

BEYOND THE MAGIC WORDS

Prompt Engineering Was Requirements Engineering All Along

How a year of stalled enterprise AI pilots exposed an atrophied discipline,
not a new one

Giovanni Leonardi

March 2024



Prompt Engineering Was Requirements Engineering All Along

Transformation · No. 309

Originally written March 2024 | Re-edited and re-rendered for the WhereBeyond Edition, 2025

“A capable model will not save you from not knowing what you want; it will simply execute your confusion faster and more fluently.”

1. The Demo That Dazzled and the Pilot That Died

Every organisation now has a version of the same story, and most have not yet noticed that it is the same story.

A team is asked to show what the new language models can do. They pick something visible and painful — first-draft replies to customer complaints, say, in a business that receives thousands a week. Within a fortnight they have a demonstration. Someone pastes a real complaint into the model, adds a few lines of instruction, and out comes a courteous, competent draft. The room is genuinely impressed; the drafts are better than some of what goes out today. A budget is found, the pilot is greenlit, and then, over the following months, the thing quietly fails to become real. The drafts that were 70 per cent acceptable in the demo turn out to be 30 per cent acceptable across the messy diversity of live cases. The complaints team stops trusting it. The pilot is declared promising and shelved, and everyone agrees the technology is not quite ready.

The technology was ready. The requirements were not.

I have watched this pattern repeat across sectors over the past year, and the striking thing is how rarely anyone names the actual failure. We reach instead for the vocabulary of the moment: the model needs better prompting. We hire for it, run internal workshops on it, circulate lists of incantations. A whole craft has sprung up around the phrasing of instructions, complete with its own folklore about which magic words unlock better output. And in treating this as a new craft, we have almost entirely missed what it is — the return, under an unfamiliar name, of a discipline most organisations spent the last two decades quietly dismantling.

2. The Same Act, Renamed

Prompt engineering is requirements engineering. Not like it, not adjacent to it, but the same essential act: specifying, precisely and testably, what you actually want a system to do. The surface has changed. You are writing prose to a stochastic model rather than a

specification to a development team. The hard part has not changed at all. And the reason so many pilots stall in exactly the same place is that they run headlong into a problem organisations have been avoiding for years — the problem of saying, exactly, what good means — and mistake it for a problem with the tool.

The observation is decades old now: the hardest single part of building a system is deciding precisely what to build. Language models did not repeal that truth. They removed our last convenient place to hide from it.

3. The Discipline We Unlearned

There was a time when specifying intent was a recognised, staffed, respected job. The business analyst sat between the people who wanted something and the people who would build it, and did the unglamorous work of turning we need to handle complaints better into something a builder could act on: the actual decision rules, the edge cases, the definition of done, the things that must never happen. It was translation, elicitation, and disciplined suspicion — because the one reliable truth of the trade was that stakeholders could not, unaided, tell you what they wanted. The tyre-swing cartoon, passed around every requirements course, made a point that has never stopped being true.

Then, for reasons that were mostly good, we moved on. Agile methods, rightly reacting against the fantasy that you could specify a large system fully in advance, taught teams to discover requirements incrementally, through working software and fast feedback. But somewhere in the translation from principle to practice, we will refine the specification as we learn too often curdled into we will not really specify at all. The dedicated analyst role thinned out. Acceptance criteria narrowed to whatever fit on the back of a user-story card. A generation of technologists grew up fluent in delivery mechanics and much less practised in the older, harder craft of pinning down intent before committing to it. We became very good at building things quickly and noticeably worse at deciding precisely what to build.

For most of that period it did not obviously hurt, because the cost of ambiguity was absorbed by people. A developer handed a vague ticket used judgement, asked a question in the corridor, filled the gap with common sense. The specification was always incomplete; humans quietly completed it. That absorption was invisible, and because it was invisible, we came to believe the specification had been complete all along.

4. Why the Model Sends the Bill

A language model does not absorb ambiguity. Or rather — and this is the crucial part — it appears to, which is far more dangerous than if it plainly refused. Hand it an underspecified instruction and it will not stop to ask what you meant. It will infer: confidently, fluently, and often plausibly, handing you back something that looks finished. Where a junior colleague's uncertainty is visible on their face, the model's is hidden inside a well-formed paragraph. A capable model will not save you from not knowing what you

want; it will simply execute your confusion faster and more fluently.

This is why the demo-to-pilot collapse is so consistent. The demo works because a human hand-picked the input and eyeballed the output — an informal, one-off act of specification performed live in the room. Production fails because nobody ever wrote down what the model was supposed to do across the full range of reality: which complaints require a human, what compensation may never be offered without sign-off, what tone the regulator expects, what a wrong answer actually costs. Those are not prompt-phrasing questions. They are requirements. They were always requirements. The model simply declines to supply them on your behalf.

Put the two crafts side by side and the identity is hard to miss.

The requirements act	Its form in "prompt engineering"
Eliciting what stakeholders actually want	Working out what you are really asking the model to produce
Disambiguating vague intent	Replacing "handle it well" with explicit rules and constraints
Specifying acceptance criteria	Defining what a good output is, testably, before you trust it
Enumerating edge cases and failure modes	Naming the inputs that must escalate, refuse, or defer to a human
Providing worked examples	Few-shot exemplars that show, rather than tell, the target

The right-hand column is what capable teams discovered they had to do. The left-hand column is what their organisations knew how to do twenty years ago and mostly stopped resourcing.

5. The Two Objections Worth Taking Seriously

The argument has two serious counters, and a perspective that ducks them is not worth reading.

The first says prompt engineering is genuinely novel, not old wine in a new bottle. You are addressing a stochastic system whose output shifts with phrasing, tone, ordering, and the particular examples you choose — sensitivities no deterministic specification ever had to model. There is real, new craft in selecting exemplars, controlling output format, coaxing a model to reason step by step rather than blurt an answer. This is true, and I do not want to wave it away. There is a genuinely new surface layer, and the people who have learned it are not fooling themselves. But notice what that layer actually governs: how faithfully the model receives a specification you have already made. It is real, and it is secondary. When a pilot fails, it almost never fails because the exemplars were ordered wrongly; it fails because no one specified what good looked like. The novel craft is real. It is simply not where the value is won or lost.

The second objection is more forceful: even if prompt-craft is really specification, it is ephemeral specification. The models are improving so fast that within a generation or

two they will infer sloppy intent well enough that all this discipline evaporates — so why invest in it? This is the argument that prompt engineering will not exist as a job in a few years, and on its own terms it may even be right about the phrasing tricks. But it has the economics exactly backwards. A more capable model does not reduce the cost of ambiguity; it raises it. A weak model given a vague instruction produces something visibly poor, and you catch it. A strong model given the same vague instruction produces something fluent, confident, and wrong in ways that are expensive to detect. The better the system becomes at executing intent, the more it punishes you for not knowing your own. Improving models retire the magic words. They do not retire the need to know what you want — they make that need more valuable, not less.

*“The better the model becomes at doing what you ask,
the more expensive it becomes not to know what you
are asking for.”*

6. What the Teams That Succeeded Actually Did

The organisations that got somewhere real with these tools over the past year were, with striking regularity, not the ones with the cleverest prompts. They were the ones who treated the problem as specification and staffed it accordingly.

Return to the complaints pilot, and take the version that worked. That team did not start with the model. They started with the complaints. They sat with the handlers and extracted the actual decision rules — the ones that lived in people's heads and had never been written down. They defined, testably, what an acceptable draft was: factually correct, within policy, in the regulated tone, and escalating anything that touched vulnerability or a compensation claim above a threshold. They assembled a deliberately awful set of cases — the ambiguous, the abusive, the genuinely borderline — and made passing those the bar, rather than the flattering demo case. They specified what the model must never do as carefully as what it should. Only then did they write the instruction, which turned out to be the easy part. Their acceptance rate on live cases settled around 85 per cent — not because their prompt was more elegant, but because their requirements were.

- They wrote the acceptance criteria before they wrote the prompt.
- They treated the hard inputs, not the easy demo case, as the specification.
- They named the failure modes explicitly, including the ones the model should refuse outright.
- They put someone who could elicit intent, not merely phrase instructions, at the centre of the work.

None of this is new. All of it would have been recognisable to a competent analyst in 2004. That is precisely the point.

7. The Uncomfortable Conclusion

It is tempting to read all this as good news — the skills are not new, we have done this before, we merely need to remember. The more honest reading is less comfortable. These pilots keep failing in the same place because the discipline they demand is one we actively let wither, and rebuilding it is harder than buying a tool or running a prompting workshop. It means valuing, and paying for, the patient work of specification again, inside a culture that spent two decades treating it as bureaucratic drag.

The enthusiasm is already racing ahead of us — stringing these models together into semi-autonomous agents that take actions rather than merely draft text. If an underspecified instruction is expensive when the output is a paragraph a human reviews, it is a great deal more expensive when the output is an action the system takes on its own. Every step in that direction raises the premium on the same old skill: the ability to say, precisely and testably, what you actually want.

The language models did not hand us a new problem. They handed us a mirror, and reflected in it is a question our field has been avoiding for years — whether we can still say exactly what we mean. Prompt engineering is not the new discipline we should be rushing to invent. It is the old one we can no longer afford to have forgotten.