The Future of Agentic AI Automated Technical Documentation is Already Here

An overview of the state of professional technical documentation past, present, and future

Part 1 of 3: Agentic documentation: How we got here

MICHAEL IANTOSCA
SENIOR DIRECTOR OF KNOWLEDGE PLATFORMS AND ENGINEERING
AVALARA

This is a three-part paper:

Preface: Why Should you Listen to Me?

Over 45 years, I've worked across nearly every role in technical documentation: writer, editor, content strategist, information architect, systems builder, platform architect, linguist, localization lead, terminologist, taxonomist, ontologist, knowledge graph evangelist, and technical development leader. An astute observer might say "It sounds like the man can't hold a damn job!" -- which might not be all that far from the truth.

I've been involved with AI since 1991, when it was called Expert Systems, through the IBM Watson era and long before generative AI became popular. Together with peers who build documentation systems at scale, I've spent decades working at the edge of emerging standards and methodologies - often long before they become mainstream. We test ideas by shipping them, supporting them, and proving their value with measurable results.

This perspective comes from practice, not punditry. I don't sell tools, frameworks, or hype. Believe me, we've developed instant hype meters as a result.

I also have formal training as a senior software engineer and systems architect. That dual background allows me to treat documentation not as an afterthought to software, but as a system in its own right.

Early AI: Painful Observations

When generative AI entered the mainstream with the debut of OpenAI's ChatGPT, the industry moved fast - understandably so. Almost overnight, a wave of stochastic chatbots emerged. Using an open-source vector database such as Pinecone and a modest amount of Python, it became trivial to assemble a RAG-style chatbot system in a matter of days. That speed was both impressive and misleading.

What became clear very early is that probabilistic retrieval layered on top of probabilistic large language models introduces fundamental weaknesses - especially for use cases that demand accuracy, repeatability, and trust. Yet years later, much of the conversation is still focused on explaining RAG itself, rather than interrogating its limits. That disconnect has been striking.

This wasn't the result of incompetence. It was pressure. Many development teams were asked - often urgently - to "do something with AI," driven by fear of being left behind. Developers reached for what was available, fast, and familiar. The result was mass adoption of approaches that were quick to demonstrate but difficult to harden into reliable and scalable systems. From the outside, it was easy to see how brittle many of these implementations were; from the inside, the tradeoffs were often unavoidable.

That fear hasn't gone away. It's now compounded by competitive pressure and commercial incentives. That's not a moral failing, it's how markets work, but it does explain why certain architectural mistakes continue to be repeated long after their limitations are well understood.

Those who recognized these issues early tended to share a common background - formal knowledge management. Experience in that discipline teaches a simple but uncomfortable lesson: effective AI systems are less about clever code and more about disciplined content, explicit context, and deterministic augmentation supported by governance. These are not shortcuts. They require time, skill, and sustained investment - things largely antithetical to corporate leadership reward systems.

The hard truth is that genuinely useful AI systems are built, not bolted on. And the work that makes them trustworthy happens well before a model ever generates its first token.

The Precision Paradox in AI

As AI systems improve in accuracy, they also become less tolerant of even small errors -- a phenomenon I describe as the Precision Paradox.

Early RAG implementations produced approximate responses. Because overall accuracy was low, minor mistakes were expected and dismissed. As models improve, the expectation of accuracy rises. Small errors become far more visible and consequential. A tiny factual inaccuracy or incorrect context fragment can break trust entirely, especially in regulated industries.

Improving precision does not simply reduce error -- it reduces error tolerance. The better a system appears, the less its mistakes can be ignored.

Key implications:

- **Probabilistic approaches have hit a ceiling.** Plain vector RAG delivers broadly plausible answers but struggles to achieve deterministic accuracy for regulated domains.
- **Precision demands structure and governance.** Knowledge graphs, ontologies, and semantic relationships provide verifiable context that probabilistic methods lack.
- **Perception matters as much as performance.** Users judge systems by their worst errors, not average quality.

With AI support bots, the deficit is initially acceptable -- like when substantially reducing support costs through self-service. But in agentic workflows, lack of precision cascades and is often catastrophic.

The False Hope and Hype of Improving AI Models

LLM spinmeisters have had a good run. Now they're mostly out of fumes.

They had to hype; they overspent on infrastructure chasing market share and future profits, dug a massive financial hole, and now there's only one way out: make everyone burn tokens.

Burn, baby, burn.

The dirty secret they won't say out loud: no matter how glossy the demos get, LLMs plus vector databases are probabilistic by design and permanently non-deterministic. There is no magical model upgrade that defeats basic math, no matter how much they'd like you to believe otherwise.

Large Reasoning Models – Unreasonable Hope

Take "large reasoning models" as the most obvious tell. LRMs do not perform formal reasoning; they are instead recursive probabilistic checks stacked on top of probabilistic checks. Does it help? Marginally. Techniques such as Chain-of-Thought (CoT) are what should have existed in these models from day one, now repackaged, buffed shiny new, and sold as “reasoning” as if it were some breakthrough. Good grief!

It is not deductive reasoning at all. It is Splenda reasoning – it looks like the real thing, sounds like the real thing, but metabolizes into nothing useful when you actually need deduction, constraint satisfaction, or grounded inference. No real context. No determinism. No guarantees. Just vibes, and a higher bill.

Large Context Models – More Hijacking

Large Context Models (LCMs) offer only marginal improvements over baseline LLMs. Added context does not create meaning; it creates surface area. More tokens mean more retrieval noise, more opportunities for spurious correlations, and more chances for the model to confidently average its way to something that merely sounds coherent.

And if you are starting to notice the pattern: more tokens to burn. Ka-ching!

Stuffing a model with a million tokens does not produce understanding any more than giving a parrot a larger library makes it a scholar. You do not get structure, intent, or truth. You just get a longer autocomplete window, one that is easier to hijack, harder to reason over, and increasingly expensive to run.

LCMs confuse availability with semantics. They assume that if the answer exists somewhere in the context blob, the model will infer which parts matter, how they relate, and which constraints are binding. In practice, the model has no grounded notion of authority or validity. It cannot reliably distinguish what is authoritative, what is provisional, what is stale, or what contradicts what. Everything is just text with weights. Context without form is just noise at scale, and the only thing scale reliably amplifies is error.

Worse, LCMs actively mask failure. The more context you add, the harder it becomes to notice when the model is hallucinating, smoothing over contradictions, or inventing glue logic to bridge gaps it does not understand. The output feels richer, more grounded, more “thoughtful,” while being no more reliable than before, and often less so. This is not comprehension; it is narrative interpolation.

And of course, LCMs are wonderfully expensive. They turn every query into a token bonfire, shifting costs from model training to perpetual inference. That is not an accident; it is the business model. If you cannot make models smarter, make prompts bigger. If you cannot guarantee correctness, drown uncertainty in verbosity. Call it “contextual intelligence” and send the invoice.

But context is not cognition. Memory is not meaning. Scaling ignorance does not turn it into knowledge; it only makes it harder to challenge.

The Uncomfortable Truth LLM Makers Don't Want You to Know

Meanwhile, anything that actually reduces LLM workload and token mouth-stuffing, such as knowledge graphs, governed content, formal context, and deterministic inference, is often commercially misaligned with current LLM provider incentives. Not because it does not work, but because it disrupts a business model optimized for token consumption.

Real reasoning systems and external inference engines do not require sustained token consumption.

And that is precisely the tension many providers would rather not highlight.

The Early Days of AI in Technical Documentation

This was more than a little amusing to me. For years prior, I had been roaming conference halls explaining the essential need and value of semantic knowledge models and stacks: taxonomy, ontology, knowledge graphs, the whole nerdy trifecta. I had slides. Diagrams. Papers. Blog posts. Many of them are still languishing on my personal website, dating back to the early 2020s, quietly whispering, "I told you so."

It was never an easy sell. Eyes glazed over the moment I began defining KM concepts that had spent decades safely tucked away in Library Science departments and the shadowy corners occupied by Knowledge Management professionals, those mysterious figures most companies technically employed but rarely listened to. To most audiences, it all sounded dreadfully academic. And boring. And irrelevant.

Oh, how times have changed.

Now everyone, and yes, seemingly everyone's AI-savvy kid, is talking about semantic meaning, structure, context, provenance, determinism, reasoning, and inference. The list goes on. There are still too many who remain confused and ignorant, but at least one hard truth has landed: probabilistic methods alone, duct-taped together with prompt hacks and jury-rigging, have reached a practical dead end, and everyone now knows it as they desperately seek remedies.

That dead end is called accuracy and trust. And it turns out you cannot make agentic workflows fly in scalable production deployments on intuition and surface plausibility alone.

So How Did the Technical Documentation Community Respond?

First, they panicked. Naturally.

The initial fear was that technical writing jobs were about to evaporate. At the moment ChatGPT emerged, this fear was objectively premature. There was no immediate risk. Yet. Panic ensued anyway.

I remember standing in the main conference room at ConVEx documentation conferences, walking through how generative AI actually worked, the mechanics, the limitations, the hallucinations. I explained, patiently, that these models were nowhere near mature enough for technical documentation professionals to be worried. Yet.

But fast-forward. It is no longer 2022. It is 2026.

Now it is time to worry. And, more importantly, to act.

As the panic subsided, writers did what writers do: they experimented. They explored ChatGPT, discovered where it was useful, and quickly learned where it fell apart. The community's collective attention landed on prompt engineering, which was the right move at the time. It was accessible. It delivered immediate gains.

More advanced and ambitious technical documentation practitioners moved on to Custom GPTs, tuning them for specific tasks and domains. Again, a reasonable step, but still insufficient for systematic, trustworthy, industrial-scale technical communication. Many of these custom solutions persist today, but they are increasingly limited compared to more robust approaches now emerging with systems-level agentic AI.

Meanwhile, retrieval-augmented generation (RAG) continued to evolve. Building RAG systems, however, required deeper software engineering skills, and these efforts quickly became the domain of software engineers with little to no understanding of the standards, constraints, and rigor required for technical communication.

Too often, those engineers absconded with the documentation corpus in the middle of the night like content thieves, sometimes with the blessing of documentation teams, more often without. The consequences of those decisions are still reverberating.

At the same time, documentation tools and systems vendors, fearful of becoming obsolete, rushed to integrate AI into their offerings. One or two content management systems providers leaned into taxonomy systems integration. Content delivery platforms shipped with integrated chatbots. Advanced editors such as Oxygen added powerful AI assistance like Positron. Linters such as Acrolinx and Congree doubled down on their already strong NLP foundations. Translation management systems, such as XTM, followed suit.

These augmentations were and remain genuinely useful, especially for organizations lacking the dedicated engineering resources they should have on their teams, but do not. Yet in the context of the end-to-end content supply chain, each of these tools addresses only a partial segment of a much larger lifecycle, from knowledge extraction through publishing, maintenance, and eventual retirement.

That will remain true until one of two things change.

Either documentation teams acquire the dedicated engineering resources they deserve and have long lacked, or documentation systems providers evolve beyond surface-level AI augmentation.

There is, however, a hard limit to this vision. Most documentation tool vendors address only isolated fragments of the content development and delivery ecosystem. To date, no vendor supports a true end-to-end documentation supply chain. More troubling, many vendors still fail to understand what documentation teams are really asking for.

When they eventually see what the most advanced organizations are already building in-house, they may finally begin to grasp both the problem and the opportunity (more brutally frank commentary on that later in this paper).

What technical documentation organizations need is end-to-end process automation and governance, from raw knowledge extraction through publication, maintenance, and retirement. The whole enchilada.

The documentation teams that do have engineering resources and that take a systems view are already building these solutions themselves. They are enjoying the gains. And yes, they are also experiencing the inevitable heartburn that comes with any genuine, epochal transformation.

Prompting, GPTs, and the Long Road to “Oh… This Is Harder Than It Looks”

Writers did what writers always do: they experimented. ChatGPT proved useful in narrow, well-defined tasks and unreliable everywhere else. Prompt engineering emerged as the first collective adaptation, not because it was elegant, but because it was accessible and immediately productive.

Some practitioners went further, building Custom GPTs tuned for specific domains and workflows. This was a logical next step, and in many cases a meaningful one. But these approaches remained bounded. They improved local efficiency without addressing the underlying system.

Most of these GPT-based solutions are still in use. They are serviceable. A few are genuinely clever. But almost all of them plateau well below the rigor, reliability, and scalability required for industrial-grade technical documentation.

Enter RAG (and Exit the Docs Team, Stage Left)

Retrieval augmented generation (RAG) changed the center of gravity.

Properly implemented, retrieval-augmented generation demanded real engineering: pipelines, retrieval strategies, orchestration, and evaluation. Inevitably, ownership shifted toward software teams, many of whom viewed documentation as unstructured input rather than a governed system with standards, constraints, and lifecycle requirements.

In many organizations, the documentation corpus quietly migrated out of the documentation function. Sometimes collaboratively. Often not.

The architectural and organizational consequences of that shift continue to play out.

Vendors Panic Too (They Just Wear Better Suits)

Vendors reacted just as quickly.

AI features appeared everywhere:

- CMS platforms added taxonomy automation and classification
- Delivery platforms embedded chat interfaces
- Editors shipped AI-assisted authoring
- Linters extended NLP-driven guidance
- Translation systems layered in generative workflows

These capabilities are genuinely useful – especially for teams without dedicated engineering support. They reduce friction and deliver real, incremental value.

But they remain incremental.

Each tool optimizes a narrow segment of the content lifecycle. Useful segments, yes – but isolated ones. None address the documentation supply chain as a cohesive, end-to-end system.

That limitation persists until one of two things changes:

1. Documentation teams gain sustained engineering capacity

2. Vendors stop augmenting workflows and start redesigning them

The problem with Option #2 is that there’s a structural ceiling most vendors can’t break through with agentic AI.

With few exceptions, commercial documentation tools own only fragments of the overall system. Only platforms that control an end-to-end content development and delivery pipeline are even positioned to tackle full lifecycle automation – and none have done so yet.

More importantly, many vendors are solving the wrong problem.

What practitioner teams are asking for is not another chatbot layered onto an existing interface. They’re asking for operational coherence.

They need end-to-end automation with explicit governance:

- From authoritative knowledge capture
- Through structured authoring and validation
- To controlled publication, maintenance, and retirement

Yes – the whole enchilada. Sauce included.

The Quiet Winners

A small number of documentation teams have already moved past this impasse.

They have engineering resources. They take a systems view. And rather than waiting for vendors to catch up, they’re building what they need themselves.

These teams aren’t experimenting with features – they’re engineering pipelines. They’re encoding judgment into workflows and turning institutional knowledge into executable systems.

It’s not painless. Transformations of this scale never are.

But for now, they’re eating well.

Next:

Part 2: The real deal: Agentic AI documentation automation – Case studies