Why System Design Matters More Than Ever

The judgment layer AI cannot replace

What was actually built

> The demo looked real. It had a UI. It appeared to work. The danger isn't bad code that looks bad — it's **bad code that looks finished.**

Schema	Unknown — data relationships undefined or broken
Security	Unknown — auth, sessions, roles, token handling all unexamined
Scale	No concept of maintenance, or what happens when requirements change

PART 2

The Metrics

What the data actually shows

THE DATA

AI-authored PRs produce 1.7× more issues than human-authored

1.7×
more issues overall
vs human-authored PRs

75%
more logic errors
most common failure mode

2.74×
more security vulnerabilities
1.5–2.74× the rate

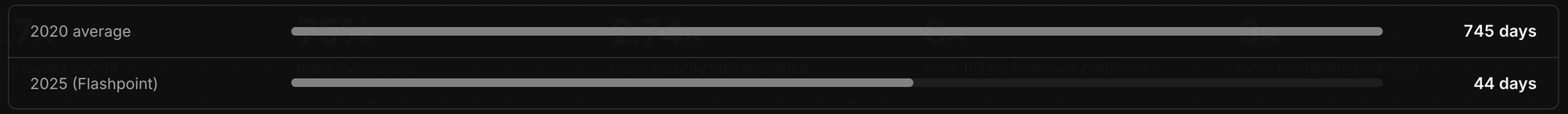
8×
more I/O performance problems
by far the largest gap

3×
more readability problems
maintainability

CodeRabbit “State of AI vs Human Code Generation,” Dec 2025 — 470 real-world open-source pull requests

SECURITY CONTEXT

The window from disclosure to first exploit is collapsing



The Hacker News — “2026: Year of AI-Assisted Attacks” (thehackernews.com/2026/05/2026-year-of-ai-assisted-attacks.html)

PART 3

The Job Market Signal

Read what companies are actually hiring for

HIRING DATA

What companies are actually hiring for

ROLE	POSTINGS (2025)	CHANGE
Security roles	66,800	+124% YoY
AI / ML roles	49,200	+163% YoY
Cybersecurity engineers (new)	20,000 added	growing
Open cybersecurity roles (US)	514,000+	+12% YoY
Entry-level general SWE	—	−28% from 2022

Robert Half 2026 Technology Hiring Report · CyberSeek 2025

THE SIGNAL

What this tells us

Judgment is the scarce resource

The scarce resource is no longer code output. Companies have plenty of that now.

Catch it, don't compete with it

Companies need engineers who can catch what AI produces — not engineers who try to out-code it.

Ask first, build second

The most durable skill: asking the right questions before anything is built.

PART 4

What System Design Actually Is

DEFINITION

System design is not an interview format

> The set of decisions that determine how a system **behaves, scales, fails, and changes** over time. It operates at every level — from a single function signature up to how services communicate.

Not diagrams	Not architecture whiteboards. The ongoing practice of deliberate decisions with full awareness of their consequences.
The key claim	It is the sum of all the choices AI cannot make well on its own

CLARIFICATION

What system design is not

Not this	Knowing one language well — valuable, but no longer sufficient on its own
Not this either	Being able to implement a feature quickly
The bar shift	Breadth of understanding now matters as much as depth. The coworker could implement. That's no longer the differentiator.

PART 5

The One-Way Doors

The decisions that calcify before you notice

ONE-WAY DOORS

Database schema and architecture are the clearest examples

Schema calcifies	Column types, normalization, index strategy — migrations are painful, risky, and usually avoided until they're critical
Architecture compounds	Wrong choices today become someone else's emergency in 18 months
Auth is expensive to retrofit	Auth built into a system not designed for it is enormously expensive. These are design tasks with security consequences — not "security tasks."
APIs lock you in	Once clients depend on an API, changing it is a breaking change

COMPREHENSION DEBT

The trade-off layer AI cannot see

> AI sees only the code — not **why your team made the decision it did.**

Invisible context	Undocumented constraints, past failures, team limitations — none of that is in the file
Design rationale	The most undervalued artifact in any codebase
Comprehension debt	The gap between how much exists in a codebase and how much anyone genuinely understands. Invisible until something breaks.

PART 6: THE HUMAN EVIDENCE

Two Interviews, Two Outcomes

THE CALL TO ACTION

We all need to be architects now

Decision-maker at every level

From variable naming to infrastructure choice — every decision is a design decision.

Not “learn more things”

The reframe isn't “I need to know more.” It's “I need to think differently about what I already know.”

Ask the right questions

The goal isn't to know everything — it's to know enough to ask the right questions and recognize bad answers.

PART 8

How to Build This

Concrete next steps

RESOURCES

How to actually get here

1. **Read these**
   System Design Interview Vol. 1 & 2 (Alex Xu) · Designing Data-Intensive Applications (Kleppmann) · Understanding Distributed Systems (Vitillo)

2. **Learn all the features of your language**
   Discriminated unions, records, type narrowing — whatever your stack exposes. These features exist to prevent whole categories of bugs.

3. **Use Claude as a teacher, not an oracle**
   Stress-test design decisions. Ask "what are the failure modes of this schema?" not "build me a schema." The judgment stays with you.

4. **Deploy real infrastructure with Railway**
   Low-friction way to experiment — spin up real databases, try migrations, feel what breaks before production does.

USING THE RESOURCE

How to actually work through it

The learning loop

1. Paste the resource into a new chat. Ask Claude to be your tutor and teach you the concepts one by one.
2. Give it a topic to start. Learn it. Then ask for quiz questions to test your understanding.
3. Occasionally ask for a case study — see the concept applied to a real scenario.
4. After the quiz or case study: things you didn't know go into a **Review** doc. Things you nailed go into **Mastered**. Move to the next topic.

The review cadence

- Every few days, share your Review doc and ask Claude to go through those topics with questions.
- After each question: decide — does it stay in Review, or move to Mastered?

> Two documents. One loop. The goal is to shrink Review and grow Mastered.

Resource

Take this with you after the talk.

Two documents. One loop. The goal is to shrink Review and grow Mastered.