

BEYOND THE CLEVER PROMPT

Your Prompt Is a Requirements Document — Start Treating It Like One

On the quiet collapse of the line between asking and specifying in enterprise delivery

Giovanni Leonardi

June 2025



Your Prompt Is a Requirements Document — Start Treating It Like One

Transformation · No. 350
June 2025

“A developer who reads an ambiguous requirement asks a question. A model does not.”

The prompt nobody called a specification

Somewhere in a delivery team this month, someone typed a paragraph into a chat window, read what came back, changed a few words, and pushed the result into a process a customer will actually touch. There was no sign-off, no version number, no note of who wrote it or why the wording changed. The paragraph told a system what to produce and how to behave. It was, in every sense that matters, a specification — and nobody in the room called it one.

That gap, between what these instructions are and what we are treating them as, is the most under-examined shift in enterprise delivery right now. We have spent the better part of thirty years arguing about requirements: how to elicit them, baseline them, trace them, stop them changing under our feet. And now, just as large language models move from novelty to load-bearing infrastructure, the requirement has quietly changed costume. It has become the prompt. Same job — tell the thing what to do — but stripped of nearly every discipline we built up around it.

The requirement in disguise

Strip a requirement back to its function and it is simply a statement of intent that a downstream party converts into behaviour. For decades that downstream party was a developer, and the whole apparatus of specification existed to close the gap between what we meant and what they built. A prompt does exactly the same work. The only thing that has changed is the recipient: the instruction is now interpreted by a probabilistic model rather than a person.

That single change should make us more careful, not less. A developer who reads an ambiguous requirement asks a question. A model does not. It resolves the ambiguity silently, plausibly, and differently each time the wording shifts.

Consider a team standing up a customer-facing summarisation feature: support cases in, tidy summaries out. The instruction that governs it runs to perhaps four hundred words. One line reads be concise. Someone, reasonably, decides the summaries are dropping detail and changes it to be thorough. Handling time per case moves the next morning,

because summaries that were three lines are now nine and agents read every one. That edit was a requirements change with a direct production consequence. It lived in a chat message. It was reviewed by no one, traced to nothing, and attributed to whoever happened to have the window open.

A prompt is the first requirements artefact in a generation that is simultaneously business-critical, natural-language, and completely ungoverned. We spent years removing the ambiguity of natural language from specifications on purpose. It has walked back in through the front door.

What actually changed

Three properties make this more than a naming quibble.

- Natural language is back at the centre. Formal requirements, user stories, acceptance criteria were a long march away from prose precisely because prose is ambiguous. The prompt returns us to the paragraph, with all its slack.
- The output is non-deterministic. The same instruction can yield different behaviour on different runs, so "it worked when I tried it" stops being evidence that it will work.
- Behaviour is brittle to phrasing. Reordering two sentences, or adding a stray always, can shift results in ways no one intended and no one can easily predict.

None of this is an argument against using the models. It is an argument against pretending the instructions that drive them are not real artefacts.

The obvious objection

The reasonable pushback is that this is configuration, not requirements — a setting like a threshold or a feature flag, and we do not wrap every config value in a change-control board. If prompts were stable, narrow switches, that would hold. But a configuration value does not carry intent, edge cases, tone, and a dozen implicit business rules in a single editable paragraph. This one does. It sits much closer to the specification end of the spectrum than the settings end, and the blast radius of a careless edit is the tell.

Treating it as a specification

What this means in practice is not heavyweight process. The point is far simpler: name the artefact, and give it the lightest discipline a business-critical instruction deserves.

1. Version it. The prompt lives in source control with everything else, carrying a history of what changed and why.
2. Review it. A change to production behaviour gets a second pair of eyes, exactly as a code change does.

3. Test it. Keep a small bank of representative inputs and expected characteristics, and re-run them when the wording changes.
4. Trace it. Someone should be able to ask why does it behave this way and follow the answer back to a decision.

We already know how to do all of this. We built it for requirements the first time around. The uncomfortable truth of this particular moment is that the most important specification in a growing number of systems is a paragraph of prose that no methodology currently claims. The teams that avoid a bad year will be the ones who notice that early — and start treating the prompt with the seriousness its job now demands.