

BEYOND THE CLEVER PROMPT

Prompt Engineering Is Requirements Engineering in Disguise

How a discipline the software profession spent thirty years learning is being quietly unlearned, at speed, in the name of a new one.

Giovanni Leonardi

May 2024



Contents

1. Executive Summary	3
2. The demo that worked and the system that did not	3
3. The specification problem never left; it moved	4
4. What the old discipline learned, and the new one is unlearning	5
5. Why we cannot see what we are looking at	6
6. The strongest case for genuine novelty – and why it deepens the point	7
7. A worked case, invented but true to life	7
8. The transformation gap, at machine speed	8
9. Recovering the discipline we already have	9

Prompt Engineering Is Requirements Engineering in Disguise

Transformation · No. 379

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

“The gap between the demo and the system turns out to be the whole of the work.”

1. Executive Summary

The profession has persuaded itself that prompt engineering is a new craft. It is not. It is requirements engineering — the old discipline of saying precisely what you want to a literal-minded executor — returned in a new costume, and returned to organisations that never mastered it the first time. The specification problem did not disappear when we started addressing machines in plain English; it simply migrated from the compiler to the model, and lost, in the migration, most of the hard-won apparatus the software profession had built to contain it.

This essay argues three things. First, that the difficulties teams are now discovering in getting reliable behaviour out of large language models are not novel technical problems but the return of ambiguity, unstated assumption, and missing acceptance criteria — the exact failure modes requirements engineering spent thirty years learning to name and manage. Second, that a set of structural forces — vendor framing, professional identity, and the peculiar seduction of natural language — are actively concealing this continuity, encouraging organisations to treat prompting as a fresh skill to be acquired rather than an old discipline to be recovered. Third, that the enthusiasm now gathering around autonomous agents raises the stakes of specification rather than lowering them, because an unclear instruction handed to a system that acts on its own judgement is more dangerous, not less, than one handed to a system that merely drafts.

The uncomfortable implication for anyone leading change is that the organisation which could never write a clear requirement will not suddenly write a clear prompt. The tooling has changed; the discipline it demands has not.

2. The demo that worked and the system that did not

There is a moment that has become familiar in the past eighteen months. A small team gathers around a screen. Someone has spent an afternoon refining an instruction to a language model — coaxing, rephrasing, adding an example or two — and now the model produces something genuinely impressive: a clean summary, a plausible draft contract clause, a correctly structured extract from a messy document. The room reacts the way rooms do to a good demo. A date gets put in a plan. The capability is, everyone agrees,

essentially there.

Then the same instruction meets the real distribution of inputs. The document that is slightly differently laid out. The customer message written in three languages at once. The edge case that the afternoon of refining never happened to hit. And the beautiful behaviour degrades — not with an error message, which a practitioner could at least route around, but with the same confident fluency it showed on the demo, now applied to an answer that is quietly wrong. The gap between the demo and the system turns out to be the whole of the work.

Anyone who lived through the requirements practices of the last three decades will feel a strong sense of recognition here. We have seen this shape before. It is the gap between the requirement that was agreed in the workshop and the requirement as it survived contact with production. The demo is the happy path; the system is the specification of everything the happy path ignored. What the team discovered was not a limitation of the model. It was the absence of a requirement.

3. The specification problem never left; it moved

Strip away the vocabulary and a prompt is a specification. It is a statement, in language, of the behaviour a system should exhibit given some input. That is precisely what a requirement is, and it always was. When we wrote a functional requirement, we were trying to constrain the behaviour of a deterministic machine that would do exactly and only what the code said — no more, no less, and nothing we forgot to say. The difficulty was never the machine's disobedience. It was our own imprecision: the assumption left unstated, the edge case never enumerated, the word that meant one thing to the business analyst and another to the developer.

That difficulty is the entire history of requirements engineering. The discipline exists because natural language is ambiguous, because stakeholders do not know what they want until they see what they asked for, and because the distance between "what was said" and "what was meant" is where defects are born. Every artefact the field produced — the use case, the acceptance criterion, the traceability matrix, the definition of done — was a device for closing that distance.

A prompt is a requirement addressed to a new kind of executor. The executor changed; the problem of saying what you actually mean did not.

What has changed with the current generation of models is the nature of the executor, and the change is real. The classical machine was deterministic and literal: it did the wrong thing reliably, which at least made the wrong thing findable. The model is probabilistic and interpretive: it guesses at intent, fills gaps with plausible invention, and can produce different outputs for the same instruction. This makes the executor more forgiving of vague input — it will always produce something fluent — and far more treacherous, because fluency is no longer evidence of correctness. But notice what has not changed. The burden still falls on the quality of the specification. If anything, a more interpretive executor punishes ambiguity more severely, because it resolves our

vagueness silently, in a direction we never see and never approved.

4. What the old discipline learned, and the new one is unlearning

The tragedy of the present moment is not that these problems are hard. It is that they are solved problems, and the solutions are being left on the shelf because they are filed under an unfashionable name. Consider what requirements engineering learned, often painfully, and how directly each lesson applies to the work now being called prompting.

- Fluency is not correctness. The discipline learned to distrust the specification that reads beautifully and specifies nothing – the requirement that says the system shall be "user-friendly" or "robust". A model's output is fluent by construction; treating that fluency as a signal of correctness is the same error, now automated and accelerated.
- Acceptance criteria come before the build, not after. The most expensive lesson of the field was that "I'll know it when I see it" is not a specification. You define, in advance, the conditions under which the thing is deemed correct. The teams now succeeding with models are the ones who write down – before they trust an output – what a good answer looks like across a range of inputs. The teams struggling are the ones judging outputs one at a time, by eye, forever.
- The edge cases are the requirement. Anyone experienced knows the real specification lives in the exceptions: the empty field, the negative number, the input in the wrong language, the record that violates the rule everyone swore was inviolable. A prompt refined against a handful of tidy examples has specified only the happy path, which is to say it has barely specified anything.
- Specification is elicitation, not transcription. The field abandoned the fantasy that requirements could simply be collected from stakeholders like dictation. Requirements have to be drawn out, challenged, and negotiated, because people cannot articulate their own tacit knowledge on demand. Prompting is elicitation too – of the practitioner's own unexamined assumptions about what "summarise this" or "assess this" actually means – and it rewards the same patient interrogation.
- The specification is an asset to be managed. Requirements were versioned, traced to their origin, and linked to the tests that validated them, because a specification nobody can find or reconstruct is a liability. The prompt that lives in one person's notes, undocumented and unversioned, its rationale unrecorded, is the reintroduction of a problem the field considered settled.

Set beside each other, the two columns are almost embarrassing in their symmetry.

The old discipline	The new costume
Functional requirement	System prompt
Acceptance criterion	Evaluation case
Requirements review	Prompt iteration
Traceability to source	(too often, nothing)

The old discipline	The new costume
Edge-case analysis	The failure discovered in production
"I'll know it when I see it"	Judging outputs by eye, one at a time

The right-hand column is not a new field. It is the left-hand column, rediscovered by people who were told, or who assumed, that the arrival of natural-language interfaces had made the left-hand column obsolete.

5. Why we cannot see what we are looking at

If the continuity is this plain, the interesting question is why it is so widely missed. The answer is not stupidity; it is a set of structural forces that each, independently, push toward the framing of prompting as new.

The first is commercial. It is in no vendor's interest to tell a market that the capability it is selling depends entirely on a specification discipline the customer already struggles with. The more attractive story is that the hard part is the model, which the vendor supplies, and that the customer's part is easy — just ask, in plain words, for what you want. "Prompt engineering" as a discrete, learnable, almost magical skill is a far better story to sell than "your requirements maturity is the binding constraint and always was."

The second force is professional identity. A new job title is a powerful thing. To be a "prompt engineer" is to be present at the founding of something, and there is understandable appeal in that, especially for people the old discipline never credentialed. But a title that severs the work from its lineage also severs it from the accumulated knowledge of that lineage. The person learning prompting from scratch, as though nothing came before, is condemned to rediscover — slowly, and in production — what the requirements literature documented decades ago.

The third force is the subtlest, and it is the seduction of natural language itself. When specification required a formal notation, or even the semi-formal discipline of a well-structured requirement, its difficulty was visible. The effort announced itself. But because a prompt is written in ordinary English, it feels effortless, and that feeling is a trap. The ease of writing the words is mistaken for ease of specifying the intent. Natural language lowers the cost of producing a specification while doing nothing to lower the cost of producing a good one — and by hiding the difficulty, it removes the very friction that used to prompt people to think harder.

"Natural language did not solve the specification problem. It hid it, which is worse, because a hidden problem attracts no discipline."

Together these forces produce a curious result: an entire practitioner community re-encountering the oldest problem in software delivery while believing itself to be doing

something unprecedented. The belief is not harmless. It determines who gets hired to do the work, what they are taught, and whether the organisation reaches for its own hard-won experience or leaves it filed under a name nobody thinks to consult.

6. The strongest case for genuine novelty — and why it deepens the point

The argument so far invites a serious objection, and it deserves to be met at its strongest rather than waved away. The objection runs: this reductive framing misses what is actually new. The executor is no longer deterministic, so specification can no longer be exhaustive in the old way — you cannot enumerate the behaviour of a system that is fundamentally statistical. Iteration is now nearly free, so the patient up-front rigour of classical requirements is not merely unnecessary but wasteful; you can try, observe, and adjust in seconds. And the interface has genuinely democratised specification, putting it in the hands of people who would never have written a requirements document. These are not trivial points. Each is true.

But each, examined, deepens the case rather than refuting it. That the executor is probabilistic does not free us from specification; it changes what a specification must contain. Because you cannot enumerate every output, you must specify the properties a good output has — its constraints, its refusals, its acceptance conditions across a distribution — which is a more demanding act of requirements thinking, not a licence to abandon it. That iteration is cheap is exactly what makes undisciplined iteration dangerous: cheap iteration without acceptance criteria is not learning, it is wandering, and it converges on whatever output happens to satisfy the person looking at the screen that afternoon. And democratisation cuts both ways. Putting specification in the hands of people untrained in its difficulties, precisely as the tools stop signalling that difficulty, is not obviously progress — it is the mass distribution of a discipline without the accompanying distribution of its craft.

The novelty is real. It simply raises the bar for specification instead of removing it. The mistake is to read "different" as "easier".

7. A worked case, invented but true to life

Consider a team in a customer-operations function that set out, in early 2024, to use a language model to triage inbound complaints — to read each message, classify its type, judge its severity, and draft a first response. The prototype was built in a fortnight and demonstrated on a set of forty representative messages. On those forty it was, by inspection, excellent: sensible classifications, well-judged severities, drafts an agent could send with light editing. The business case was written on the strength of it.

The trouble began at scale. Run across a week of real traffic, the system's severity judgements diverged from the team's own on roughly a fifth of cases — and, tellingly, the divergences clustered. Messages that were angry but trivial were being escalated; messages that were calm but described a serious regulatory breach were being soothed and de-prioritised, because the model, reasonably enough, keyed on tone. The instinct in

the room was to fix the prompt: add a line, add an example, tune the wording. Each change fixed the case in front of them and silently shifted the behaviour on cases they were not looking at. The team was iterating, energetically, and going nowhere — because they had no fixed definition of what a correct severity judgement was, only a rolling sequence of individual reactions to individual outputs.

What broke the deadlock was not a cleverer instruction. It was three days of unglamorous work that any requirements practitioner would recognise on sight. The team sat with the compliance and operations leads and wrote down what severity actually meant: the specific conditions — a named regulatory trigger, a vulnerable-customer indicator, a financial threshold — that made a complaint serious regardless of its tone. They assembled a set of about two hundred labelled examples spanning the real distribution, deliberately over-weighting the awkward cases, and made passing that set the definition of an acceptable change. Only then did they touch the prompt again, now able to tell improvement from mere movement.

The severity error rate fell below the threshold the business had needed all along. But the artefact that delivered the result was not the prompt. It was the written specification of severity and the labelled set that operationalised it — a requirements document and an acceptance-test suite, produced in 2024, for a probabilistic system, by a team who had been told they were doing prompt engineering.

8. The transformation gap, at machine speed

This is where the subject stops being a matter of technique and becomes a matter of organisational truth. For years the central puzzle of transformation has been the distance between intent and reality — between the capability an organisation believes it is buying and the capability it can actually operate. Language models have not closed that gap. They have illuminated it, and accelerated the rate at which organisations run into it.

An organisation adopting these tools is not, whatever the plan says, primarily adopting a technology. It is being confronted, quickly and publicly, with the quality of its own thinking about what it wants. The firm that has always been able to state precisely what "good" looks like — that has clear policies, unambiguous definitions, people who can articulate the rule behind the practice — will specify effective prompts almost as a matter of course, because it is merely writing down what it already knows. The firm that has run for years on tacit judgement, undocumented exceptions, and "ask the person who knows" will find that it cannot specify the prompt, because it was never able to specify the requirement, because it never actually knew, in a transmissible form, what it wanted. The model does not create this deficiency. It exposes it, and it does so at a speed that leaves nowhere to hide.

The organisation that could never write a clear requirement will not suddenly write a clear prompt. The constraint was never the tool. It was the clarity.

The enthusiasm now building around autonomous agents — systems that do not merely draft for a human to check but take sequences of actions on their own judgement —

sharpens this to a point. A vague instruction to a system that produces a draft is contained by the human who reads the draft before acting. A vague instruction to a system that acts is contained by nothing until the consequences arrive. The move from assistance to autonomy is precisely the move that removes the human safety net that has been quietly compensating for weak specification all along. Everything the discipline knows about the cost of ambiguity applies with more force, not less, the moment the system stops asking permission.

9. Recovering the discipline we already have

The conclusion is not that prompting is trivial, nor that the new tools are overrated. They are genuinely powerful, and using them well is genuinely hard. The conclusion is that the difficulty has a name, a literature, and a body of practice that most organisations already possess and are declining to use because it arrives disguised.

The practical move is one of reframing rather than retooling. The person the market calls a prompt engineer is doing the work of a requirements engineer against a probabilistic executor, and should be equipped accordingly: taught to write acceptance criteria, to hunt edge cases, to elicit tacit intent, to version and trace the specification, to distinguish improvement from movement. The evaluation set should be treated with the seriousness once reserved for the test suite, because it is the test suite. And the specification — the prompt, the examples, the definition of good — should be managed as the asset it is, not left to decay in an individual's notes.

We are fluent, as a profession, in the new vocabulary. We would do better to remember that we were once fluent in the thing it describes. The clever prompt was never the achievement. The clear specification behind it always was — and the organisations that will get the most from this technology are the ones honest enough to admit that this is where their difficulty has been all along.