

BEYOND THE DEMO

The Capability Cliff: Why Organisations That Automate Before They Understand Pay for It Later

On the difference between a process that runs itself and an organisation that no longer understands its own work

Giovanni Leonardi

January 2025



The Capability Cliff: Why Organisations That Automate Before They Understand Pay for It Later

Transformation · No. 400

Originally written January 2025 | Re-edited and re-rendered for the WhereBeyond Edition, 2025

“Automation does not fail at the moment you switch it on; it fails quietly, months later, in the exceptions you stopped being able to see.”

1. The morning the exceptions disappeared

Every automation programme has a moment its sponsors remember fondly. It arrives about six weeks after go-live. The dashboard that used to show a queue of flagged cases now shows almost nothing. Straight-through processing is running above ninety per cent. The team that once numbered thirty is down to a handful. Someone in the steering meeting says the word transformational, and for once nobody flinches.

It is a good moment. It is also, very often, the moment the ground begins to move.

The pattern that recurs across these programmes is not that the automation was badly built. The models were adequate; the integration held; the pilot, if anything, demonstrated too well. The pattern is that the organisation automated a process it had never actually understood — and, in doing so, quietly dismantled the one mechanism it had for understanding it. That is the capability cliff. You do not fall off it on go-live day. You walk confidently along the top for a quarter or two, and then discover there was never any ground beyond the edge.

2. Comprehension is the thing being spent

We are living through a period in which the pressure to automate has decoupled almost entirely from the question of whether we understand what we are automating. The board has asked for a number — how many processes, by when — and the number is being met. The tooling has finally caught up with the ambition: where an earlier generation of automation could only follow rules a human had written down, the current generation reads, drafts, classifies, and decides, and does so in fluent, confident prose. The demonstrations are genuinely impressive. That is precisely the problem.

Because fluency is not comprehension, and the two are easily confused — by the system, and by the people watching it. A process that produces plausible output at speed looks,

from the steering committee, exactly like a process that is working. The difference only surfaces at the edges: the case that does not fit the pattern, the input that is subtly malformed, the upstream change nobody flagged. In a well-run manual process, those edges were where the organisation's real knowledge lived. The experienced hand who said that doesn't look right was not following a rule. They were exercising an understanding built up over years of seeing the work go wrong in small ways and correcting it.

Automate the ninety per cent that is routine, redeploy the people who handled it, and you have not simply made the process cheaper. You have removed the population in which the understanding was resident, and you have severed the feedback loop through which it was renewed.

The cost of automating before you understand is not paid at go-live. It is paid later, in the exceptions you can no longer see and the judgement you no longer have anyone to exercise.

3. Why sensible organisations do it anyway

It would be comforting to put this down to carelessness, but the organisations that walk off the cliff are usually the diligent ones. Several forces push in the same direction, and they are all rational in isolation.

- The mandate is quantitative. When the target is number of processes automated this year, the incentive is to pick processes that can be automated quickly — which tends to mean processes nobody has taken the trouble to fully map, because mapping them would slow the count.
- The demo sets the expectation. A capable system handling a curated set of cases in a sponsor's meeting establishes, in everyone's mind, that the work is essentially solved. The curated cases are the ones the builders already understood. The uncurated ones are the whole risk, and they are not in the room.
- Understanding is invisible on a business case. The line item for the six experienced people who catch the exceptions reads as cost. The tacit knowledge they carry does not appear as an asset anywhere, so removing them looks like pure saving. You cannot defend a number that no ledger records.
- Fluent output disarms scrutiny. When a system explains its own reasoning in confident, well-formed language, the natural human response is to trust it more, not less — even though the fluency and the correctness are produced by different mechanisms and can come apart completely.

None of these is a mistake in the ordinary sense. Each is a reasonable response to a real pressure. Together they produce an organisation that is measuring its throughput closely and its comprehension not at all.

4. A composite from the exceptions desk

Consider a shared-services function processing supplier invoices — forty thousand a month, drawn from several thousand vendors on inconsistent terms. Before automation, thirty people ran it. About ninety-two per cent of invoices went through cleanly; the remaining eight per cent — mismatched purchase orders, split deliveries, currency edge cases, the vendor who invoices in a format no one else uses — landed on an exceptions desk staffed by the most experienced people in the team. That eight per cent was not the residue of the work. It was the work. It was where the team's understanding of how the business actually bought things was stored and applied.

The automation targeted the ninety-two per cent, and hit it. Straight-through processing rose; the team was cut from thirty to six; the business case was, on its own terms, a triumph. Then, over the following two quarters, three things happened that no dashboard had been built to show.

1. The exception rate drifted upward — from eight per cent toward thirteen — because an upstream change in how purchase orders were raised began producing mismatches, and no one downstream was close enough to the work to notice the source.
2. The six remaining staff, who had inherited a cleaned-up queue rather than grown up in the messy one, could clear exceptions but could not diagnose them. They could tell you that an invoice had failed, not why the failures were multiplying.
3. The automated path, faced with the newly ambiguous cases, did not stop. It resolved them — confidently, and increasingly wrongly — because a system optimised to keep the throughput number high has no incentive to hesitate.

The organisation had automation and a growing, invisible error rate and no one left who could explain it. That is the cliff. Not a dramatic failure at launch, but a slow erosion of a capability that had been doing quiet, essential work, discovered only when someone finally reconciled the numbers.

5. The strongest objection — and why it does not save us here

The serious counter-argument deserves stating at full strength, because it is usually right. Every wave of automation has provoked exactly this anxiety, and every previous time the anxiety was overblown. We automated payroll, and no one mourns the ledger clerks' lost understanding of tax tables. We automated manufacturing, then typesetting, then vast swathes of what a spreadsheet replaced. Progress is abstraction: we build on layers we no longer personally understand, and that is a feature, not a failure. Most of us could not explain how the compiler beneath our code actually works, and we are right not to care. On this view, "understand before you automate" is just the latest costume for a very old reflex, and comprehension will follow automation as it always has.

That argument is correct about the last century and wrong about this decade's target. The disanalogy is specific.

Earlier automation	This wave
Encoded stable, well-understood, largely deterministic processes	Points at judgement-laden, ambiguous, non-stationary work
Failed legibly — it stopped, threw an error, produced an obvious blank	Fails fluently — it continues and produces confident, plausible, wrong output
Sat on a layer beneath that stayed still	Sits on a layer — data, vendors, regulation, behaviour — that keeps moving
Comprehension could safely follow, because the ground did not shift	Comprehension must lead, because there is no stable floor to fall back on

Abstraction is safe when the layer beneath it is stable and its failure modes are legible. You can forget how the compiler works because it does not silently change its mind, and when it breaks it breaks loudly. The capability cliff appears exactly where neither condition holds: where the underlying process keeps moving, and where the automation fails by carrying on rather than by stopping. The old objection is a good description of automating the known. It is a dangerous one when smuggled into the automation of the not-yet-understood.

6. Comprehension first, and kept

The conclusion is not a counsel of despair, and it is certainly not do not automate. It is that the sequence matters, and that we have been running it backwards.

“Understand the work, then automate the part you understand, and keep the people whose job is to understand the rest.”

In practice that means treating comprehension as a deliverable in its own right, not a by-product. Map the process — genuinely, including the exceptions — before automating it, on the principle that if you cannot describe how the work fails, you are not ready to hand it to a machine. Automate the stable, understood core, and instrument the frontier: make the exception rate a headline metric, watched as closely as throughput, because it is the vital sign that tells you whether the ground is still where you left it. Keep experienced people on that frontier — not as a fallback rota, but as the organisation's reservoir of understanding, and as the apprenticeship through which the next generation acquires it. And when the numbers look transformational at go-live, treat that as the moment to raise your scrutiny, not to stand it down.

Automation does not fail at the moment you switch it on; it fails quietly, months later, in the exceptions you stopped being able to see. The organisations that will do well in this period are not the ones that automate the most, or the fastest. They are the ones that never let go of understanding their own work — and that recognise the seductive quarter

after go-live, when the exceptions seem to have disappeared, for what it usually is: not the summit of the transformation, but the edge of the cliff.