# What Did AI Actually Change in My Business?

## And how would I know? A short guide for owners and operators

*By Michael Coon*

## The question I couldn't answer

Your people are probably using AI already. Maybe you've paid for a tool. Maybe someone on your team built something and showed you a demo. It looked good, and it may even be running now. But if someone asked what it changed in the business, you might have a hard time answering.

I know that spot, because I've been in it.

A while ago I was asked which of my projects had made the biggest difference to a client, and how I'd measured it.

I've been building software for almost thirty years. I've led teams, designed large systems, and started three companies. I could not answer the question. I could describe what I'd built in enormous detail. I could not tell you, for most of it, what happened afterward.

That bothered me enough to go back through projects that mattered to me and check. Of the projects I examined, I could show that roughly a quarter had clearly changed something for the business that paid for them. For about a third, I couldn't tell; the information existed and I'd never seen it. Nearly half hadn't changed anything, and I could now say why.

I wrote a book about what I found. It's for engineers. This is not that book. This is the short version for the person who approves the money, because the question I couldn't answer is one you're about to be asked, about AI, by your own board, your partners, or yourself.

What did AI actually change in my business? And how would I know?

## Building got cheap. What didn't.

Here is what has changed in my work over the past couple of years, stated as plainly as I can.

Someone at your company can now describe a problem to an AI tool and get working software back in days. Not a mockup. Working. I do this routinely, and it's likely someone on your staff has tried it too, with or without telling you. The thing I spent most of my career learning to do, turning a request into a functioning system, has become the cheap part.

So "can we build it?" is no longer the hard question. For a first version, the answer is usually yes.

What remains difficult:

Deciding what's worth building at all.

Knowing, afterward, whether it changed anything.

Keeping it working once the person who built it has moved on.

Having someone answerable when it's wrong.

And here's the part I'd most like you to hear, because I learned it by getting it wrong: faster building makes those harder, not easier. When software took six months, you had six months to notice you were building the wrong thing. Now you can build the wrong thing in thirty days and barely notice the waste, because thirty days doesn't feel like enough time to have wasted anything.

I'd like to give you the questions I now ask before anything gets built, and after. There are six. They're not technical. They're the questions I should have been asking for twenty-five years and mostly didn't, because somebody else in the building was supposed to.

# Six questions

These are the questions, in order. I'll give you each one in plain language and then show you all six on one real project.

## 1. What should be different?

Not what should exist. What should be different in the business if this works.

"We'll have an AI assistant that answers customer questions" is a description of a thing. "Customers get a correct answer almost immediately, without pulling in a senior person" is a change in the business. The second one could come true without any software at all, which is how you know it's the real goal.

Write the goal as one sentence with no product in it. If you can't, you don't have a goal yet. You have a request. That's fine, but know which one you're holding.

Then ask why that matters, twice. The first answer is usually the thing in front of you. The second is usually the thing the business actually cares about.

## 2. What has to be true for that to happen?

Every plan takes things for granted. "We believe an AI assistant will cut the time it takes to answer customer questions." Underneath that sentence sit the things that have to be true for it

to work: that the answers are written down somewhere it can read, that they're current, that customers will accept a reply from it, that your staff will trust it instead of checking every answer, that the vendor won't change the price, that the person who set it up stays. Each one is a bet. The plan rarely lists them, because the people making it already believe them.

So list them. Everything you can think of that would have to hold for this to work. Most will look obvious once written down, and that's fine; obvious bets are cheap to carry. You're looking for the ones that make you pause.

In my experience the dangerous ones tend to hide in three places, so check those on purpose:

Who, exactly, is this for? Not "the sales team." Which people, doing what, with what tolerance for a new way of working.

How does it actually reach them? What sits between "it works" and "they use it every day"?

What has to stay the same? Would a vendor's next release, a change in your market, or one person leaving make this pointless before it pays off?

Then rank the whole list by two things: how badly it hurts if this one is wrong, and how little you actually know about it. The one at the top is where the next question goes.

## 3. What's the cheapest way to find out?

Before you build the thing, test the bet at the top of the list. Sometimes the cheapest test is three phone calls. Sometimes it's a spreadsheet. Sometimes it's a small, disposable piece of software whose only job is to answer one question. The point is that it's cheap and it's built to be thrown away.

Write three things down before the test starts:

What question does this answer?

What result would make us stop? Decide this before you see the result, because afterward you will talk yourself into continuing.

What do we do with each answer?

And allow for the answer to be "build nothing." That's a legitimate outcome of a good test, and it's the one engineers are worst at recommending, because we're paid to build. If you're the one approving the money, you have to be the one who makes "nothing" an acceptable answer out loud, or you'll never hear it.

## 4. What actually happened?

After it's live, three layers, in order:

Did it reach the people it was for? Not "was it rolled out." Are they using it.

Did their behavior change? Are they doing something differently than before.

Did the business condition move? The sentence from question one. Is it truer now.

Each layer needs a "before" measurement, taken before you start: how long the task took, how many got done, how much it cost. Without it, everything you measure afterward is a number with nothing to compare against. This is a week of work at most and it's the week most often skipped, because it produces nothing anyone can demo.

And watch for the most comfortable sentence in any status report: "it works." It's almost always true. It's about the software, not the business, and it answers none of the three layers.

## 5. What did the business get?

The people using the thing and the people paying for it are rarely the same, and they measure in different units. A tool the staff loves can lose money. A test that produced no software can save a year. The question for you is what the business got back, in your units. I'll come to those below.

## 6. What do we decide next?

Four possibilities, and only three ever get discussed.

Continue: the evidence supports it, do more.

Adjust: the evidence says the problem is somewhere else, move the effort there.

Stop.

And the fourth, which is what happens when nobody chooses: drift. The project continues because it was continuing. The budget renews because it renewed last year.

Most of the projects I examined ended in drift. The cure is cheap and uncomfortable: at the review, name the default out loud. "If we do nothing, we keep paying for this and keep not looking at it." Once drift has a sentence, somebody has to choose it on purpose, and almost nobody will.

Write down what was decided, why, and when it gets looked at again.

# How it works in real life

The CEO of an industrial equipment manufacturer asked me for an AI system. His company had thirty years of sales records; he wanted software that could read them and draft proposals for new customers.

What should be different? When I asked, the answer was cash. Proposals built around standard configurations would let him stock standard builds instead of custom ones, and standard builds tie up less money in inventory. That sentence has no software in it. That was the real goal.

What has to be true? Two things, before his money went into a build: that thirty years of records could actually be read by a machine, and that the system could write a proposal as good as a person's. If either was false, the inventory goal would never move.

The cheapest way to find out: one week, on the first one.

It took a day to learn the records weren't data at all. They were scanned images of paper. I spent the rest of the week getting a machine to read them anyway, and by Friday it could. But the week also surfaced the real finding: years earlier he'd had a digital sales process built to capture this information cleanly, and his people had abandoned it because it was too hard to use. The data the whole idea depended on didn't exist in usable form, and no software was going to create it.

What actually happened: a test that worked, and showed the problem sat upstream of anything I could build.

What the business got: no software, a week of my time, and a decision made with the evidence in front of him instead of after a six-month build. He chose to fix the sales process first and come back to software when there was something for it to recommend.

I'd been asked for an AI system. The right answer was a week of looking, followed by nothing. It's one of the few projects I can say clearly mattered.

# The four things a business measures

When I say "in your units," I mean four things. Everything the people using a tool experience, time saved, frustration avoided, eventually has to be translated into one of these before it means anything to the business.

Money. Revenue, cost, cash. The one engineers are most reluctant to ask about, as if it were impolite. The manufacturer's goal was cash in inventory. Yours is probably something as concrete.

Risk. Exposure you've taken on without noticing: a system nobody but its builder understands, a tool that touches customer data, a dependence on one vendor or one person. Risk is a cost that only shows up when it's realized, which is why it gets added freely.

Capability. What the company can now do that it couldn't before. This one is undercounted, because it doesn't show up in a quarter.

Trust. With customers, with staff, with partners, with whoever approves the next budget. Trust is spent and earned by every one of these projects, and it rarely appears on a report.

Effects and consequences can point in opposite directions. I co-founded a company whose product its users loved; it executed more than half a billion dollars in trades. Then we were hacked, and the engineering was mine. Even then, most of the users who wrote to us were asking when it would be back. The company still closed, because our funding and fees had never been designed to absorb a loss like that, and nobody, including me, had been watching

those numbers. The users got what they wanted. The business didn't survive it. If you only look at whether people like the thing, you will mis-score it.

## Who keeps it running, and who answers for it

A tool that was cheap to build isn't cheap to own. Once it's part of how work gets done, two questions stay open for as long as it runs, and they're the two least likely to be on anyone's list.

Who keeps it running? The person who set it up is often the only one who understands it, and that person changes jobs, gets busy, or was a contractor to begin with. I once built a system that made a team's work much faster, and most of them wouldn't rely on it as it stood, because when something went wrong they couldn't see where to look. If one person leaving would leave you with a tool nobody can fix, that belongs on your list of bets from question two.

Who answers when it's wrong? Software doesn't take responsibility for its mistakes. When a wrong number reaches a customer, someone at your company has to notice it, explain it and fix it. Decide who that is before it happens, and ask how they'd find out. A tool that fails loudly is an annoyance. A tool that's quietly wrong for three months is a risk you've been carrying without knowing it.

AI tools can help with a lot of this. They can't be the person who answers to you.

## A twenty-minute exercise

Pick one AI effort at your company. Something bought, built, or being tried right now. One that's already cost money, if you have one.

Fill this in. Short answers are fine. Blanks are fine too; the blanks are the point.

What should be different in the business if this works? One sentence. No product in it.

Why does that matter? And why does that matter?

Who, exactly, is it for, and are they using it today?

What are we assuming has to be true? List three. Circle the one you know least about.

What did we measure before it started (how long the task took, how many, how much), and what does it read now?

What has the business gotten so far, in money, risk, capability, or trust?

What result would make us stop? Has anyone written that down?

If the person who built or set this up left tomorrow, who keeps it running?

When it's wrong, who answers for it, and how would you find out?

If we do nothing, what happens? (Say it out loud.)

When this is next reviewed, and by whom?

Then look at your blanks. For each one, write the name of the one person at your company who could fill it in, or "nobody," which is its own answer.

That page is more than I had for most of my projects over twenty-five years. If you've filled it in once, you know more about that effort than most people know about theirs.

# Who I am, and an invitation

I'm Michael Coon. I've spent almost three decades building software, led engineering at companies large and small, and co-founded three companies. I now run a small consultancy where I work with teams that now build with AI or use AI as part of their staff, and I spend most of my time on the questions above rather than on the building.

If you try the exercise, I'd like to know what you found. Especially the blanks, and especially if the answer to "what should be different" turned out to be harder than you expected. Reply and tell me. Sometimes one of these conversations turns into work for my consultancy. Usually it doesn't, and that's fine. I'm collecting what owners actually run into, and the honest ones are the useful ones.

Michael Coon, Thorium Development Group, thoriumdev.com, mike@thoriumdev.com