

A FIELD STORY FROM A REAL SAP PROJECT

# Build in SAP with *AI*

*What really happens when an ABAP veteran and an AI assistant replace a legacy transaction, and how to do it without getting burned*

---

**Davide Mancuso**

automateandgraduate

# Contents

## PART I · THE PROMISE AND THE TRAP

- 1 The Report Nobody Opened
- 2 Your New Pair Programmer Has Never Seen Your System

## PART II · BEFORE A LINE OF CODE

- 3 Read the Old Code First
- 4 The Mockup That Almost Became a Contract
- 5 Specs That Survive Contact with Reality
- 6 Pre-Flight: Ten Checks Before You Write Code

## PART III · BUILDING WITH AN AI PAIR

- 7 Backend: The Names That Don't Exist
- 8 UI5 for an ABAP Developer
- 9 Proving It Works

## PART IV · SHIPPING

- 10 The Things Nobody Transports
- 11 Training in a Week
- 12 Getting IT to Say Yes
- 13 What Changed, and What Didn't

## APPENDICES

- A The Pre-Flight Checklist
- B Verification Rules for AI-Generated SAP Code
- C Prompt Patterns That Worked
- D Building the AI Connector to ADT
- E The Templates
- F Glossary

SAMPLE EDITION · CHAPTER 1 ONLY

# 01

## The Report Nobody Opened



**E**LENA APOLOGIZED FOR THE SPREADSHEET BEFORE SHE EVEN SHARED HER screen.

"It's a bit of a mess," she said, which turned out to be generous. The file had four hundred rows and a color scheme that only made sense if you had invented it. Orange for requested. Yellow if a depot had answered the email. Green once the racks were on a truck. Red if someone was going to get a phone call that night. There was a second tab where promises from the depots were copied in by hand, and a third one called DO NOT DELETE that nobody had opened in months, because nobody remembered what it did.

Elena ran regional supply chain at Norbrook Industrial. Norbrook makes architectural glass, the big panels you see on office

towers and airport terminals, and glass that size does not travel in boxes. It travels upright, strapped to steel racks shaped like an A. The racks are expensive and heavy, and Norbrook reuses them. A rack leaves a plant loaded, gets unloaded on a construction site, comes back empty to one of two depots, North Yard or Lakeside, and sits there until a plant asks for it again.

When a plant has glass ready and no racks, the glass waits. So does the customer's crane, which somebody has booked for Thursday and will be billed for whether or not anything arrives.

"The plants email the depots," Elena said. "The depots email back. And Mark keeps this thing going."

I asked if there wasn't a transaction for it. I already suspected the answer, because I had heard the same pause on other calls with other clients.

"There is," she said. "We don't use it."

## — ZRACKPLAN

The transaction was called ZRACKPLAN. Someone in IT had written it years earlier, and on paper it covered everything the spreadsheet did: rack demand per plant and week, what had been requested, what the depots had confirmed, what had shipped. Every line even had a status light.

I opened it the same afternoon. It ran in SAP GUI inside a browser tab, a dense gray grid that took long enough to load that I had time to refill my coffee. I picked the biggest plant and one of the busiest rack types and looked at the current week.

Eight racks requested. One shipped. Green light.

I checked the code that evening. The light turned green as soon as the first shipment was posted, no matter how many were still

missing. Somebody had probably written it that way on purpose, for a process that worked differently back then, and nobody had touched it since. But the effect was that a planner looking at that screen on a Tuesday would see a crane booked for Thursday, seven racks still sitting at a depot, and a reassuring green dot.

A few days later I asked Mark Delaney, the logistics analyst who kept the spreadsheet alive, whether anyone had ever reported it.

He shrugged. "Green means somebody shipped something. We stopped looking at it a long time ago."

Nobody had filed a ticket. That's the part that stayed with me. The program didn't crash and didn't throw errors, so as far as IT was concerned it worked. The planners had simply moved on and rebuilt the process in Outlook and Excel, and Mark spent a good part of every week phoning depots and recoloring rows so the plants could ship.

I asked him if I could watch him do it once, and the following Wednesday he shared his screen for most of the afternoon.

It started with his inbox. Each plant sent its rack requests in its own format: one plant pasted a small table into the email body, another attached a spreadsheet, a third simply wrote "need 6 of the tall ones for Thursday, same as last week." Mark read each one and typed the numbers into the big file. Then he wrote to North Yard and Lakeside asking what they could send. North Yard usually answered within the hour. Lakeside answered when someone there had time, which on a busy day meant around six in the evening, and if the answer hadn't arrived by then Mark picked up the phone.

Around four o'clock he opened the DO NOT DELETE tab to show me what it did. It held a block of formulas that converted the dates in the depots' replies, because one depot wrote dates month first

and the other day first, and a mix-up had once sent twelve racks to the wrong week. Somebody had fixed it with a formula three years earlier. That somebody no longer worked at Norbrook.

By the end of the afternoon the file was mostly green and Mark looked tired. "Thursday's fine," he said. "Friday I'm not sure yet."

What struck me wasn't that the process was clumsy. It was that all of it, the requests, the confirmations, the shipments, already existed somewhere in SAP or could easily have existed there. The data wasn't missing. People had just stopped believing the screen that showed it.

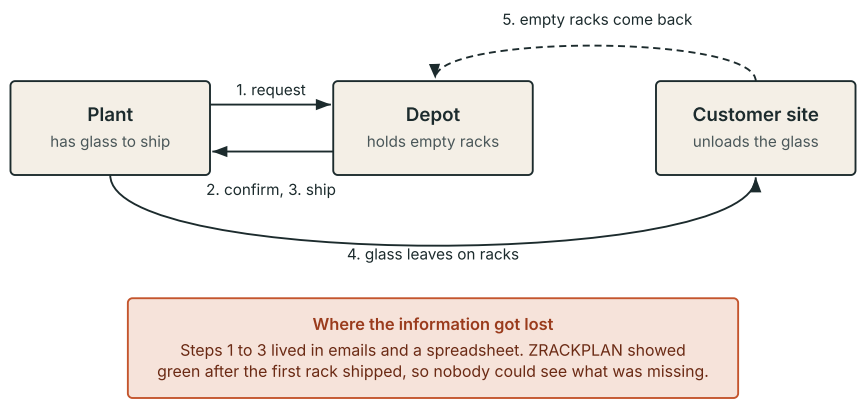


**Figure 1.1** The rack loop at Norbrook. No racks, no shipment.

I've seen some version of ZRACKPLAN at almost every client I've worked for. A report written for a manager who left years ago. A "temporary" cockpit from the last upgrade. An ALV list so slow that everyone exports it on Monday morning and works from the Excel copy for the rest of the week. They usually started out reflecting how the business worked. Then the business changed and the code didn't, or it changed in a rush under somebody else's change request, and the documentation, if there ever was any, stayed behind.

The users don't complain. They work around it. That's why these programs survive: from IT's side nothing looks wrong, and the cost lands on one person in the business who quietly does by hand what the system was supposed to do. In this case it was Mark. In your company it probably has a different name.

What Elena wanted sounded simple. Two apps people would actually open, one for the plants and one for the depots, with a status that told the truth. Fewer emails. No more Wednesday evening calls to Lakeside.



## — THE ESTIMATE

I should say something about myself here, because it matters for the rest of the story.

I've written ABAP for most of my working life. Reports, enhancements, interfaces, the occasional heroic debugging session at two in the morning before a go-live. I can read an old report the way other people skim the news. What I had never done was build a

UI5 application. Not even a tutorial. Fiori was something I configured around and talked about in meetings, but the JavaScript side, the views and controllers and manifest files, was somebody else's job.

So when I sat down to estimate the work, I estimated it the way I would have five years ago. Read and document both legacy programs. Design tables, services and business logic. Build two apps while learning the framework. Test against the old numbers. Training, cutover, and a generous buffer for business decisions, because the business always needs longer to decide than IT needs to build.

I came out at close to three hundred person-days. Most of a year, alone.

I didn't think it was pessimistic. It's also the kind of number that kills a project like this before it starts. Try telling a budget committee you need a year to replace a transaction that technically works. Most ZRACKPLANS are still in production for exactly that reason.

The project ended up taking about a month. I built it and deployed it on my own.

I want to be careful with that number, because it's the kind of claim that gets repeated without its context. It doesn't mean the assistant wrote the apps while I watched. I made every design decision, read every line before it went anywhere near a transport, and spent plenty of evenings fixing things the assistant had gotten wrong with great confidence. It also doesn't mean the business moved faster. Decisions took as long as decisions always take. What changed was the work in between: reading, drafting, scaffolding, comparing, documenting. Those parts went from weeks to days, and on a project like this they are most of the calendar.

## — WHAT MADE THE DIFFERENCE

I wasn't faster. I had help.

By then I'd been using an AI assistant at work for a while, mostly for small things at first: explaining an obscure function module, drafting a test class, turning my notes into something a manager could read. For this project I went further and connected it to the development system through ADT, the same interface Eclipse uses. That meant it could read the actual source code, the real table definitions, the dictionary objects as they existed in that system, instead of my summary of them.

The first week was strange, in a good way.

On day one we had a mockup of the Plant View. It was just a single web page dressed up to look like Fiori, nothing deployed anywhere, but it had the same filters as ZRACKPLAN, rack codes that looked real, and a progress indicator on every line. I could show it to Elena and Mark that same week. Normally they would have received a specification they would never finish reading.

The review was short. Elena liked the progress indicator and asked twice whether the numbers were real (they weren't; I'd written that in red across the top of the page, and she asked anyway). Mark said very little. That evening he sent me an Excel file with a design of his own that he had already sketched, which turned out to matter a great deal. That story belongs to Chapter 4.

On day two the assistant read both legacy programs, every routine and every select, and wrote a technical reference that would have taken me most of a week. And while we went through the code together we found problems nobody had mentioned. In one branch, demand was counted twice. Requests from past weeks were being overwritten. Racks with zero stock dropped off the list

entirely, so a planner couldn't even see they were missing. We checked each of these against live data the same day.

The zero-stock one is a good example of how this worked in practice. Reading the code, the assistant pointed out that the program built its list starting from the stock table, so a rack type with no stock at a depot would never get a row. That was a hypothesis, nothing more. I picked a rack type that I knew was in short supply, ran the old transaction, and the line simply wasn't there. Then I looked at the plant's requests for the same week: they had asked for four. A planner looking at ZRACKPLAN would see nothing at all for that rack, not even a red light. The racks the plant needed most were the ones the screen hid.

Ten minutes from suspicion to proof. That rhythm, where the assistant reads and suspects and I check against real data before believing anything, became the way I worked for the rest of the project.

It also changed the mood of the project. When I showed Elena her own numbers and explained why the old screen had been misleading her team, the conversation stopped being about whether to spend money on a replacement and started being about how soon.

## — CLIENT 100

The same week, I asked the assistant for help with an error.

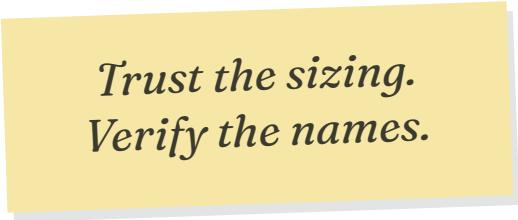
I needed to publish an OData service, the interface the new apps would use to talk to the backend, and the development system refused. The message said publishing wasn't allowed in a customizing client. The assistant answered straight away, clearly and politely: log on to client 100 and publish from there.

I almost did it. I had the logon screen open and was typing the client number when something felt off, and I checked the client list instead. The system had exactly two clients, the technical one that every system has and the development client I was already in. No 100.

There never had been. The assistant had seen client 100 in thousands of other systems, in blog posts and documentation and forum threads, and handed it to me as if it had looked it up in mine.

The actual fix turned out to be a different publishing tool that skips the client check entirely, and it took a while longer to find. The detour itself cost only ten minutes. But it wasn't a one-off, so I started keeping notes. A few weeks later I compared an analysis document the assistant had written from the legacy code against the actual objects. The output table name was wrong. So was the field holding the row type, three routine names, and the row type values themselves. Every category of name in the document was wrong. And yet the document was right about almost everything else: how many routines there were, which called which, roughly how big each one was, what each part did.

I wrote six words on a sticky note and stuck it to my monitor.



*Trust the sizing.  
Verify the names.*

The assistant was very good at the shape of things. It knew how an ALV report is usually put together, how a RAP service is usually layered, how a Fiori list usually behaves. What it couldn't know were the specific names, numbers and settings of a system it had

never seen, so it filled those gaps with whatever was most likely, in exactly the same confident tone as everything else.

Most of this book is about living with that. The assistant saved me months. It also cost me hours whenever I forgot that note was there.

---

## FIELD NOTE

When the assistant gives you something that belongs to your system, a table name, a client, a role, a transport, ask whether it could actually have read it. If it couldn't, it guessed. Check before you act on it. Almost every hour I wasted on this project came from skipping that question.

---

## — ABOUT THIS BOOK

I wrote it for people who work in SAP and are wondering what these tools are really like on a real project. Developers who've asked an assistant for ABAP and got something that looked perfect and didn't activate. Functional consultants and project leads who want to know whether AI changes what they can promise a client. It does, though not where most people expect. And IT managers who have to decide whether their teams should be allowed to use any of this.

You don't need to know UI5. I didn't. You don't need to understand how language models work, either. You need some SAP experience and the patience to check things.

You also don't need to read it front to back, although that's how the story works best. If you already use an assistant every day and want the practical parts, start with Part II. If you're the person who has to approve or refuse these tools for a team, Chapters 6 and 12 and Appendix D are written with you in mind. And if you just want to connect an assistant to your own development system this weekend, Appendix D walks through it step by step.

The book follows the project in order. Part I is about how the assistant behaves once it's connected to a real system. Part II covers everything before the build: reading the old code, the mockup, the specification, and the checks I now run before writing anything. That's where the assistant helped most and where some of its mistakes were most expensive. Part III is the build itself, backend and frontend, and how I proved the new numbers matched the old ones, or didn't. Part IV covers what nobody demos: transports, the steps no transport carries, training material, and getting IT to approve the whole thing. The checklists and verification rules are collected in the appendices, so you can use them without rereading the book.

A word about the story. The project happened and the lessons are real. Everything else has been changed: the company, the industry, the people, the systems, the numbers, the words people said. Norbrook doesn't make glass, and there is no Elena or Mark, although every project I've worked on had someone just like them.

Early on, before there was any code, I asked Mark what a good result would look like for him. I expected something about the apps or the status lights.

"I'd like to go home on a Wednesday without calling a depot," he said.

Keep that in mind as you read. The speed is nice. What counts is whether the people at the other end can stop working around the system.