
SECURE DEVELOPMENT

A sustainable case for hand-typed **code**

Why hand-typed code still earns its place in modern, AI-assisted engineering.

typed

AI writes a lot of our code now, and that's mostly a good thing. But the research keeps returning the same finding: code nobody typed is code nobody fully understands. Here is a short case for hand-typing the parts of a system that guard the door.

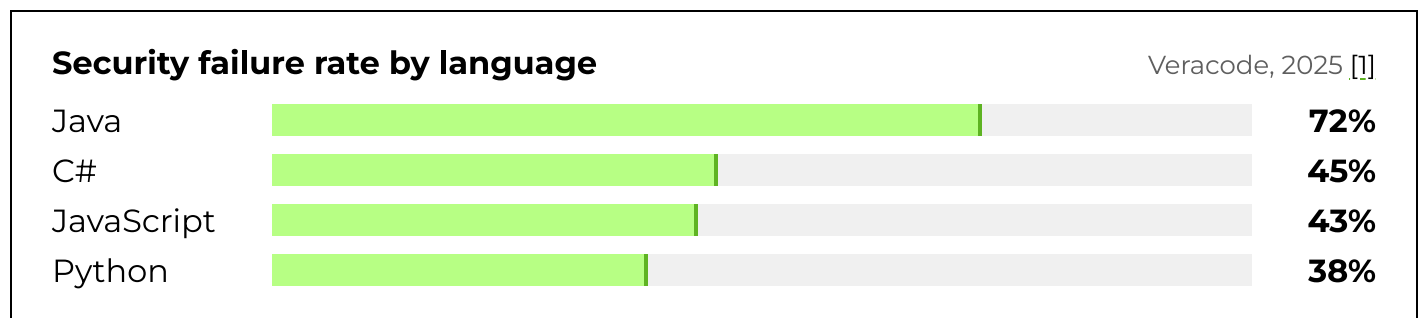
45% of AI-generated code samples failed security tests across 100+ models. [1]	1 in 5 package suggestions pointed to libraries that don't exist. [4]
More sure AI-assisted developers wrote less secure code and trusted it more. [3]	8× more duplicated code blocks in 2024, as copy-paste overtook refactoring. [7]

THE PREMISE

This isn't an argument against AI. We use it every day for scaffolding, tests and the chores that used to eat an afternoon. Our point is narrower. Typing code by hand forces you to make every decision in it. That friction used to be unavoidable, so nobody counted it as a security practice. Now that it's optional, we think it's worth keeping on purpose for the code where one wrong decision becomes a breach. The data is cited; the conclusions are ours.

The code runs. That's the problem

In July 2025, Veracode gave more than 100 language models 80 coding tasks and ran the output through static analysis. In 45% of cases the model introduced a known OWASP Top 10 flaw; on cross-site scripting it failed 86% of the time. Newer models wrote more working code, but no more secure code. [1]



Counterpoint. Agentic tools with scanners in the loop have produced no exploitable SQL injection or XSS in some testing; authorization logic remains the weak point. [2]

Less secure, and more certain

A Stanford team ran 47 developers through five security-related tasks, some with an AI assistant and some without. The assisted group wrote significantly less secure code, and was more likely to believe it was secure. [3]

“Such tools may lead users to be overconfident about security flaws in their code.”

Perry, Srivastava, Kumar & Boneh, ACM CCS 2023 [3]

A flaw you're worried about gets reviewed. A flaw you're sure isn't there ships. Type a query yourself and you decide how its inputs get in.

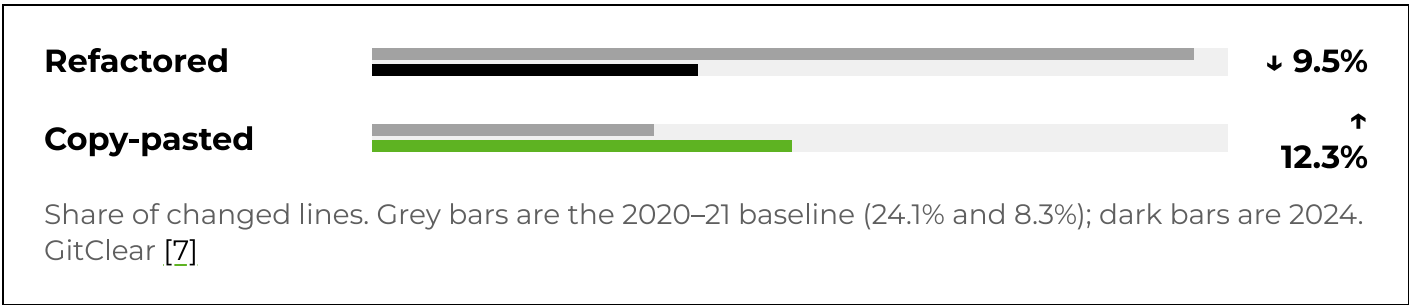
Caveat. An older model and a small sample; the overconfidence effect is the durable finding. [3]

Dependencies nobody chose

Across 576,000 samples from 16 code models, 19.7% of suggested packages didn't exist. [4] Attackers now register those invented names and wait for someone to paste the install line. It's called slopsquatting.

43% of invented names repeated on all ten reruns, so they're predictable. [4]	30,000+ downloads in three months for an empty package under a hallucinated name. [5]	4.6–6.1% hallucination range for 2026 frontier models. Lower, still exploitable. [6]
---	---	--

Code duplication raises a similar risk. GitClear's analysis of 211 million changed lines found copy-pasted code rising and refactoring falling as AI tools spread. [7] Patch a validation bug in one copy and the others stay exploitable.



Intuition is built by typing

We haven't found a study that measures this cleanly, so read it as our view: engineers who stop writing the sensitive parts of a system eventually lose the ability to judge them.

The 500ms case. In 2024, engineer Andres Freund noticed SSH logins running about half a second slow and traced it to a backdoor in xz Utils, rated CVSS 10.0. [8] No AI was involved. What caught it was someone who knew his system well enough that half a second felt wrong.

05 • WHERE AI HELPS

The same technology finds bugs people miss. Google's Big Sleep agent found a SQLite flaw that fuzzing had missed, and later one that Google says was known only to attackers, patched before exploitation. [9] People set the target and verified the fix; we recommend the same split for writing code. 46% of developers distrust AI accuracy. [10]

A checklist for teams

TYPE BY HAND

- ❑ Authentication, sessions and password reset
- ❑ Authorization: who can see and change what
- ❑ Input handling and output encoding
- ❑ Secrets, keys and cryptography
- ❑ Every package name, checked on the registry

LET AI HELP

- ❑ Scaffolding, boilerplate, type definitions
- ❑ Tests, especially adversarial inputs
- ❑ A second-pass review that attacks your code
- ❑ Explaining unfamiliar code before you rewrite it
- ❑ Prototypes and scripts you'd delete without worry

One test: if this line fails at 2 a.m., will the person on call understand why it was written this way? If the answer depends on reconstructing a prompt, it should have been typed.

SOURCES

1 Veracode, 2025 GenAI Code Security Report, Jul 2025. veracode.com

2 Cloud Security Alliance, "Vibe Coding Security Debt," Apr 2026. cloudsecurityalliance.org

3 Perry et al., "Do Users Write More Insecure Code with AI Assistants?" ACM CCS 2023. arxiv.org/abs/2211.03622

4 Spracklen et al., "We Have a Package for You!" USENIX Security 2025. arxiv.org/abs/2406.10279

5 Mend.io, "The Hallucinated Package Attack: Slopsquatting," 2026. mend.io

6 "The Range Shrinks, the Threat Remains," preprint, 2026. arxiv.org/abs/2605.17062

7 GitClear, AI Copilot Code Quality 2025; DevClass coverage, Feb 2025. gitclear.com

8 Elastic Security Labs, "500ms to midnight," 2024; CVE-2024-3094. elastic.co

9 Google, AI security announcements, 2025; SecurityWeek, Nov 2024. blog.google

10 Stack Overflow, 2025 Developer Survey press release, Jul 2025. stackoverflow.co