RESEARCH

You Can't Buy Capability

Buying a build makes you dependent. Building capability makes you free.

By **Alan Babbitt** · Intelligent Systems Architect · July 2026 · 5-minute read

The premise

Almost every request arrives in the same shape: *can you build me something that does X.*

Sometimes that's exactly right, and the honest answer is a quote and a delivery date. But a good share of the time, the thing being described isn't missing software. It's a missing ability. The person doesn't need a tool built. They need to be able to do the thing.

Those two look identical on the way in. They are not remotely the same on the way out.

Buy a build and you own an asset that works until it doesn't, at which point you go back to whoever made it. Build capability and you own the ability to make the next one yourself. One of those compounds. The other one depreciates.

> *A build answers one question. Capability answers the questions you haven't thought of yet.*

The tell

There's a simple test for which one you're actually looking at.

Imagine the person who set it up is gone. Not dramatically — they just moved on. Now: does the thing still work? Can someone in your shop change it? If a client asks for something slightly different next month, does that require a phone call to an outsider?

If the answer is that everything holds until something needs to change, and then it stops — you bought a build. That's not a mistake. It's just important to know, because the bill for that arrives later, quietly, as a dependency you didn't price.

The uncomfortable version of the same test: if the tool vanished tomorrow, would your people be better at their jobs than they were a year ago? If not, nothing was actually transferred.

What capability is actually made of

"Get better at AI" isn't a plan, because it isn't one thing. It's five, and they fail independently. Someone can be strong in three and stuck in two, and the two are what's holding everything up.

THE FIVE AXES OF WORKING CAPABILITY

1. **Fluency.** How well you can actually talk to it. Most people are running every session cold, re-explaining their business from scratch, and judging the tool by answers it had no chance of getting right. Fluency is usually the single biggest jump available to anyone, and it's the cheapest.
2. **Reach.** How much of your real work you've pointed it at. You can't apply it to work you've never named. Most people have aimed it at two or three obvious things and left the rest of the week untouched, mostly because nobody ever sat down and listed what repeats.
3. **Repeatability.** Whether the good result happens again. A prompt you retype is a task. A prompt you save is a tool. The difference between someone who is impressive with AI and someone whose firm benefits from AI is almost entirely this.
4. **Integration.** Whether the work moves on its own, or moves because you carried it. Most operations have real seams — a place where a person copies something out of one system and pastes it into another. Those seams are where time goes and where errors get born.
5. **Judgment.** Knowing where it must not run unsupervised, and designing that in on purpose. Not a nervous feeling about AI. An actual decision about which cases route back to a person, made before something goes wrong rather than after.

Most self-assessment gets this wrong in a specific way: people rate themselves on Fluency, because that's the axis you can feel. Reach and Repeatability are the ones that quietly decide whether any of it shows up in the business.

Why a build doesn't create capability

This isn't a knock on building things. We build things. The point is narrower: a delivered system transfers a ***result***, and capability is a different cargo that has to be loaded deliberately.

When someone else does the work, the useful part — the hundred small judgment calls about what to point it at, what to trust, where to stop — happens in their head and stays there. You get the output. You don't get the reasoning that produced it. That's why the second request is just as dependent as the first, and the fifth one too.

There's a second reason, less obvious. A system built for how your work looked in March starts drifting the moment your work changes. If nobody inside can adjust it, the drift just accumulates until the whole thing gets quietly abandoned and replaced with the old manual way. That happens more often than anyone admits, because nobody writes a memo announcing they stopped using the thing.

When a build is genuinely the right answer

Three cases, plainly:

- **You can describe it to a supplier.** A website, a logo, a set of social posts, a brochure. Defined work with a defined finish. Paying advisory rates to think about it is a waste of your money.
- **It's genuinely one-and-done.** Some things don't need to change and don't need to be extended. Build it, own it, move on.
- **The skill has no use beyond this task.** Not everything is worth learning. There's no prize for doing your own plumbing.

Outside those, the build tends to be the expensive way to solve the cheap half of the problem.

Bottom line

The question worth asking before any AI spend isn't "what should we build?" It's *what do we want to be able to do a year from now that we can't do today* — and then working backwards.

Sometimes that leads to a build, and you should buy one without guilt. Often it leads somewhere else: five or six people who can each do a thing they couldn't do before, in a business that no longer stops when one person is out.

Buy the build when the job is defined. Build the capability when the job keeps changing. Most jobs keep changing.

The five axes above are what our capability check measures — where you're strong, where you're stuck, and which one to fix first. It takes about two minutes at **isaadvisory.com/training →**