

BEYOND THE WORDING

Your Prompt Is Not a Specification

Why treating prompts as requirements imports the word and leaves the discipline behind

Giovanni Leonardi

April 2025



Your Prompt Is Not a Specification

Transformation · No. 347

April 2025

*"A specification you cannot test is not a specification —
it is a wish written in confident prose."*

In a delivery review not long ago, someone turned the screen around, pointed at a carefully worded paragraph of instructions to a language model, and said: "That's our specification now." Around the table, people nodded. The paragraph was genuinely good — layered, specific, the product of a fortnight's tuning. And yet nobody present could answer the three questions any competent analyst would have put to a specification a decade ago. What happens when it produces the wrong answer? Who agreed its output was right in the first place? And what, exactly, has changed since the last time it worked?

That silence is the subject of this piece. The claim that prompt engineering is the new requirements specification has become one of the defining slogans of the moment, and it is half right in a way that makes its other half dangerous. It is right that natural-language instruction has become a primary artefact of building software — that a paragraph of English now sits where a signed-off requirement used to sit. It is wrong in what it quietly assumes: that because the prompt occupies the same position, it inherits the same discipline. It does not. The discipline is precisely the part the textbooks leave out.

1. What the analogy gets right

Give the analogy its due, because there is real substance in it. For most of the last two decades, the person who could translate a fog of stakeholder intent into a precise, buildable statement held a particular kind of leverage. That skill has not disappeared; it has changed clothes. When the interface to the machine becomes ordinary language, the ability to say exactly what you mean — to decompose a vague ambition into unambiguous, ordered, testable instructions — becomes valuable again in a very direct way. The craft of precision under ambiguity, which is what requirements elicitation always was at its core, is genuinely continuous with what a good prompt author does when they sit down to make a model behave.

So the slogan is not empty. The prompt really has become a load-bearing artefact — the place where intent is captured and from which behaviour follows. If that were all the analogy claimed, I would have no quarrel with it. The trouble is that it claims more, by implication, and the implication is where teams get hurt.

2. What the textbooks leave out

The guides teach wording. Role framing, worked examples, step-by-step decomposition, the careful ordering of instructions — all of it useful, none of it the point. Because requirements engineering was never really about the wording either. Its value lived in four things the prompt literature barely mentions.

- Agreement. A requirement was a treaty between people. Its worth was not in the elegance of its phrasing but in the fact that a sponsor, a user, and whoever had to sign the cheque had argued their way to a shared meaning before anyone built anything.
- Traceability. You could follow a requirement forward to the design, the test, and the line of code that satisfied it — and backward from a defect to the intent it violated.
- Acceptance. "Done" was defined before the build began, in terms a second party had accepted. The test existed before the thing it tested.
- Change control. When the world moved, you knew what to re-examine, because you knew what depended on what.

A prompt, as most teams actually hold it, has none of these. It is written by one person, judged by that same person against their own private expectation, versioned — if you are lucky — as a string in a file behind a commit message that says "improved prompt," and validated by the fact that the demo looked good on Tuesday.

Watch what that costs. A team ships a summarisation feature whose behaviour is governed by a three-hundred-word prompt they proudly call their source of truth. Three weeks later, support tickets climb: the summaries have started dropping the final section of long documents. The team has changed nothing. The provider has updated the model sitting behind the same unchanged interface. And because the prompt carried no acceptance test and no pinned notion of what "correct" had ever looked like, nobody can even establish where the regression lives — in the model, in the prompt, or in the documents. They spend four days rediscovering, painfully, that a specification you cannot test is not a specification — it is a wish written in confident prose.

3. The compiler moved underneath

There is a deeper reason the discipline matters more here, not less. Requirements engineering grew up on top of a deterministic build. However far the toolchain sat below you, it was faithful: the same source produced the same program, so if the behaviour changed, something in your artefacts had changed. That single assumption is what made traceability possible. It is the spine of the whole practice.

The prompt sits on top of a compiler that is neither deterministic nor yours. It can return different output for the same input from one run to the next, and — worse — it can be revised by someone you have never met, on a schedule you do not see, behind an interface that looks identical before and after. Break the deterministic-build assumption and traceability snaps at the spine. A defect can no longer be reasoned backward to a cause inside your own work, because the cause may live in a system you neither wrote nor pinned.

A prompt without a recorded model version is undated. Its meaning is a function of a compiler you do not control and did not pin – and a specification whose meaning drifts while its words stay still is the one thing requirements engineering was built to prevent.

4. But isn't this the dream coming true?

The strongest version of the counter-argument deserves a real hearing, because it is genuinely good. It runs like this. For fifty years requirements engineering dreamed of the executable specification – a statement of intent that is also the thing that runs, closing the gap between what you asked for and what you got. Prompts, paired with evaluation suites, finally deliver it. You write intent in English; you write evals that check the intent; the distance between specification and behaviour collapses to nothing. Non-determinism is merely a testing problem, and we have tested probabilistic systems before.

I think this is the best objection available, and I think it fails on a single word: agreement. An executable specification is not the same as an agreed one. An eval encodes one person's guess at what the stakeholder meant, frozen at one instant; it automates the checking of intent, never the negotiation of it. And negotiation was always the hard part. The genuine difficulty of requirements was never getting the sentence written down – it was discovering that the sponsor, the user, and the regulator each meant something different by the same sentence, and reconciling them before code was cut. Prompting has not solved that problem. It has concealed it, because the model will always return something plausible, and plausibility is a very effective disguise for un-negotiated intent.

*“Plausibility is a very effective disguise for
un-negotiated intent.”*

The eval, too, is only ever as honest as the person who wrote it, and it can pass this month and fail the next with not one line of your own code changed. Executable, then – yes. Agreed and stable – no. And those were the two properties that made a specification worth the name.

5. What is actually worth carrying across

The right response is not to drop the analogy but to take it seriously enough to import the whole discipline rather than only its vocabulary. Four practices travel directly, and they are the four the wording guides omit.

1. Treat every prompt as untested until it has acceptance criteria a second person has agreed to. The prompt is the wish; the eval a second party signed off is the specification. Until then you have written prose, not a requirement.

2. Pin and version the model, not only the prompt. A prompt's meaning is a function of the model it runs on. Recording the prompt while leaving the model implicit is like versioning your source and letting the compiler change silently – and then being surprised when the binary does.
3. Recover elicitation. The analyst's oldest craft – dragging tacit, unspoken intent into the open before building on it – did not vanish because the interface became English. Skip it and the tacit-requirements gap returns in a new costume: as confident, fluent hallucination that no one thought to forbid because no one thought to ask.
4. Keep a provenance trail. Which prompt, on which model, with which examples, produced which behaviour, judged against which criterion. Without it, every regression is an archaeology dig.

What requirements engineering enforced	What the prompt-as-spec usually has
Agreement negotiated between parties before build	One author's private intent, never reconciled
Traceability from intent to test to code	A string in a file with a vague commit message
Acceptance defined before the build began	"The demo looked right on Tuesday"
Change control over a known dependency graph	Silent drift as the model is revised upstream
A deterministic, owned compiler	A probabilistic compiler owned by someone else

None of this is nostalgia for heavyweight documents. The forms that requirements engineering once wrapped around these ideas were often bloated, and good riddance to the worst of them. What deserves to survive is not the paperwork but the four disciplines underneath it – and they deserve to survive precisely because the new substrate punishes their absence faster and more quietly than anything we have built on before.

6. The price of the slogan

The slogan will keep its appeal, because it flatters the moment. It tells a generation of builders that a craft their predecessors spent careers learning has been compressed into the knack of writing a good paragraph. Some of that is even true, and the part that is true is exhilarating. But the teams still standing when the novelty wears off will be the ones who heard "the prompt is the new specification" not as permission to stop doing requirements engineering, but as an instruction to finally start doing it again – on ground that gives less warning before it gives way. The prompt did not retire the discipline. It raised the price of skipping it.