, change and delete each record, enforce it in the database rather than on the screen, then test it as somebody with no permission at all.	Finding 2. The biggest defect class in every study.
2	Connect it to company identity, so that when somebody leaves they lose access to every application at the same moment.	Finding 2. Orphaned apps keep live credentials alive.
3	Get the secrets out of the code. No key or password in anything the browser downloads. Move them to a vault and rotate them.	Finding 2. 1,275,105 keys leaked in 2025.
4	Design the data flow. Decide which system holds the true record, define which way data moves, and stop two systems writing the same field.	Finding 8. The builder has the business knowledge, not the data model.
5	Build the connectors into the ERP, the practice system and the accounting package. Many have no modern interface, and some need a scraping robot pointed at a supplier portal.	Finding 1. This is precisely where the app stalls at 80% .
6	Fix the data. Decide which of the three customer tables is authoritative, repair the field storing numbers as text, and write down what status code 7 means.	Finding 8. Organisational knowledge, not a coding task.
7	Add the audit log and retention. Who did what and when, kept as long as the regulator requires, and producible during an examination.	Finding 6. No framework specifies this for your builders.
8	Make unattended automation safe. Decide what happens when step four fails at 2am with nobody watching, and make it safe to run twice.	Finding 3. Almost-right is the normal failure mode.

#	What an expert has to do	Why — the evidence
9	Check the supply chain. Verify that every package actually exists and is genuine, since roughly 20% of AI package references are invented.	Finding 3. Slopsquatting.
10	Add change control. Give it a version, a test, and a way back to the last version that worked.	Finding 3. 22.7% of defects are permanent.
11	Record an owner. Name the person, classify the data, state the purpose, and revisit it when that person leaves.	Finding 1. You cannot maintain what you cannot list.
12	Nail down the sovereignty terms. Get all four answers from Section 5 in writing, for this application and this model.	Finding 5. Absent from the literature entirely.

Invented packages are the smaller half of item 9. In November 2025 the Shai-Hulud **2.0** campaign compromised roughly **800** legitimate npm packages, ran during installation, and harvested **775** GitHub tokens along with **373** AWS, **300** GCP and **115** Azure credentials — the dependency that should worry you is not the one that does not exist, but the trusted one that does.

Let's look at items 4, 5 and 6 again

They are knowledge problems wearing the costume of programming problems, and item 6 has no technical solution anywhere. Somebody in your building knows which customer table is the real one, why the discount field holds text, and what status code 7 meant back in 2019, and that knowledge sits in one or two heads without ever having been written down.

Which leads to what we think is the single most useful observation in this document.

Employee-built software does not create data debt. It exposes data debt that spreadsheets have been hiding for a decade.

The moment an application connects two systems, those two systems have to agree with each other, and they almost never do.

THE BOTTOM LINE

This is why the last mile needs a person (often a Forward Deployed Engineer) rather than a better model. AI can generate the connector, but it cannot decide which of your three customer records is the true one. Only your organisation knows that.

8. The Rise of the Forward Deployed Engineer

The **10x Engineer** doesn't code 10x faster, they convert disparate employee meetings into architected data flows.

IN PLAIN ENGLISH

Somebody still has to finish these apps, and the whole industry has landed on the same answer: an engineer who comes and sits with your team rather than one you hire. Postings for this role grew tenfold in a year, and Palantir, OpenAI, Anthropic and Google are all competing for the same people. They do four things at once — write down the knowledge that only lives in your staff's heads, build the connections into your older systems, make the app secure, and get IT and compliance comfortable enough to approve it. You pay by the hour rather than adding a permanent salary, which fits, because this work comes in bursts. One warning: insist that whatever they learn about your business gets written down and left behind. Otherwise you are renting an understanding of your own company.

Section 7 describes twelve things that have to happen after the AI stops. Somebody has to do them, and over the past eighteen months a particular kind of somebody has become the standard answer across the industry.

The forward deployed engineer, or FDE, is an engineer who works inside the customer's organisation rather than at a vendor's office. Palantir invented the pattern around 2005 and for years employed more embedded engineers than conventional product engineers. The model stayed a Palantir peculiarity for the best part of two decades, and then AI arrived and everybody copied it at once.

1,000%

growth in forward deployed engineer job postings, year on year through early 2026

Analysis of 1,000 FDE job postings, 2026

Palantir still leads on volume, followed by OpenAI, Anthropic, Google, Databricks and Scale AI, with the steepest growth rates appearing at vertical AI companies. Mid-to-senior roles cluster between **\$300,000** and **\$550,000** in total compensation, and staff and principal roles at the frontier labs run past **\$1.2** million.

Why the model came back

The reason is not a talent shortage. It is a failure rate. Roughly **95%** of enterprise AI pilots produce no measurable business impact, and the cause is almost never the quality of the model: it is everything around the model, in the customer's own systems, with the customer's own data.

One forward deployed engineer described the work in a sentence that could serve as the caption for Section 7 of this document.

The model is usually the cleanest part. The hard part is finding the workflow nobody documented, and the data source people actually trust.

That is items 4, 5 and 6 of our specification, described by somebody who does the work rather than somebody who studies it. The undocumented workflow and the trusted data source are not technical objects at all. They are organisational knowledge, and they cannot be extracted by a model, a consultant working from a written brief, or a platform, however well governed.

The four jobs inside the job

What makes the role effective is that it combines four functions that companies normally split across four different people, none of whom talks to the others.

- 1. Organising the team and its knowledge.** The FDE sits with the people who have the problem and writes down what has never been written down: which system holds the true record, why a field contains text instead of a number, what an undocumented status code means, and which of three customer tables anybody actually trusts. Item 6 of Section 7 has no technical solution, and this is the only reliable way it gets solved.
- 2. Coding the gaps.** When the AI stops at **80%**, somebody has to build the connector into the ERP that has no modern interface, or the scraping robot pointed at a supplier portal, or the reconciliation logic between two systems that disagree. That is engineering work, and it needs an engineer.
- 3. Hardening what the employee built.** Access control at the data layer, identity, secrets, audit logging, error handling for automation that runs unattended, and a way back to the last version that worked. Section 2 of this document explains why the person who built the application cannot do this, and why the failure is invisible to them.
- 4. Working with IT and compliance.** This is the function nobody else performs and the one most often left out of the job description. The FDE translates in both directions, turning a business user's intention into something a security team can approve, and turning a compliance requirement into something the platform enforces by default. Without it, governance stays a document and the applications keep shipping around it.

Why hourly beats headcount

The economics matter as much as the skills, and they are the reason the model suits mid-market companies specifically.

The last mile is episodic rather than continuous. It arrives in bursts, when an application is ready to graduate from a demonstration into production, and then it goes quiet until the next one. Hiring a senior engineer

converts that spiky demand into a permanent salary, a manager, a retention problem and a backfill risk, and at the compensation levels above it is a competitive hiring fight that most mid-market companies will lose.

Buying the same expertise by the hour matches the cost to the shape of the demand. It scales up when three applications are ready at once, scales down to nothing when none are, and ends when the work ends rather than becoming a line item that outlives the project.

Four ways to get the last mile done

Approach	What it costs you	Where it fails
Hire senior engineers	Permanent salary, management overhead, recruitment cycle of three to six months	Converts episodic demand into fixed cost, and you are bidding against frontier labs for the same people
Absorb it into internal IT	Roadmap capacity, silently	Internal teams are measured on feature delivery, so the last mile loses every prioritisation meeting it enters
Traditional consultancy	Fixed-scope project fee	Scope is agreed before anybody knows what the data actually looks like, and ownership ends at handover, which is exactly when an AI system needs the most support
Forward deployed engineer	Hourly or day rate, variable	Dependency risk if the knowledge leaves with the engineer. See below.

The biggest risk in the FDE model

The FDE model has a real failure mode, which is dependency: the engineer accumulates the organisational knowledge from function one, and if that knowledge leaves when they do, you have rented understanding of your own business rather than acquiring it.

Make the written artefacts a deliverable in their own right, including the data map, the decisions about which system is authoritative, and the documented meaning of the fields nobody could previously explain. If an engagement ends and the only record of what was learned is in somebody else's head, the engagement failed regardless of how well the software works.

THE BOTTOM LINE

The last mile is real work that arrives in bursts and demands scarce skills. Buying it by the hour matches the cost to the shape of the demand, which for most companies short of engineering and IT bandwidth is the fastest route from a promising demonstration to something the business can actually depend on.

9. What to Do on Monday

IN PLAIN ENGLISH

Sort the apps into three groups by how sensitive the data is, and apply the heavy controls only to the top group. Then do the one thing that decides whether any of this works, which is to offer an amnesty: ask your staff to tell you what they have already built, and do not punish anybody for it. If people fear punishment they will hide the apps, and hidden apps are the whole problem. After that, make the safe path the easy path, because if your approved tool is slower to start than the public one your staff will use the public one.

First, stop treating every app the same

A dashboard showing public data does not need what a client portal needs, so sort them into three tiers and spend your effort on the top one.

Three tiers, three levels of effort

Tier	What it looks like	What it needs
Tier 1	No company data, or public data only. One user.	An entry in the inventory and a named owner, and nothing more.
Tier 2	Internal company data. Several users. No personal or regulated data.	Inventory, single sign-on, secrets management, audit log, named owner.
Tier 3	Personal, client, financial or regulated data. External users.	Everything in Section 7, completed by an expert before it goes live, with no exceptions.

Then next 11 moves

- 1. Find out what already exists.** You cannot govern an unknown quantity, so check expense reports for platform subscriptions and identity logs for connections to build platforms.
- 2. Declare an amnesty.** Ask people to register what they have already built, with no penalty whatsoever. Punishment produces concealment, and concealment is the entire problem you are trying to solve, which is why this single step decides whether the programme succeeds.
- 3. Make the approved path the easy path.** People take the route of least resistance, so if your sanctioned platform is slower to start than a public one they will use the public one and you will have achieved nothing.
- 4. Put identity in the platform.** Every application inherits company sign-on, and no application manages its own passwords.

5. **Put permissions in the connector.** The connection to a system of record carries the user's own permissions, so nobody can build an application that reads data they themselves cannot read.
6. **Ban secrets from application code.** The platform holds the keys and the builder never sees one.
7. **Log everything by default.** Audit records should appear without anybody configuring anything.
8. **Gate the launch, not the build.** Let people build freely, and apply the control the moment an application touches real data or gains a second user.
9. **Name an owner for every application.** An application with no owner gets switched off after notice, which is how you avoid accumulating a graveyard of software nobody dares delete. You may want to have only one user in the organization pre-configure connectors.
10. **Count things rather than surveying them.** Track applications in production, defect rate, time to fix, and applications with no owner. Section 4 explains why a satisfaction survey is worthless here.

THE BOTTOM LINE

If you do only one thing, offer the amnesty. Everything else depends on knowing what is already out there.

PART THREE

How We Know, and What We Cannot Know

10. A Weekend Hobby Went Into Production

IN PLAIN ENGLISH

The person who invented this term said it was for weekend hobby projects you throw away afterwards. Companies now use the identical method to build tools holding customer records, so nothing about the method has changed and only the stakes have. Professional programmers understand this perfectly well: **78%** of them use AI tools every month, and **76%** of them refuse to work this way.

Andrej Karpathy coined the term in February 2025 with a precise definition: you tell an AI what you want, accept what it gives back, and never read the code. You can, he wrote, "forget that the code even exists". He was equally precise about what it was for, and what it was for was throwaway weekend projects.

The correction that drew the line

Simon Willison drew the useful line on 19 March 2025, pointing out that using AI to help write software is not the same activity as vibe coding, and that the boundary between them is the code review. A developer who reads what the model produced is a professional using a tool, and one who does not is vibe coding. Willison's own rule is that he will not commit code he could not explain to somebody else.

What working programmers actually think

The 2025 Stack Overflow survey collected **49,009** responses from **160** countries, and the split it found is a sharp one.

78.5%**of developers use AI tools every month***Stack Overflow Developer Survey, 2025 (n=49,009)*

76%

of the same developers reject vibe coding as a practice

Stack Overflow Developer Survey, 2025

Only **3.1%** report high trust in what the tools hand them, and among experienced developers that figure falls to **2.5%**. They use these tools constantly and trust them very little, which is roughly what happens when somebody actually reads the output.

Why this is the whole problem

The word travelled from weekend projects into production systems without the practice changing to match. A method designed explicitly for work that does not matter is now producing applications that hold client data, and Sections 2 and 3 of this document are a consequence of that single mismatch.

THE BOTTOM LINE

A practice designed for disposable work is now running production systems, and everything that follows is a consequence of that.

FINDING 7

11. Both Sides Are Right. But They're Timing Different Races.

IN PLAIN ENGLISH

Vendors are not lying when they say a team built eight apps in a month, because they are measuring the moment the app first works. The researchers are measuring what that same app looks like a few months later. Both are correct, and nobody has yet measured the whole journey, which means nobody can honestly tell you what this really costs.

The two bodies of literature look as though they are calling each other liars, when in fact they are standing at different points on the track holding different stopwatches.

The same app, measured twice

	What vendors publish	What researchers publish
What is measured	Time to a first working version	Condition of the app after it ships
Typical claim	8 apps in 30 days. Five days cut to twelve hours. Three to five times faster.	98% contain a vulnerability. 22.7% of defects never fixed.
When measured	The moment the build finishes	Weeks or months later
Unit	A project or a team	An app, a commit, a defect
Who is ignored	Whoever operates it afterwards	Whoever needed it in the first place

Both sets of numbers can describe the same application. A team can absolutely ship eight applications in thirty days, and those eight applications can absolutely be full of holes, since nothing in the evidence prevents it and it is in fact the expected outcome.

The study nobody has run

Not one source follows a set of employee-built applications from first version through security review, correction, deployment and twelve months of ownership, and then reports the total. Until somebody does, no enterprise can calculate a real return on any of this, because you can only price the first stage, which is conveniently the cheap one.

Three questions that make a speed claim useful

- 1. What state was the application in when the clock stopped?** Demonstrated to a room, or in production with real users and real data?
- 2. Who finished it, and how long did that take?** This is the number that never appears in the case study.

3. What condition are those applications in now, and who maintains them? Ask specifically about the ones built eighteen months ago.

The vendors agree with us

A platform vendor whose entire business is selling a tool that lets non-engineers build software surveyed **50** executives and vice presidents at companies of **500** to **10,000+** employees, and its own research found the following.

- **70%** disagree that this practice will replace core engineering.
- **72%** plan to employ the same number of engineers next year, or more.
- **Top three worries:** security (**25.6%**), compliance (**19.2%**), and the model simply being wrong (**17.9%**).

A company with every commercial reason to claim the opposite is reporting that expert engineers are still required. The sample is small enough that the decimal places should be treated loosely, but the direction is what counts, and it matches every independent source in this review.

THE BOTTOM LINE

When a vendor shows you a speed number, ask one question: who finished it, and how long did that take?

FINDING 8

12. The Studies Are Watching the Wrong People

Almost all of this research examines professional programmers.

Yet almost none of your builders are professional programmers.

IN PLAIN ENGLISH

Almost all the research studies professional programmers, but the people building apps in your company are usually in operations, finance or client service, and nobody has properly studied them yet. That cuts both ways, since the benefit to them may be much larger than the studies show and so may the risk. What we can say safely is this: if trained programmers who read every line still produce these faults, people who never read the code will not do better.

Read the method sections back to back and the pattern jumps out immediately.

Who each study actually watched

Study	What it counted	Who it watched
METR randomised trial	A developer, and a task	16 experienced open-source developers
302,600 -commit study	A code commit	Contributors to public repositories
Stack Overflow survey	A developer	49,009 professional developers
Veracode model tests	A generated code sample	No people at all, models only
Symbiotic scan (1,072 apps)	A deployed application	Unknown. The builder is invisible.
CSA scan (5,600 apps)	A deployed application	Unknown. The builder is invisible.

Every study that watches people is watching professionals, and every study that examines applications has no idea who built them. Your own population sits outside both: an employee who is not a developer, building on company data, inside a company with compliance obligations. Nobody has studied them.

This cuts both ways, and honest reading requires saying so

- **The upside may be understated.** METR watched expert developers doing hard work in large codebases, whereas an operations manager building her first small application has no comparable skill to lose. The gain for her could be far larger than anything in these studies, and nobody has measured it.
- **The risk may also be understated.** Every scan looked at applications visible on the open internet, so anything sitting behind a company login appears in no sample anywhere. Those are the ones holding the more sensitive data.

What we can say safely

One inference is airtight: if trained programmers who read every line still produce these defect rates, people who never read the code will not do better. That is the weak form of the argument, and it is enough, because the direction is certain even though the magnitude remains unmeasured.

THE BOTTOM LINE

Treat the direction of this evidence as reliable and the exact percentages as approximate.

13. What This Evidence Cannot Tell You

IN PLAIN ENGLISH

We should be clear about what this evidence cannot do. Nobody has studied ordinary employees building apps, only professional programmers, and nobody has looked inside company firewalls, only at apps on the open internet. The field is about **18** months old, and much of the research is published by companies that sell security tools. None of this makes the findings wrong, but it does mean you should treat the direction as reliable and the exact percentages as approximate.

1. **The population is wrong.** No rigorous study watches the employee who is not a developer, which is the most important hole in the field and one somebody should fill.
2. **Only visible applications have been examined.** Every scan looked at the open internet, so the applications behind your firewall, which hold more sensitive data, have never been studied by anybody.
3. **The field is young and largely unreviewed.** Much of the strongest work here is a preprint or a vendor report, and the subject is about **18** months old.
4. **Much of the evidence comes from people selling a cure.** That does not make it wrong, though independent replication would make it stronger. Appendix A records every publisher's commercial interest so you can weigh it yourself.

The four studies this industry needs

- **A cohort study** following employee-built applications from first version through twelve months of ownership, reporting the total cost.
- **A scan behind the firewall**, with the consent of the companies concerned.
- **A controlled trial with non-programmers**, to finally measure the effect on the population that actually uses these tools.
- **A control framework for the citizen builder** from a standards body, since the Cloud Security Alliance has confirmed that none exists.

THE BOTTOM LINE

Trust the direction of this evidence and treat the decimal places as approximate.

14. Who Wrote This, and What We Sell

JetBridge published this document, and JetBridge operates AirCode, a platform for employee-built applications. We have a commercial interest in these findings, since Finding 6 and Section 7 describe a need our product exists to meet, and those two sections are worth rereading with that in mind.

There is one further connection. A principal of JetBridge is a researcher at METR, the organisation behind the randomised controlled trial in Section 4. We had no part in that study, the disclosure appears in full alongside the finding itself, and our argument does not rest on it.

What AirCode is

- **A build platform** with the Section 9 controls, so identity, permissions, secrets, audit logging and inventory all happen without the employee doing anything.
- **A sovereignty position** that answers all four questions in Section 5. We do not train models on customer prompts or usage, and enterprise customers can run the harness and an open model on their own hardware.
- **Forward deployed engineers** who complete the twelve items in Section 7.

Section 7 shows that the last mile requires an expert human, and most platforms in this category hand you a governed platform and then leave that work to your engineering team. Findings 8 and 6 explain why this fails in a mid-market company, which is that you do not have spare senior engineers and neither does anybody else. The ones you have are committed to the revenue roadmap, and the last mile quietly taxes it.

We supply the engineers alongside the platform, which turns the last mile into a variable cost rather than permanent headcount.

Potential Trade Offs Between Sovereignty and Efficiency

AirCode does not train on customer prompts or usage, whereas platforms that do get steadily more efficient, because they learn from their customers and spend fewer tokens on the same task. We never get that benefit, so our platform uses more tokens than a platform that learns from you, and we accept it.

Slightly increased inefficiency may be the receipt for the privacy.

When you should not buy this

- **You have a large engineering team with spare capacity.** Then you do not need our engineers, so buy a platform and use your own people.
- **You have no compliance obligation and no proprietary process worth protecting.** A public platform will serve you perfectly well and cost you less.
- **You want a demonstration rather than a production system.** None of this applies to you, so go and build something on a weekend, exactly as Karpathy intended.

15. How We Read the Field

34 sources digested: What happens when a non-programmer ships software?

IN PLAIN ENGLISH

We read **34** documents about employees building apps with AI, and for each one we noted how big the study was, when it was done, and what the publisher sells. Then we compared them. We also list our own weaknesses and any potential conflict of interest.

We looked at everything published between February 2025 and August 2026 that either stated a method or reported original data:

The corpus at a glance

Category	Count	What kind of evidence	Example
Academic	8	Original research, peer review or preprint	A study of 302,600 code commits
Standards body	5	Synthesis across other studies	Cloud Security Alliance research notes
Security research	7	Scans of real, live applications	A scan of 1,072 deployed apps
Industry survey	4	Self-reported, large samples	49,009 developers surveyed
Platform vendor	5	Marketing, case studies, small surveys	A survey of 50 executives
Public argument	5	Opinion, incidents, forum debate	A production database destroyed live

What we recorded about every source

We noted seven things about each source:

- **Date.** This field moves in months rather than years, so anything much older than a year needs checking before you lean on it.
- **Method and sample size.** Sixteen developers and forty-nine thousand developers support very different claims.
- **Unit of analysis.** A person, a commit, or a deployed application. These are not interchangeable, and mixing them is how bad statistics get made.
- **Population.** Professional developers, or ordinary employees? Section 12 explains why this turns out to matter more than anything else on the list.
- **Access.** Free, gated, or paywalled.
- **Commercial interest.** What does the publisher sell? Profiting from a finding does not make the finding wrong, but you should be able to see the incentive.

- **Prescription.** Does it only name a risk, or does it also tell you what to do about it? Very few manage the second, as Section 6 sets out.

Where this review is weak

We state our own limits, mostly because so few of the documents we reviewed state theirs.

1. **This is a narrative review, not a systematic one.** We worked without a pre-registered protocol, and a different reader might reasonably weight these sources differently.
2. **The field is 18 months old.** Peer-reviewed work is still thin, so we included preprints, and a preprint has not been checked by anybody.
3. **We do not combine the numbers.** A percentage of applications and a percentage of developers cannot be averaged together, so we report each separately rather than manufacturing one large statistic.
4. **We sell something adjacent.** Section 14 says what. Sections 15 to 13 name no product.
5. **We are connected to one of the sources.** A principal of JetBridge (creators of AirCode) is a researcher at METR, whose randomised trial we rely on in Section 4. The full disclosure sits in that section, beside the finding it affects.

Appendix A — The Source Corpus

Every source, with its date, its sample size, and what its publisher sells.

This appendix lists the **34** sources of this review. "Interest" records what the publisher sells. A finding is not false because the publisher sells a product. But you must know.

A.1 Academic sources

Source	Date	Method and sample	Principal result	Interest
A Survey of Vibe Coding with Large Language Models (arXiv 2510.12399)	Oct 2025	Systematic review of more than 1,000 papers	Defines 5 development modes. Formalizes the practice as a constrained Markov decision process.	None
Debt Behind the AI Boom (arXiv 2603.28592). Liu, Widyasari, Zhao, Irsan, Chen, Lo	Mar 2026	Static analysis of 302,600 AI-authored commits in 6,299 repositories, 5 assistants	484,366 problems. Code smells 89.3% . More than 15% of commits introduce a problem. 22.7% of problems survive to the current version.	None
Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity (METR, arXiv 2507.09089)	Jul 2025	Randomized controlled trial. 16 developers, 246 real issues.	19% slower with AI. Expected 24% faster. Estimated 20% faster after the trial.	See disclosure, Section 4
Vibe coding: programming through conversation with artificial intelligence (arXiv 2506.23253)	Jun 2025	Conceptual and qualitative	Characterizes conversational programming as a distinct practice.	None
From Prompting to Verification: How Experience Shapes Vibe Coding Practices (arXiv 2605.24521)	May 2026	Qualitative study of practice	Experience changes the activity from prompting to verification.	None
Context Before Code: An Experience Report on Vibe Coding in Practice (arXiv 2603.11073)	Mar 2026	Experience report	Context supplied to the model determines the result more than model capability.	None
Computer Science Achievement and Writing Skills Predict Vibe Coding Proficiency (arXiv 2603.14133)	Mar 2026	Correlational study	Prior technical achievement predicts success with the practice.	None
Vibe Coding: Toward an AI-Native Paradigm for Semantic and Intent-Driven Programming (arXiv 2510.17842)	Oct 2025	Conceptual	Proposes intent-driven programming as a paradigm.	None

A.2 Standards bodies and institutions

Source	Date	Method and sample	Principal result	Interest
Cloud Security Alliance — The Vibe Coding Governance Gap	2 Jun 2026	Synthesis. Includes a scan of 5,600 production applications.	0 of 5,600 applications had CSRF protection, security headers or correctly scoped access policies. No major framework covers the citizen developer. Adoption took about 16 months.	Non-profit
Cloud Security Alliance — Vibe Coding's Security Debt: The AI-Generated CVE Surge	4 Apr 2026, updated 20 May 2026	Synthesis of Veracode, Apiiro, Georgia Tech, Escape.tech data	3 to 4 times faster delivery; 10 times the vulnerability rate. 35 attributed CVEs in March 2026 alone. About 20% of package references are invented.	Non-profit
Cloud Security Alliance — Vibe Coding Security Crisis: Credential Sprawl and SDLC Debt	31 Mar 2026	Synthesis	Credential exposure and process debt in AI-assisted development.	Non-profit
NIST AI Risk Management Framework; OWASP Top 10 for LLM Applications; CSA MAESTRO; CSA AI Controls Matrix	2023 to 2026	Control frameworks	None contains dedicated guidance for the citizen developer, as reported by CSA, June 2026.	Non-profit
Gartner and Forrester	—	—	No named report on this category was found. Coverage remains inside low-code and citizen development.	Analyst

A.3 Security research

Source	Date	Method and sample	Principal result	Interest
Symbiotic Security — We scanned 1,072 vibe-coded apps	2 Jun 2026	Crawled 65,643 addresses; fully scanned 1,072 deployed applications	98% had at least one vulnerability. 16% critical. Mean 5.9 for each site. 6,185 total. 172 sites permitted unauthenticated deletion.	Sells code security
Wiz Research — Common security risks in vibe-coded apps	2025 to 2026	Fingerprinting and configuration testing of deployed applications	1 organization in 5 is exposed. Four repeating failure classes. Wiz then worked with Lovable on remediation.	Sells cloud security
Xint.io / Theori autonomous penetration test, reported by SecurityWeek	22 Jul 2026	30 -minute autonomous scans of 3 applications	434 exploitable problems. 23 critical, including 11 hardcoded secrets and 6 debug-mode remote code executions.	Sells penetration testing
Veracode — model-level testing	Reported Apr 2026	Tested more than 100 language models	45% of generated samples introduced an OWASP Top 10 vulnerability. No improvement over time.	Sells code security
GitGuardian — credential exposure	2025 to 2026	Counting of leaked secrets	1,275,105 leaked AI service key instances in 2025, an increase of 81% year on year.	Sells secrets detection
Escape.tech and Georgia Tech CVE attribution	2025 to 2026	Vulnerability scanning; CVE attribution	More than 2,000 critical vulnerabilities in 1,400 production applications. 35 attributed CVEs in March 2026.	Mixed
IBM Cost of a Data Breach	2025	Annual breach cost study	Breaches that involve AI tools average 6.5 million US dollars, about 22% above the traditional figure. Shadow AI breaches average 4.63 million US dollars, about 670,000 US dollars more than other incidents.	Sells security services

A.4 Industry surveys

Source	Date	Method and sample	Principal result	Interest
Stack Overflow Developer Survey	May to Jun 2025	49,009 responses, 160 countries	78.5% use AI tools monthly. 3.1% highly trust the results. 66% report "almost right, but not quite". 45% say debugging takes longer. 76% reject vibe coding.	Community
Microsoft WorkLab	2025	Workplace survey	75% of employees use AI tools that the employer did not approve. 78% brought their own tool.	Sells AI tools
Cisco AI Readiness Index	2024	Enterprise survey	48% of employees have entered non-public company information into AI tools.	Sells networking
Deloitte AI Governance Survey; IBM Institute for Business Value	2025	Enterprise surveys	23% of organizations have a formal AI governance framework. 60% have no formal AI usage policy.	Consulting

A.5 Platform vendor publications

Source	Date	Method and sample	Principal result	Interest
Superblocks — The State of Enterprise Vibe Coding	18 Nov 2025	Survey of 50 C-suite and VP respondents, companies of 500 to 10,000+ employees	70% disagree that this replaces core engineering. 72% will employ the same number of engineers or more. 50% report non-engineering teams building applications. Concerns: security 25.6%, compliance 19.2%, model error 17.9%.	Sells the platform
Superblocks 2.0 announcement	2026	Product announcement and case studies	Claims vibe-coded apps are the leading enterprise attack vector; no citation given. Case study: 8 production applications in 30 days by 10 semi-technical builders. Another: 3 to 5 days reduced to 12 hours.	Sells the platform
AWS and Superblocks joint agreement, reported by TechCrunch	3 Aug 2026	Commercial announcement	Deployment inside customer private cloud with Aurora and Bedrock. Statement that data never leaves. Addresses residency; silent on training rights.	Commercial
Supabase — The Vibe Coding Master Checklist	16 Aug 2025	Practitioner guide, about 12 minutes	Security, data, performance and deployment checklist. Developer audience. No external research cited.	Sells the database
Wiz and Lovable joint remediation	2025 to 2026	Vendor collaboration	Updated system prompts and security guidance after the research findings.	Commercial

A.6 Public debate and incidents

Source	Date	Type	Principal content	Interest
Andrej Karpathy — origin of the term	Feb 2025	Social media post	Defines the practice as accepting AI output without reading the code. Recommends it for "throwaway weekend projects".	None
Simon Willison — Not all AI-assisted programming is vibe coding	19 Mar 2025	Practitioner essay	Sets the boundary at code review. States the professional rule: do not commit code that you cannot explain.	None

Source	Date	Type	Principal content	Interest
Replit agent production data loss, reported by Fortune and the AI Incident Database (incident 1152)	Jul 2025	Incident report	An AI agent executed destructive commands during a code freeze and destroyed production data. The vendor described it as a catastrophic failure.	None
Brad Menezes — Owning the Last Mile (Forbes Technology Council)	22 Oct 2025	Paid-membership column	Defines the last mile as the transition from prototype to production. Separates probabilistic from deterministic functions. Cites no data.	Sells the platform
Cybernews and 404 Media reporting of practitioner forums	Jun 2025; May 2026	Journalism on forum sentiment	Developer accounts of maintenance load and skill loss. Anecdotal; no sample.	None

A.7 Additional sources for Section 8

These concern the labour market for engineers rather than the evidence on vibe coding, so they sit outside the **34**-source corpus and are listed separately.

Source	Date	Method and sample	Principal result	Interest
Analysis of 1,000 forward deployed engineer job postings	2026	Job posting analysis, n= 1,000	FDE postings grew more than 1,000% year on year through early 2026, and about 800% during nine months of 2025. Mid-to-senior total compensation \$300–550K ; staff and principal roles at frontier labs \$600K–1.2M+ . Customer-facing discovery now appears in over 70% of postings.	Sells FDE recruiting
The New Stack — Why OpenAI and Anthropic are hiring forward deployed engineer teams	28 May 2026	Industry reporting and practitioner interviews	About 95% of enterprise AI pilots produce no measurable business impact, attributed to implementation rather than model quality. Model originated at Palantir around 2005. Notes that internal teams optimise for feature delivery and consultants work to fixed scope, so ownership fades when AI systems most need support.	Trade press
Reporting on FDE adoption across AI labs and vertical AI companies	2026	Industry reporting	Palantir leads on hiring volume, followed by OpenAI, Anthropic, Google, Databricks and Scale AI, with steepest growth among vertical AI companies.	Mixed

Appendix B — Glossary

Plain definitions of every technical term in this document.

Term	Definition as used in this document
Vibe coding	The practice of building software by instructing an AI model in plain language, without reading the code that the model produces.
Citizen developer	An employee who builds software applications but whose profession is not software development.
Last mile	The work that remains after the AI tool produces a version that operates. Section 7 specifies it.
Access control	The rules that decide which person can read, create, modify or delete each item of data.
Row level security	Access control applied to individual records in a database, not to the database as a whole.
Authorization	The check that decides whether an identified person is permitted to do a specific operation.
Authentication	The check that confirms the identity of a person.
System of record	The system that holds the authoritative version of a set of data.
Connector	Software that moves data between an application and a system of record.
Shadow AI	The use of AI tools by employees without the approval of the employer.
Prompt	The instruction in plain language that a person gives to an AI model.
Slopsquatting	An attack in which a person registers a software package name that AI models invent, and supplies malicious code under that name.
Code smell	A property of code that indicates a probable maintenance problem. It is not a defect by itself.
Forward deployed engineer	An engineer of the supplier who works inside the customer organization on the customer systems.
Sovereign AI	A set of four separate properties: data residency, model ownership, operational control, and training rights. Section 5.3 refers.
CVE	Common Vulnerabilities and Exposures. A public identifier for a known security defect.
CSRF	Cross-site request forgery. An attack that makes a browser send an unwanted command to an application.

Appendix C — Method Notes

How sources were chosen, how numbers were handled, and how the language works.

C.1 Selection

We searched for published material about vibe coding, AI-generated code quality, AI-assisted developer productivity, and citizen development governance, from February 2025 to August 2026. We selected sources that state a method or report primary data. We excluded material that repeats another source without addition, and search-optimized aggregation pages that give no method.

C.2 Treatment of numbers

We report each figure with its source and its sample size. We do not combine figures from different units of analysis. A percentage of applications and a percentage of developers are not comparable, and we do not present them as comparable.

C.3 Treatment of commercial interest

We record the commercial interest of every publisher, including our own. We do not exclude a source because it has a commercial interest. Several of the strongest empirical studies in this corpus come from companies that sell security products. We report the interest and let the reader judge.

C.4 Language

We wrote this document in deliberately plain language, keeping the voice active and holding each technical term to a single meaning throughout. Appendix B defines those terms. The intention is that a reader with no technical background and a reader who writes software can both take the same meaning from the same sentence.

C.5 Version

Version 1.0, August 2026. The evidence in this field changes quickly. Check the date of each source in Appendix A before you rely on it.

How This Whitepaper Was Drafted

JetBridge, makers of the enterprise vibe coding platform AirCode, conducted this meta study in August 2026. A JetBridge principal, Mischa Spiegelmock, is a researcher at METR, a non-profit AI research lab whose research is cited in this meta study.

The contents were written by humans with the editorial assistance of Claude (Opus 5), and additional editing from JetBridge's Chief Technology Officer, Adam Szeptycki.

For technical questions or inquiries contact Adam@JetBridge.com or visit AirCode.build.