The Best Part of Programming Was Never Typing Code

PHUSE Single Day Event, Basel

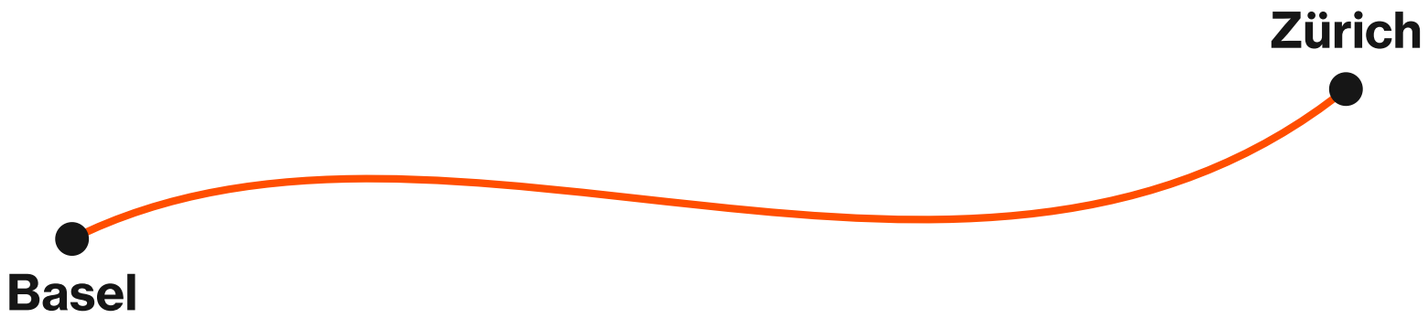
Tiina Kirsilä
Novartis AG

September 17, 2026

Disclaimers

- Views and opinions are the author’s own, not the affiliated organization.
- Accurate to the best of the author’s knowledge at the time of preparing the presentation.

Basel → Zürich · 1h 13min



And it was magical.

NOVARTIS

Why that magical moment scared me

The best moment of my year was the most unsettling moment at the same time.

Why programmers like to write code in the first place

- Problem solving
- Creativity
- Flow state
- The satisfaction of building something that works

Two things that have made our clinical data analysis work special

Both broke on that train.

Illusion #1: Few people can do this

Being one of the few felt like value.

1h 13 minutes; anyone with the prompt can do it too.

Illusion #2: The code was the one thing I fully controlled

So much of our job is outside our hands:

- Shifting pipeline priorities
- Delayed data
- Endless cross-functional alignments

Inside my code, I was in control. It was the one room that was mine.

On the train, even that room stopped being only mine. It was invaded by AI. Soon it would be invaded by any colleague with a prompt.

Same answer, different code

In the early days of genAI, it didn't write well documented, elegant code. By the time we reached Möhlin, it had already written more clean, documented code than I could have in a day.

Elegant and correct

```r
safety_pop <- adsl |>
  filter(SAFFL == "Y")

safety_pop |>
  summarise(
    n_subjects = n(),
    mean_age   = mean(AGE),
    .by = TRTA
  )
```

Unclear but correct

```r
f=function(d){
d=d[d$SAFFL=="Y",]
g=unique(d$TRTA)
n1=c();x=c()
for(i in 1:length(g)){
tmp=d[d$TRTA==g[i],]
n1[i]=nrow(tmp)
x[i]=sum(tmp$AGE)/n1[i]
}
r=data.frame(g,n1,x)
names(r)=c("TRTA","n1","x")
r
}
f(adsl)
```

Did my job just become a commodity?

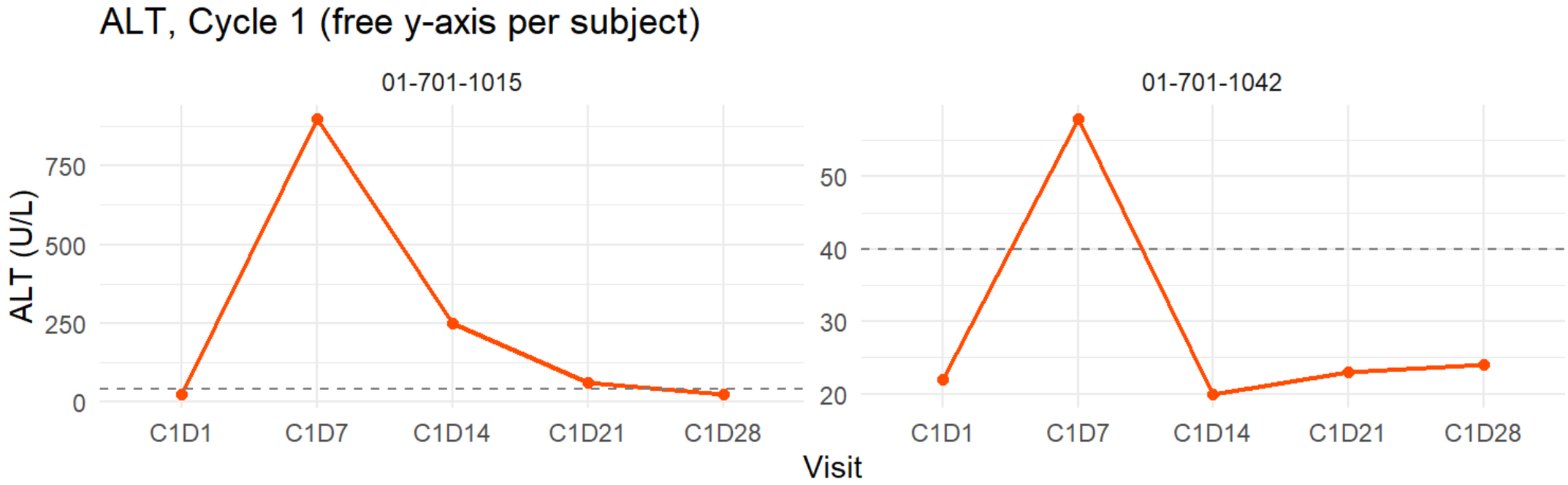
But the app looked perfect

- Tests passed
- It ran, reactivity worked
- It looked just like I’d designed it

Then magics started to turn back to reality

The y-axis wasn't consistent across patients for labs.

The clinicians need to see potential safety concerns easily.

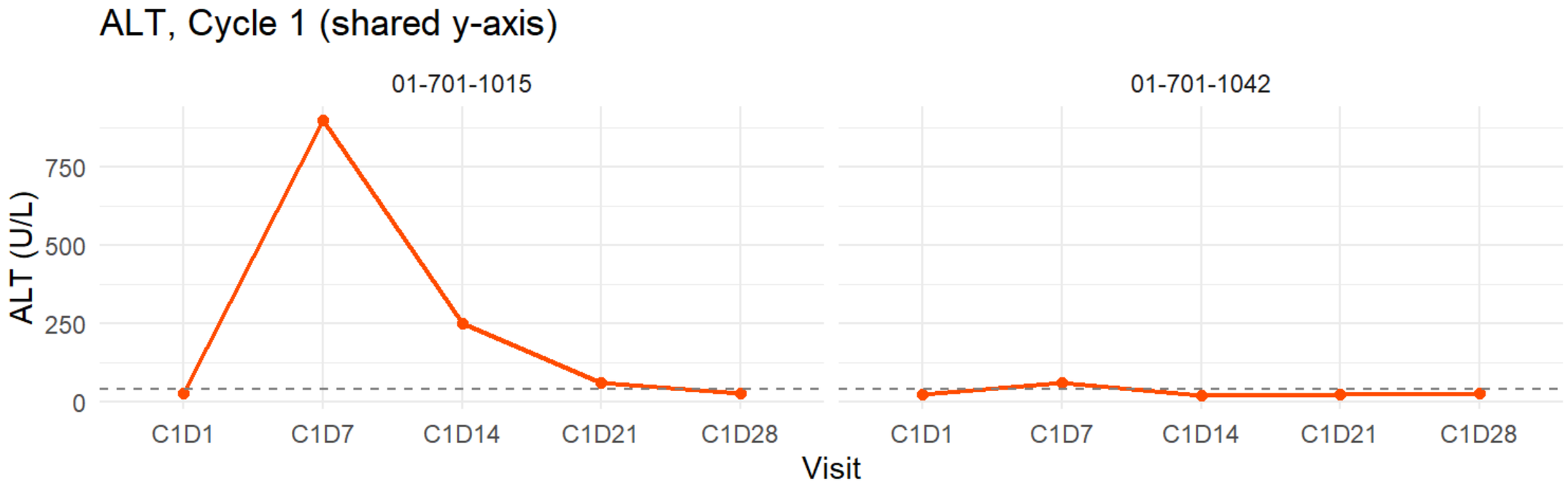


Simulated data, for illustration only.

NOVARTIS

Back to basics

AI didn’t catch it. The tests didn’t either. I was back to using the everyday knowledge we have.



Simulated data, for illustration only.

NOVARTIS

So who built the app?

AI typed it.

I knew the architecture I wanted, the requirements for MVP, and how to ensure scalability from the get-go. I planned it. I caught the typical issues. I was orchestrating the whole process.

The typing was never the only value. The train ride proved it.

This isn’t new

> “The proper, primary aim of programming is … not to produce programs, but to have the programmers build theories of … how the problems at hand are solved.”

Naur, P. (1985). *Programming as Theory Building*.

Forty years ago. AI just made it impossible to ignore.

Impact on knowledge work

- Anyone can find a plausible diagnosis
- Anyone can prompt a plausible app
- Generation is democratized

What isn’t democratized

- Knowing when the plausible answer is silently wrong
- Willingness to sign your name

The signature

A doctor’s prescription isn’t valuable because it’s hard to type. Anyone can.

It’s valuable because an accountable person stands behind it.

GxP is the same. Traceability and reproducibility are the signature.

It only means something if the signer understands what they signed.

If the value was the typing

I still code. A lot. Because I still love it.

Producing the numbers per specs is the easy part. The job continues to be knowing where they came from, how we got them, and why they’re right.

That part - being trusted with “is this right?” - the train couldn’t touch.

...and then what?

- I was still the one solving a problem
- I was still the one being creative. AI obeyed my logic and creativity.
- At first, on the train, it didn't feel like I *built* it. It did only once I took the time to review it properly later. And that was still *fun* and *satisfying*.

Is that enough?

Mystery yet to be solved: does labor lead to love when it comes to code?

There's research: the 'IKEA effect'. We value and care for what we build ourselves more than what we're handed.

So here's the real risk of AI-written code:

1. If we didn't put enough labor into it, do we still care enough to catch when it's wrong?

2. If we didn't put enough labor into it, will we understand it?

3. If we didn't put enough labor into it, will we maintain it?

More ideas than time - until now

I’m sure you and your teams have had more ideas than time. The time consuming typing was the bottleneck that killed most of them.

The app built on the train was a 1h and 13-minute execution of an idea that died in the backlog years ago.

Now, months after the train ride, much more ambitious ideas have emerged.

The best part was never typing code

It was imagining something that didn’t exist yet

and knowing we could build it.

Now we can imagine bigger.

Thank you

Tiina Kirsilä

tiina.kirsilae@novartis.com

linkedin.com/in/tiina-kirsila

NOVARTIS

How this talk was made

AI helped with:

- Slide styling & grammar (Claude Haiku 4.5 via Posit Assistant)
- Literature search (Claude 4.8 Opus)

AI helped to type this. The thinking and the accountability are mine.

References

- Naur, P. (1985). Programming as theory building. *Microprocessing and Microprogramming*, 15(5), 253–261. https://doi.org/10.1016/0165-6074(85)90032-8
- Norton, M. I., Mochon, D., & Ariely, D. (2012). The IKEA effect: When labor leads to love. *Journal of Consumer Psychology*, 22(3), 453–460. https://doi.org/10.1016/j.jcps.2011.08.002