CODEGUARD KICKOFF · A PRACTITIONER'S VIEW

Secure Software Engineering in a GenAI World

What actually changes when AI writes (most of) the code.

NabuTech - KYC/KYB startup - SAAS "oldie" - CS Lang Lover (FP)

Koen Handekyn · Quine & Partners · NabuTech

Co-authored by Cassandra (Koen's COS / PA) · Claude Cowork


PEACE OUT

WHY THIS CONVERSATION

The gap between how software is taught and how it is built has widened — fast.

The length of real tasks an agent can complete reliably is doubling.

×2

every ~6 months

METR · "task-completion time horizon" · measured on software tasks through 2025

The models improved, yes. But what really changed is the surrounding stack: context engineering, **preview infrastructure,** layered review, agent-aware workflows. Tasks that used to take a day sometimes ship in ten minutes.

A field report from running a multi-tenant KYC platform on this stack. Not a forecast. Not a vendor pitch.

THE SCALE SHOCK

In the next 5 years, more code will be written than in the whole history of software.

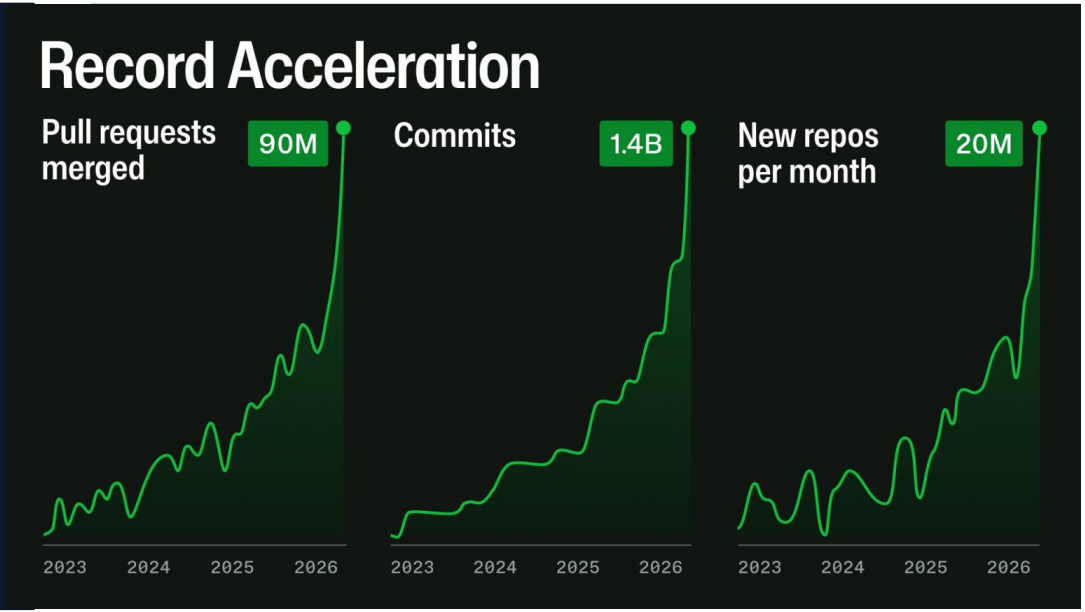
GITHUB · APR 2026

30X

today's scale

"By February 2026, it was clear that we needed to design for a future that requires 30X today's scale."

The claim is aggressive. GitHub's own planning horizon is the evidence under it.



THE CASE STUDIES ARRIVED

Major rewrites are moving from months to weeks — sometimes days.

BUN

+1M / -600K Rust rewrite

Rust rewrite merged; a Zig-removal PR was auto-flagged as "AI slop."

CLOUDFLARE

94% API Next.js surface

vinext rebuilt the Next.js API surface in under a week with one engineer directing AI.

LADYBIRD

25K / 2w C++ → Rust

LibJS parser/compiler components ported with AI; test suites reported zero regressions.

THE REVIEW PROBLEM

"The size of these commits makes them near-impossible for humans to review."

— The Register, May 14 2026

Why it matters for CODEGUARD

The pattern is repeatable.
Well-specified API or subsystem, strong tests, AI-assisted rewrite.

Reviewability becomes a gate.
The question is whether a human can validate the change surface.

This is maybe why PR size belongs in the quality model?

Sources: The Register on Bun, May 14 2026 · Cloudflare Blog on vinext, Feb 24 2026 · Ladybird "adopts Rust," Feb 2026.

THE SHIFT

(Me and my) Engineers type no code. They design *the constraints* that let AI write it.

The job is still engineering — just at a higher level of abstraction. Intent in, verified behaviour out.

HARNESS-BASED DEVELOPMENT

The code is (mostly) easy. The harness that verifies it is the real product of engineering.

Harness

The machinery around the code that tells us — automatically, continuously — whether it is correct.

Types, tests, linters, CI pipelines, schema checks, property tests, evals, observability. In 2026 these stop being 'hygiene' and become the primary deliverable.

Why it changed

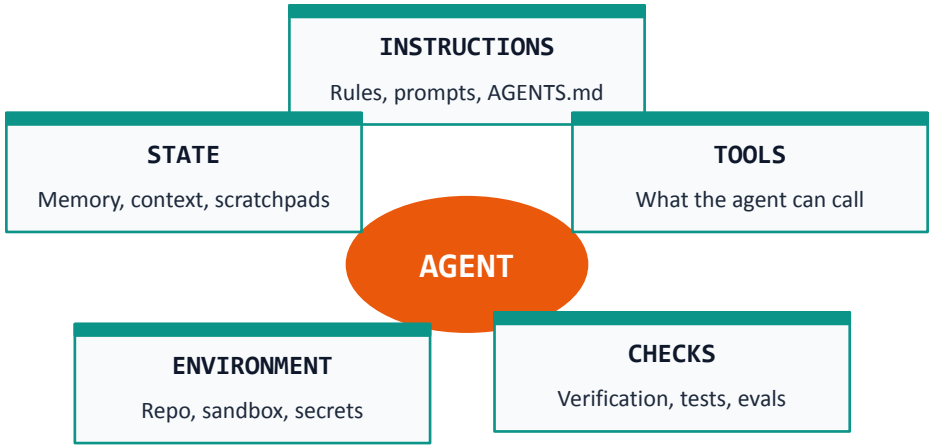
AI can now generate plausible-looking code at near-zero cost.

That makes the bottleneck verification, not authoring. You ship as fast as your harness can say `yes`.

Teams without a harness aren't slower — they're flying blind. Teams with a strong harness compound.

INSIDE THE HARNESS

Everything outside the model weights — five subsystems that decide what gets realized.



TICKETS AS THE PLANNING SUBSTRATE

GitHub issues are no longer a backlog. They are the work product.

The ticket is the brief.

Analysis, design notes, acceptance criteria, the files likely to change, the test plan — all live in the issue itself.

It's written for two readers at once: the engineer who supervises, and the agent that executes.

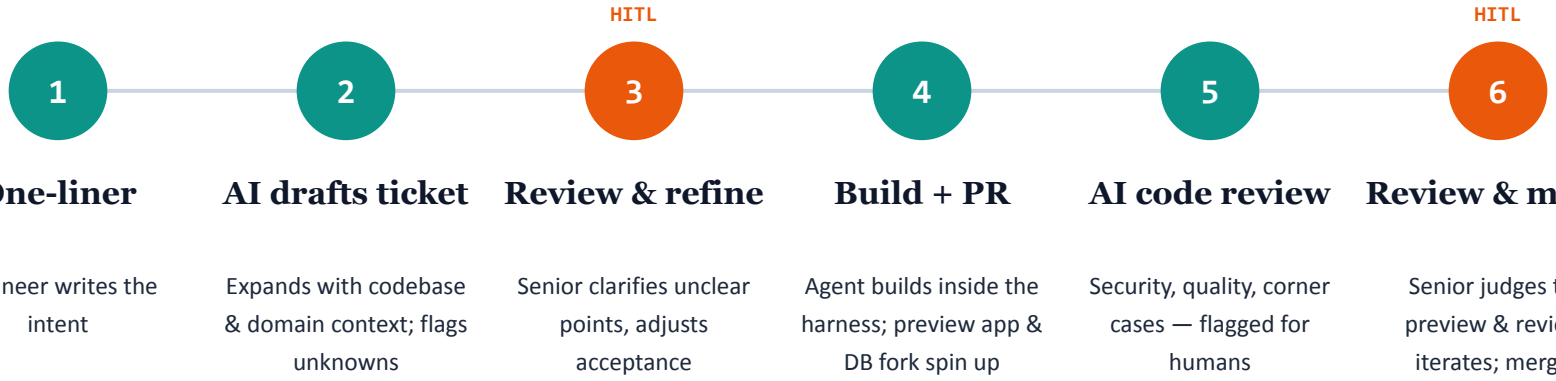
#842 · Scope participant doc uploads by case

```
## Context
Short-lived JWTs carry participant_id,
case_id. Enforce via RLS policies.

## Acceptance
- participant sees own docs only
- storage policy mirrors DB policy
- cross-tenant test suite passes

## Files likely to change
supabase/policies/documents.sql
```

Where do build plans and analyses live? Markdown in git? Confluence? Our pick: **`GitHub issues and PRs`** — the ticket is the brief AND the record. *(BUT we are likely going to change that soon)*

HOW WORK FLOWS

From a one-liner to a merged PR — with two clear human-in-the-loop moments.



↻

SECURITY REVIEW · *a second loop running around the main flow*

Continuous scan, **opens issues** for what it finds, **drafts PRs** that re-enter the flow at review. **OpenAI Aardvark (Codex security)** and **OpenAI Daybreak (May 2026)** are the current commercial examples — every commit, every PR.

THE REAL WIN ISN'T CODE SPEED

The biggest gain is understanding the real requirements faster.

Requirements are unstable when first written.

You don't really know what you need until you see something close to it running.
Cheap prototyping + realistic previews surface wrong assumptions early. The team converges on the actual problem faster — and that's worth more than any typing speedup.

AI didn't speed up coding so much as it sped up understanding.

THE OLD STORY

Spec → code → discover spec was wrong

Each loop measured in days or weeks.

THE NEW STORY

Sketch → preview → refine the spec

Each loop measured in minutes. The spec catches up to reality.

ONE DEVELOPER, SEVERAL AGENTS

git worktree turns one engineer into a supervisor of 4–5 parallel threads.

Multiple isolated working directories.

Each worktree = one agent session working on one ticket. They don't collide. You rotate attention between them.

4–5 concurrent threads is the sustainable ceiling. Past that the limit isn't the machine — it's review bandwidth and the context you can hold in your head.

For the first time in this wave, the AI is ready before we are. The bottleneck has flipped back to the human — which is exactly where it should be.

from 'waiting on ai' to 'ai waiting on me'

WHAT GETS HARDER, NOT EASIER

This mode is not a free lunch for the humans inside it.

THE QUIET STRESS

"Do we still have this under control?"

Code ships faster than any one human can fully trace.

The pace of change is relentless — tools, models, best practices all shift month over month.

Reviewing agent output all day is cognitively heavier than writing code yourself.

THE ROLE SHIFT

Developer → PM + manager of agents

Part product manager: understand the domain, write specs, judge what's worth building.

Part engineering manager: hand work out, review, give feedback — but the "reports" are agents.

Many engineers chose the career to avoid exactly this. That's a real transition cost.

Curricula need to prepare for this too — not just the tools, but the shape of the job.

THE REVIEW PARADOX

"What is good enough?" — the question no one had to ask before.

AI reviews are thorough. Too thorough.

Bot reviews flag every edge case, every potential improvement, every style nit. Often the suggestions are technically correct.

The problem: reviewing the review is harder than reviewing the code. Engineers report spending more cognitive energy deciding what to ignore than what to fix.

Signal-to-noise is the new bottleneck.

FROM THE TEAM

"Reviewing the code review is very difficult — a lot of edge cases, often it's actually true."

— Engineer reviewing AI output

"Too many times with stupid bot reviews."

— Senior developer on signal-to-noise

"It generates a lot of tests — some don't seem very important."

— On test noise from AI

If AI writes ½ as much bugs, but we code 10x faster, we still have 5x more bugs created

The fix isn't turning off reviews. It's tuning them: fewer rules, higher threshold, team-specific context.

THE SECOND-ORDER EFFECT

The harness learns!

Recurring agents, not heroic refactors.

Several agents run on a schedule against the repo. Their proposals enter the same review flow as any other PR.

HARNESS UPDATER · *weekly*
Mines the week's tickets, PRs and review threads. Proposes edits to AGENTS.md / CLAUDE.md and new reusable skills.

DOC CONSISTENCY · *daily*
Checks documentation against the code. Flags drift. Opens issues for stale READMEs and out-of-date examples.

CODE CLEANUP · *weekly*
Sweeps for dead code, duplication, lint debt and obvious simplifications. Drafts PRs.

FROM THE FIELD

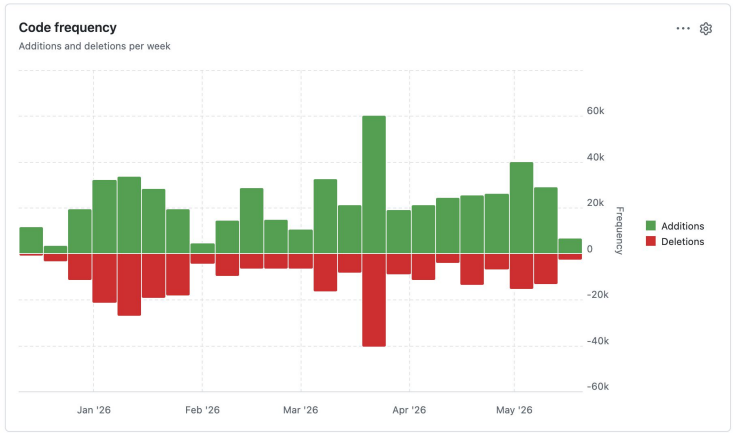
"Every harness component exists because the model can't reliably do something on its own. As models improve, these assumptions become outdated."

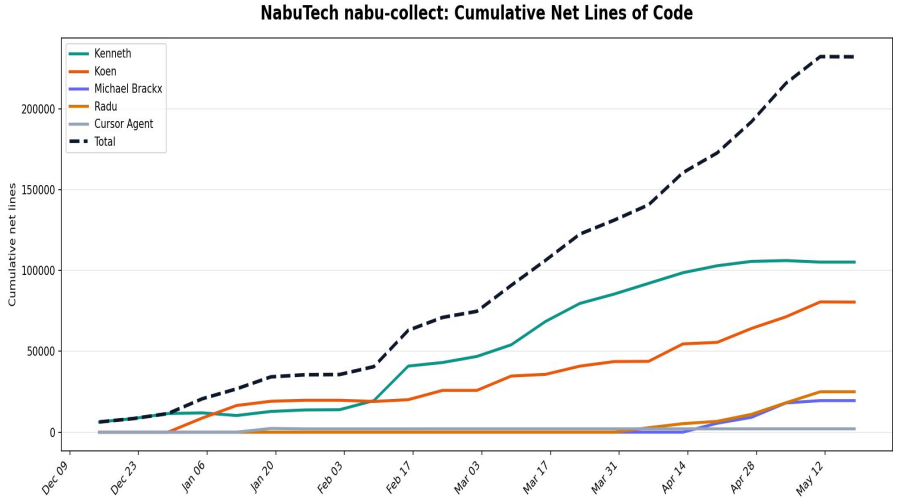
— *walkinglabs, Learn Harness Engineering, Lecture 12*

So the loop adds AND subtracts. *Each Friday we also ask: which harness rule is no longer earning its keep?*

THE COMPOUNDING EVIDENCE

215,000 lines of production code. Five months. A team that grew from 1 to 4 (2.5 FTE).

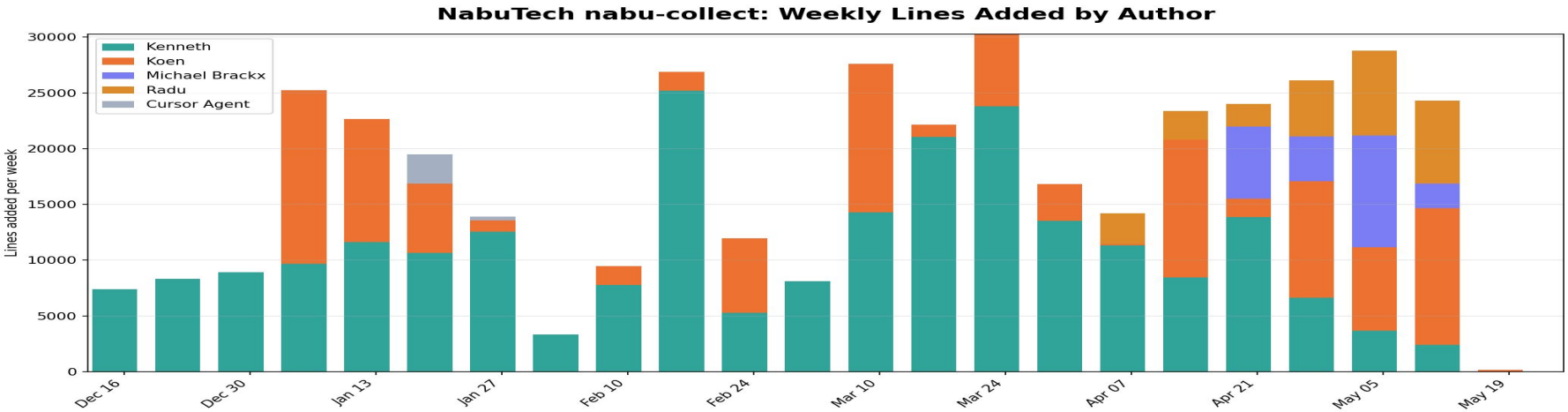




The slope doesn't flatten — it steepens. *Each week the harness catches more, so the next week ships faster.*

SMALL TEAM, OUTSIZED OUTPUT

Weekly lines added by author — and why the lead developer's output is dropping.



Notice Kenneth's (teal) output dropping as the team grows. *He's the gatekeeper — as more agents and developers ship code, his time shifts from writing to reviewing. The review paradox, visualised.*

1,769	170K	4	25K
commits in 5 months	lines TypeScript	developers	peak lines/week (1 dev)

THE QUALITY BOTTLENECK

The first review-quality indicator is brutally simple: how much changed?

PR size = `lines added` + `lines removed`

PICKING A STACK IN 2026

A new first-class criterion: how fluently AI writes in it.

Used to be

Best tool for the job

Language fit, type safety, performance, ecosystem depth, team experience.

A willingness to pay a learning-curve tax for a theoretically better tool.

Now

What the model writes fluently

Popular = well-represented in training = productive. Velocity on familiar stacks dwarfs the theoretical wins of niche ones.

The cost of 'clever' has gone up.

WHAT TEAMS ACTUALLY REACH FOR

A non-scientific but recognisable 2026 default stack.

Language	**TypeScript + Python**	*Highest AI fluency; covers most domains.*
Frontend	**React / Next.js**	*Massive training corpus; components generate well.*
Backend	**Node, FastAPI**	*Conventional patterns → predictable AI output.*
Database	**Postgres via Supabase**	*Chosen for RLS — not for convenience. See next slide.*
Infra	**Vercel previews, DB forks**	*Every PR gets a live preview; every review is visual.*
AI layer	**Codex (or Claude)**	*Interchangeable now. The harness is the moat, not the model.*

None of these are 'best' in isolation. The choices compose into a stack that AI is fluent in.

ARCHITECTURAL CHOICES MATTER MORE, NOT LESS

Pick architectures that localize correctness — so AI has fewer places to get it wrong.

CONFESSION: aardvark found several CVE due to RLS, but still positive on the choice and principle

The case study: Row-Level Security.

In a classic 3-tier architecture, authorization lives in the app layer — sprinkled across controllers, services and queries. An AI-generated route might forget the rules.

RLS moves those rules into Postgres. Define the policy once; it applies to every query, every client, always.

Same move as prepared statements for SQL injection — removes a whole class of bugs by construction.

BEFORE · 3-TIER

Auth rules scattered in app code

47 places to remember. AI can miss one. Humans do too.

AFTER · RLS IN POSTGRES

Auth enforced at the data layer

One policy. Applies to every query, every client, always.

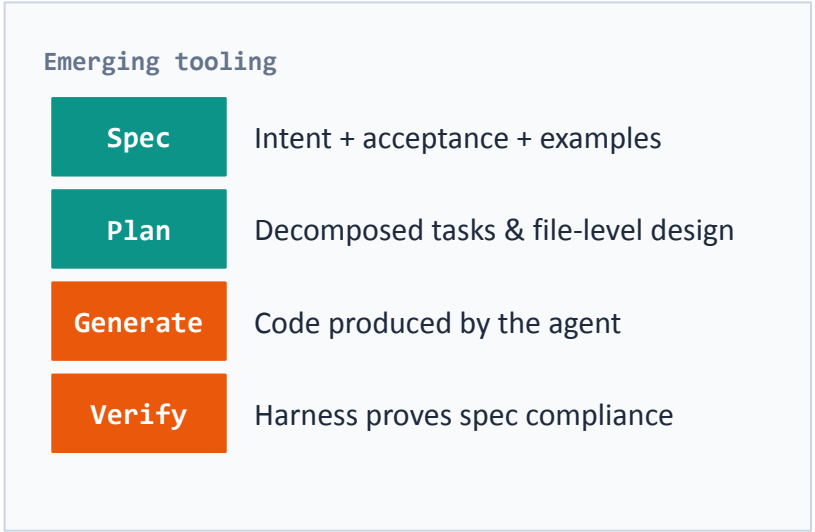
WHAT COMES NEXT

Spec-driven development: the spec is the source, code is the build output.

Intent first.

Write an executable specification — what the system must do, and how we'll know it does. Generate an implementation from it. Verify against the spec.

When requirements change, edit the spec. Regenerate. The code is no longer the precious artifact.



BEYOND CODE: VIBE-DESIGNING PHYSICAL OBJECTS

When CAD is programming, AI becomes your design partner.

OpenSCAD is a CAD tool where you program your designs.

No mouse-driven modeling. You write code that describes geometry: cubes, cylinders, boolean operations, parametric dimensions. It compiles to a 3D object.

This means AI can design physical objects. You describe what you want, an agent writes the OpenSCAD code, you preview, iterate, and print.

WHY THIS MATTERS FOR SE

PATTERN

Code-as-design spreads beyond software

Infrastructure-as-code, policy-as-code, now physical-objects-as-code. The AI leverage is universal.

VERIFICATION

Same challenge, different domain

How do you verify a 3D design is structurally sound? Same question as 'is this code secure?'

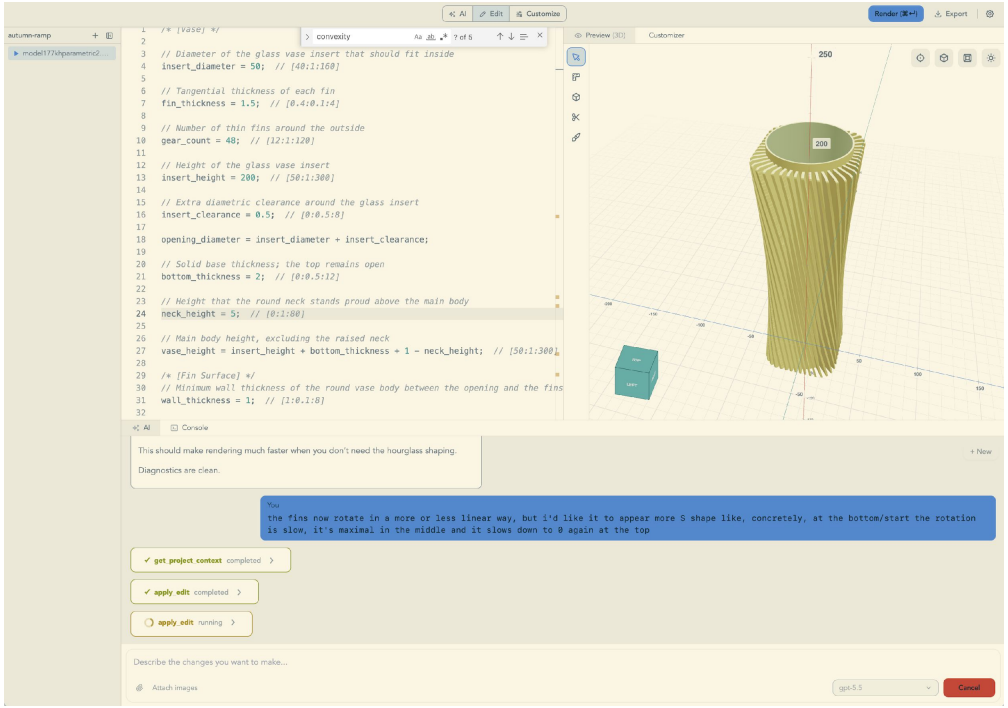
ACCESSIBILITY

Non-programmers can design

Describe what you want in words. The barrier to creation drops to zero. Quality assurance becomes the bottleneck.

VIBE-DESIGNING, LIVE

Code on the left. 3D on the right. Plain English at the bottom.



WHAT YOU'RE LOOKING AT

A glass-vase insert — parametric, fluted, printable. The code (left) defines diameters, fin thickness, neck height. The preview (right) updates as parameters change.

THE PROMPT

"The fins now rotate in a more or less linear way, but I'd like it to appear like a sine wave — concretely, at the bottom/insert the rotation is slow, it's maximal in the middle, and it slows down to 0 again at the top."

THE POINT

I am not a CAD designer.

I describe geometric intent in natural language; the agent translates it into parametric code; the preview tells me whether we agree.

AND SO IS THIS DECK

Slides as source: JavaScript in, .pptx out. Agents read, edit and remix it the same way.

Every slide you've seen is a block of JavaScript.

One file, ~2,000 lines, built with **pptxgenjs**. The whole deck is in git: diff-able, reviewable, regeneratable.

Which means Claude Code, Cursor, or any coding agent can author slides like any other source file. "Add a slide that…" → an edit → **npm run build** → done.

Same harness pattern. The .pptx is the preview; the .js is the truth.

build_deck.js

```
// SLIDE 2 — Pacing & what I'll skip
{
  const s = pres.addSlide();
  contentTitle(s, "Pacing", "Twenty minutes…");

  s.addText("I need everyone awake.", {
    fontFace: FONT_HEAD, color: C.coral,
  });
  footer(s);
}
```

WHEN THE AGENT MODIFIES ITSELF

A personal AI chief of staff that improves its own code, prompts, and capabilities.

Cassandra is an autonomous agent running as my chief of staff.

She monitors email across three companies, classifies and forwards invoices, prepares briefings, tracks commitments, and manages operational context.

The unusual part: she runs on a file-first runtime where she can read and modify her own prompts, skills, and code. When something breaks or could work better, she proposes the fix.

This presentation was built by Cassandra. The email requesting hotels in the Dolomites was written and sent by her. In Italian.

SECURITY IMPLICATIONS

SELF-MODIFICATION

The agent changes its own behavior

It can fix bugs in its email handling code. But what prevents it from weakening its own guardrails?

TRUST BOUNDARY

Human-in-the-loop, but for how long?

Today I approve external actions. Trust builds. Autonomy grows. Where's the safe boundary?

SUPPLY CHAIN

Skills, prompts, and tools as attack surface

A compromised skill file changes agent behavior silently. No binary to scan.

AUDIT TRAIL

Git as the security log

Every change is committed. But who reviews the commits of an autonomous agent?

Self-modifying agents are here. The security model for them isn't.

A PRACTITIONER'S WISH-LIST FOR CODEGUARD

Six research questions — with my hot takes.

1. Benchmark AI security reviews

How do LLM-based reviews compare to SAST, DAST, and professional pen-tests?

MY TAKE · *Complimentary*

2. Measure vulnerability incidence

Rate of security flaws in AI-generated vs human-written code, by category.

MY TAKE · *Irrelevant soon*

3. Self-modifying agent security

When agents change their own code and prompts, what's the trust/verification model?

MY TAKE · *Soon key*

4. Injection in dev tools

Can a malicious dependency or README inject instructions into an agent's context?

MY TAKE · *Interesting*

5. Review fatigue quantified

How much security signal do developers miss as AI-generated PR volume increases?

6. Harness quality metrics

Can we measure whether a project's harness is 'strong enough' to catch AI-introduced vulnerabilities?

AI makes secure code cheaper than ever.

The harness is the security architecture now.

Build build build.

Koen Handekyn · koen.handekyn@quine.be · +32 499 69 05 71

Quine & Partners · NabuTech