Two developers, three decades apart

—

the bottleneck was never typing

Jens Klein · Torben Baumgartner

PloneConf 2026 · Business Track · Possibility

notes · 1 — Two developers, three decades apart — the bottleneck was never typing

[Notes legend: [regie] · "fixed spoken lines" · free text = speak freely]
[JENS — slide stays up; never read it. Both on stage from the start,
Torben relaxed. No "thank you for having us" — straight into the story.]
2020, Dresden, an open-space session at the German Plone Tagung: moving image scaling out of the database. Everyone agrees it should happen. There is consensus, there are ideas. Nobody builds it.

2020 · 2024 · today

- 2020 — everyone agreed. **Nobody built it.**
- 2024 — a few hundred hours, a five-figure sum. **Not commissioned.**
- today — **open source**, twelve sites migrated onto it

notes · 2 — 2020 · 2024 · today

[JENS]
2024: a customer asks me for exactly that. I quote it — three variants on one base, a few hundred hours, a five-figure sum.
It was not commissioned.
"The idea was fine, the price was not."
And the customer was right — at that price it was not worth it.
Today it is built, it is open source — and all twelve sites of the Austrian Architecture Forum are migrated onto it.
[Never "in production" — say "migrated onto it"; cutover is after the conference. No customer name, no exact figures, no hint the customer decided wrongly.]

"The bottleneck was never typing.
It was boundaries."

notes · 3 — The bottleneck was never typing. It was boundaries.

[JENS — the thesis, once, plainly. Slow.]

"The bottleneck was never typing. It was boundaries."

Boundaries of capacity, of skill, of roles. And what an organisation does with the space they leave behind — that is the story of the next forty minutes.

[Avoid the word "AI" this early if possible — the story carries without it.]

Who's talking

- **Jens Klein** — 34 years in IT · Plone core developer since 2003
- **Torben Baumgartner** — AI Integration Engineer · more dimensions GmbH

notes · 4 — Who's talking

[JENS — introductions, short. The single family mention, then never again:]

"I'm Jens — 34 years in IT, Plone core developer since 2003. Torben is an AI integration engineer — and my son. Born the year before I joined this community."

[HANDOVER to Torben. He introduces himself through the thesis, not a CV.]

April 2021

- diploma thesis: **image recognition**
- six months of work
- **19 months before ChatGPT**

notes · 5 — April 2021

[TORBEN]
My diploma thesis: image recognition. Six months of work.
Submitted April 2021 — nineteen months before ChatGPT.
About AI, without AI. The last cohort that had no choice.
"The result would be twenty minutes of work today.
The learning would not."
[Never do the division — no factor, no math. The juxtaposition carries.]
That gap — six months to twenty minutes — is what this talk explains.
And the second half of that sentence is what the end of this talk is about.

Working with AI — the ladder

1. **Prompting** — asking well
2. **Shaping context** — your docs, examples, counter-examples, tools
3. **Training** — you will almost never need it

notes · 6 — Working with AI — the ladder

[TORBEN]

Everyone in this room hears something different when someone says **"AI"** — so here is the map we actually work with. Three rungs, by cost:

Prompting — asking well. Cheap.

Shaping context — giving it your documentation, your examples, your counter-examples, your tools. Medium effort. Fine-tuning? A technique inside this rung, not a rung of its own.

Training — building or rebuilding a model. Expensive, and you will almost never need it.

The message this room does not expect: nearly all the value is on rungs one and two. **"Our own AI"** does not mean **"our own model"**.

[Introduce the words "ladder" and "multiplier" — the website teasers use both.]

Sovereignty lives on rung two.

Own the context — not the model.

notes · 7 — Sovereignty lives on rung two.

[TORBEN]

Sovereignty lives on rung two: your domain knowledge, your docs, your tools, your access rules.

You do not have to own the model — you have to own the context you give it.

That is affordable. And it is yours.

"Which model should we buy?"

- the variable that matters **least**
- and the only one you **don't own**

notes · 8 — Which model should we buy?

[TORBEN — the spoken lead-in is part of the delivery, never drop it:]

The first question in this room is usually: which model should we buy?

We all carry a scoreboard in our heads — the public rankings.

"Honest answer from our own work: for your own use, the top twenty perform much the same. You are optimising the variable that matters least — and the only one you don't own."

[No model names, no rankings, no second number.]

[Then PLANT — we come back to this at the very end:]

"The model is rarely the limit today. The limit is everything between the model and the person — the tools, the context, the way of working. Most of that layer is still primitive. Keep that in mind."

[If running late, this beat is the first cut of the whole talk.]

Case A — two teams, one handoff

- web team: knows the **customer** — can't write the DSL
- backend team: writes the **DSL** — doesn't know the customer
- briefings · queues · guessing · extra cycles

notes · 9 — Case A — two teams, one handoff

[TORBEN — slow down. The centre of the talk. A story about two teams, not about a tool.]

Our product runs on a domain-specific language — think of it as a small programming language that exists only for this one product.

A customer wants a change: ninety percent is web work. But almost always, something in that language needs adjusting too.

The web developers don't write it. The backend team does — but they don't know the customer. So: briefings, queues, guessing. And guessing sometimes went wrong — extra cycles.

The bottleneck was a knowledge handoff between two teams. Who had the customer couldn't write the language; who had the language didn't have the customer. Never typing speed.

[Do not name the product or the DSL — not cleared.]

What we changed

- documentation · examples · **counter-examples**
- a connection to the **running systems**

notes · 10 — What we changed

[TORBEN]

We taught the assistant our language: documentation, examples — and explicit counter-examples. **"Not like this"** matters as much as **"like this"** for a language with quirks.

Today the web developers describe the change in plain text and get working code in our language back.

And the turning point:

"The big jump did not come from a better model. It came when the assistant could look things up in the running system instead of guessing."

It reads the customer's actual custom formulas — it looks up instead of guessing.

[Avoid "fine-tuned" — it will be heard as model training. Avoid bare "MCP".]

What happened to the company

- the handoff is **gone**
- backend team: **retiring legacy** · modernising
- **the company is hiring**

notes · 11 — What happened to the company

[TORBEN]

The web team handles customer changes end to end now. The handoff is gone.

The company can take on more front-end work, serves more customers —

"The company is hiring."

Front-end positions. Not a forecast — a job ad.

The backend team got their time back. Where did it go? Retiring legacy code. Containerising. Modernising.

"They got to the real work."

The bottleneck did not vanish — it moved, to the customer interface.

A bottleneck on the demand side is a good bottleneck.

One sentence on sovereignty: the knowledge about our own language went into a form our own team can use — not to a vendor.

Case B — AI as an invisible component

- a large German online retailer for Italian specialities
- categorising **support email**

notes · 12 — Case B — AI as an invisible component

[TORBEN — faster tempo than Case A.]

Previous job: a large German online retailer for Italian specialities. Not a tech company — that is the point.

The task: categorising incoming support email. With rules and heuristics that is a monster project. With AI it is a manageable one.

And here the AI is not a chatbot — it is an invisible component in the workflow. As a customer you never see it.

When does this work? Three conditions

1. the result need **not** be deterministic
2. an error rate is **acceptable** (~2 %)
3. massive development effort **saved**

notes · 13 — When does this work? Three conditions

[TORBEN — the checklist to photograph.]

When does this work? Three conditions:

The result does not have to be deterministic. An error rate is acceptable — think around two percent. And in exchange you save massive development effort.

All three true — then very complicated things become coverable. The errors are the price, and the price is non-negotiable.

This was the rare case for the top rung of the ladder: a small model, trained on an open-weight base — narrow domain, huge volume.

Rule of thumb: much variety, low volume — context is enough.

Low variety, high volume — training pays.

[Do not stress where it runs — that is Slot 4's stage this afternoon.]

Same headcount. Deeper work.

- easy half — **automated**
- complex cases — **more time**

notes · 14 — Same headcount. Deeper work.

[TORBEN]

The effect: same headcount. Nobody was cut.

The easy half is automated. The freed time goes into the complex cases — which now get handled deeper.

Transparency rule, one sentence: the customer doesn't see the AI, but the people working with it know it is involved — and they can correct it. Invisible is not concealed.

Land the pattern: three organisations — a software company, a retailer, and one more coming — same move every time: freed capacity went into capability, not into cuts.

[HANDOVER to Jens — mark the line shift: Torben talked about what changes inside one company; Jens now — what changes underneath all of them, the shared open foundations institutions run on.]

24 November 2025

Not a single line of code by hand
until Christmas Eve.

notes · 15 — 24 November 2025

[JENS — self-deprecating entry.]

34 years in IT. My habits are my best asset. They were also the problem.

24 November 2025 — a new model generation ships. I make myself a rule: not a single line of code written by hand until Christmas Eve. One month. No exceptions.

What happened: the habits broke. I learned to prompt. And the real shift — from improvising with a chatbot to structured, specification-first work: write down what to build, let the machine build it, test against the spec.

Why this matters to this room — the honest answer to **"how do I get my senior people to use this"**: the obstacle is not the tool, it is the habits. And your best people have the strongest habits.

"It changes reach, not worth."

notes · 16 — It changes reach, not worth.

[JENS — the suit image, spoken; no imagery on the slide.]

"Without the suit, Stark is not nothing — he's brilliant. He's just not an Avenger. The suit changes reach, not worth."

And a suit without Stark doesn't fly.

For this room:

"Your best people stay your best people. What changes is how far they get."

Same developer, before and after

- 2024 — deployment tooling, **written by hand**
- 2026 — a family of packages in **six months**

notes · 17 — Same developer, before and after

[JENS]

The before/after has a control group: me. I wrote the Kubernetes deployment tooling for Plone by hand, starting 2024. I could do this work before — it still didn't happen at scale. That is the point.

Then the habits broke, and it exploded: a family of packages in six months. Not random features — the things this community had discussed for a decade and never built. For lack of capacity. Nothing else.

For the buyers in the room:

"The open platform you depend on can now deliver what sat out of reach for a decade. Not because anyone got smarter — because the queue got shorter."

2020 → 2026

- image scaling out of the database
  discussed — open space, German Plone Tagung, Dresden 2020 → **built**
- Plone on PostgreSQL
  argued on stage — 2020 → **built**

notes · 18 — 2020 → 2026

[JENS — precision matters; the two receipts are not equally hard.]

The documented one: image scaling out of the database. An open space at the German Plone Tagung, Dresden 2020. Everyone agreed, nobody built it. That is the item from the cold open — discussed 2020, priced 2024, built 2026.

The direction — tell it as biography:

"I argued Plone on PostgreSQL on stage in 2020 — with the tool that existed then. The native successor stayed what it was: conversations after the talk. This year it got built."

[Never claim the 2020 talk contained the 2026 design — it showed the tool of the day.]

How it was built

- **specification first**
- 44 commits in week one — not one big drop
- more test code than production code — **1.4 : 1**

notes · 19 — How it was built

[JENS — practice description, no adversary, one breath:]

"Specification first. Forty-four commits in the first week, not one big drop. And more test code than production code."

[No spoken decimals — the exact ratio lives on the slide. If a spoken figure is wanted: "nearly one and a half times as much".]

Cloudbrine

- a new **storage & deployment foundation** under Plone
- **open source**
- the technology was its own talk — yesterday. *Slides online.*

notes · 20 — Cloudbrine

[JENS — two sentences, no more:]

Cloudbrine is a new storage and deployment foundation under Plone — the layer that keeps every page and file safe. And it is open source.

"The technology behind it was its own talk — yesterday, at this conference. The slides are online."

[No recording promise — upload timing unknown.]

Austrian Architecture Forum

- **twelve organisations · one shared platform**
- web archive back to **2002**

notes · 21 — Austrian Architecture Forum

[JENS — names allowed and wanted; slow down on the numbers, they are the credibility.]

Where the stack actually is: the Austrian Architecture Forum — the network of the country's architecture houses. Practically one per state, plus the national bodies, plus a festival.

Twelve organisations, one shared platform.

And the web archive going back to 2002 came along — twenty-four years of institutional memory, kept.

The migration

- Plone 5.2 in Docker → **Plone 6.2 on Kubernetes**
- **fully open source — end to end**
- all migrated · final acceptance · go-live = a **DNS switch**

notes · 22 — The migration

[JENS]
The migration they had deferred for years became possible: from Plone 5.2 in Docker to Plone 6.2 on Kubernetes — an outdated release brought current, onto the infrastructure layer the industry standardises on for running software in any data centre.
And it stays fully open source, end to end — nothing proprietary between the CMS and the hardware. Runnable in-house or at any European provider.
Status, calm, one beat:
"Every site is migrated and running. We are waiting for the go from the customer, who is testing the migrated sites — which is quite a lot of work — and we are fixing the last minor glitches."
[GLITCHES, never bugs. Never "in production". No blame, no reasons beyond this sentence.]

Version lock-in,
not vendor lock-in.

"The price changed, not the platform."

notes · 23 — Version lock-in, not vendor lock-in.

[JENS]

What they had was version lock-in, not vendor lock-in — locked to an old release by the sheer work an upgrade needed. They never wanted to leave the platform. They couldn't afford to stay current on it — the upgrade never made it to the top of any budget, for years.

That boundary fell. The freed capacity went into the upgrade everyone kept deferring — the same move as the backend team in Case A.

"The price changed, not the platform."

Go. Rust.

- an operator in **Go** · a transcoder in **Rust**
- *neither is a language I can write*
- `github.com/toalba/wows-cb-tracker` — **public**

notes · 24 — Go. Rust.

[JENS opens:]

This year I shipped a Kubernetes operator in Go and a high-speed transcoder in Rust. Neither is a language I can write.

An operator — software that runs and heals other software. A transcoder — a converter that has to be very fast.

[HANDOVER — TORBEN takes the floor and keeps it:]

Same on my side — my hobby project this year is built in both:

"I'm fluent in neither. Rust for the part that runs in the browser, Go for the part that collects the data — each picked because it fits the job."

And it's public.

[The invitation to check lives on the slide — the link, not a challenge.]

What still matters

being able to **program**	**mandatory**
mastering the language	no
a specialist per language	no

notes · 25 — What still matters

[TORBEN — the three levels; keep them apart:]

Being able to program — still mandatory. Non-negotiable.
Mastering the particular language — no longer required.
Hiring a specialist per language — no longer required.
Languages still exist for reasons — Go is right for operators, Rust for speed. The point: you can now choose the technology by fit, not by what your team already knows.

[HANDOVER — JENS takes it back and holds it into the next block:]

Organisations choose technology by their people's skill inventory, then live with that choice for a decade. Nobody calls it lock-in. It is — skill lock-in. And it is dissolving.
For hiring that means: good generalists over language specialists.
What stays human: the mechanics, the same in every language — testing awareness, security awareness, error handling, operations.

[Calibration: "not as important anymore" — never "doesn't matter".]

How do you learn this?

notes · 26 — How do you learn this?

[DIALOGUE ZONE, minutes 36–43, rehearsed as one stretch. Slide stays up.]
[JENS — hands over with the recommendation:]
"If you take one thing from my half: force the habit change. A month of not writing a single line — that's my recipe."
[TORBEN — direct, no pause:]
"I would never advise a beginner to do that."
You need knowledge. A language. Architecture — the machine never does that part fully. Fundamentals first — without AI, or with AI as your explainer. And that takes discipline.
[TORBEN — the picture; translate "aim assist" in the same breath:]
Think of a game that adds aim assist as a new feature — the game aims for you. Aiming is the one mechanical skill that kind of game is built on; it decided whether you got to play. Writing code was that skill for us.
"It took the skill that let you in. It never touched the one that decides how far you get."
[TORBEN on Jens:]

You played this map for thirty-four years. Every route, every climb. What you never did is play with the assist on — it did not exist when you learned. Your month without a line was exactly that: switching on the new feature, knowing the ground.

[TORBEN on himself:]

I learned the map without the assist too — my thesis was 2021, before any of this existed. The last cohort that had no choice.

[TORBEN on today's beginners:]

"Switch it on and win in week one — and never leave the spot you are standing on. Or learn the map first, lose for a while, and then go anywhere."

[JENS — takes the picture back:]

That is why my recipe is right for me — and wrong for them. The senior's half: trust the aim assist, and let go of your aiming skills. Thirty-four years of aiming — and the discipline is in the letting go.

And why that is safe for me and not for them: I can still look at the

assisted shots and tell a good one from a bad one.

"The aiming goes. The eye stays."

The eye only stays because I aimed for decades.

"He has the map and needs the assist. They have the assist and need the map."

notes · 27 — He has the map and needs the assist. They have the assist and need the map.

[Switch to this slide only AFTER Jens has spoken the line.]
[TORBEN — his standing, one line, said plainly:]
"I got the map first and the assist second. That was timing, not virtue."
[TORBEN:]
"The right recipe depends on where you start."
[JENS — lands it:]
"And both cost the same thing: discipline."
The beginner's is resisting the assist — mine was trusting it. At both ends of a career.

multiplier × 0 = 0

The factor was always the person.

notes · 28 — multiplier × 0 = 0

[JENS — give it room; the suit comes back:]
Assist, suit, model — all the same thing: a multiplier. It changes reach, not worth. And multiplier times zero is zero.
The suit doesn't make an Avenger out of nobody; the assist doesn't make a player out of someone standing where they started.
"The factor was always the person."
[TORBEN — the room's real question; keeps the floor into the outlook:]
"Can I hire juniors and hand them AI?" — Not by itself. It works under conditions — and you just heard them.

The future as we see it

- programming in **plain English**
- the job: **architect · orchestrator**

notes · 29 — The future as we see it

[TORBEN — marks the mode switch explicitly:]

"This is no longer a report. This is the future as we see it."

Where this might go: people programming in plain English, needing domain knowledge — and when complexity grows, the job becomes architect, orchestrator.

[No dates, no roadmaps, nothing that sounds like a product prediction.]

Who aims tomorrow?

- juniors: **design discussion**, not review queue

[JENS — takes the worry; it is his to take:]

If nobody writes code anymore — where does the judgment come from?

Back to the picture from five minutes ago:

"The eye only stays because somebody aimed. Who aims tomorrow?"

Where do the next architects grow?

A partial answer, from my own practice — the CTO of Thoughtworks wrote this in September: review is overloaded; judgment has to move before the code, into specification and design. Then education moves with it:

"If you want your juniors to grow, put them in the design discussion, not in the review queue."

That is a Monday-morning change.

[TORBEN — qualifies, softly, once:]

A seat at the design table is not a workshop. It doesn't replace having built things yourself. Both can be true — the question goes to the closing panel this afternoon.

[Optional, first cut in this block:]

"A specification captures what you can say. Not what you know."

The next leap does not need a better model.

Everything you saw today: first-generation tools.

notes · 31 — The next leap does not need a better model.

[TORBEN — harvest the planted sentence:]

Remember the sentence from the map: the limit is rarely the model.

It's the layer between the model and the person — the tools, the context, the working method.

Everything you saw today — the DSL case, the storage foundation, twelve organisations — was done with the first generation of those tools. Primitive ones, honestly.

"The next leap does not need a better model."

It needs better tools around the models — and unlike the models, that layer is something every organisation in this room can own, build and buy today.

The open question we leave the day with: what happens when that layer gets good?

[HANDOVER to Jens for the close.]

What fell — what didn't

- capability boundaries — **fell**
- judgment — **stayed**
- verification — **got heavier**

notes · 32 — What fell — what didn't

[JENS — landing only, no new material. Optional opener:]

"We are not going to pretend this is free."

"The bottleneck was never typing. It was boundaries. The boundaries about capability fell. The ones about judgment did not — and the ones about verification got heavier."

[If time is tight: first two sentences only.]

The boundary you built your organisation around may no longer exist.

Go and check.

[JENS]

"The boundary you built your org chart and your architecture around may no longer exist. Go and check."

[Then the community line — let it stand alone, nothing after it:]

"For a decade we had every freedom to improve our platform.
The freedom was real — exercising it was priced out.
That is the price that changed."

This afternoon — this stage

- Panel: **Digital Autonomy & AI** · 16:15
- next: **Stefano Marchetti** — reuse as a sovereignty practice

notes · 34 — This afternoon — this stage

[JENS — own it, park it, invite. Three sentences, one gesture:]

"We leave that question open — on purpose."

[Half-second pause, voice down.]

"This afternoon, a quarter past four, everyone who spoke today is back on this very stage."

[Point at the floor — it is literal.]

"That is where it gets argued. Come and watch us disagree — and bring your own questions."

[One full breath. Then the bridge to Stefano — exact wording with Alex. Say the time only if checked Friday morning.]