The Future of Agentic AI Automated Technical Documentation is Already Here

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

MICHAEL IANTOSCA
SENIOR DIRECTOR OF KNOWLEDGE PLATFORMS AND ENGINEERING
AVALARA

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 built-in bullshit 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 agentive 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.

The Real Deal: Agentive Process Automation

As the title says, the future of technical documentation is already here. Most simply haven't seen it in operation yet.

Individual AI tools are useful; they'll always play a role, however, the real power of AI is in agents and agentive workflows to partially or fully automate documentation processes.

A handful of technical documentation shops with strong software engineering resources and skill are paving the agentic way for all shops. They are the proverbial “canaries in the cave” pioneering what is yet to arrive across the technical documentation spectrum. They’re also experiencing the once-in-a generational transformation it brings, the good, the bad, and the ugly.

Before we dive into the nuts and bolts, let’s step back. You might ask: What is the most important element when approaching this? Is the technology and tools? The skills? The training? Management buy-in? It’s all of the above, but above all…

It’s *process analysis*.

Step One: Process Analysis

Most organizations building AI systems - particularly agentic, process-oriented solutions - are addressing the wrong problem. This is not a failure unique to technical documentation; it is an industry-wide pattern. Engineering teams tend to concentrate on implementation details: code, models, orchestration frameworks, and tooling. While complex, these components are largely deterministic and solvable with the aid of stochastic AI.

The primary challenge lies elsewhere: *accurately modeling the domain*. This requires a comprehensive, end-to-end understanding of how the content supply chain actually operates in production, including non-ideal paths, exceptions, and failure modes. Achieving this understanding demands process analysis, systems thinking, and sustained engagement with subject matter experts (SMEs). These activities are often mislabeled as “soft skills” and subsequently deprioritized. In practice, they are foundational system inputs.

Technical documentation pros don’t just “know the tools.” They accumulate hard-won operational knowledge over years of doing the work. Very few people - even inside the discipline - actually understand the full ecosystem. That knowledge is tacit, experiential, and almost never written down. Engineers outside the domain don’t stand a chance of reconstructing it on their own. It’s not in Jira tickets, PRDs, or architecture diagrams. And no, it can’t be magically reverse-engineered by AI engineers or agent frameworks. If it isn’t explicitly captured and validated, it simply does not exist.

What *is* typically documented are high-level workflows and idealized happy paths. Missing are the hundreds of micro-decisions, heuristics, exception-handling rules, and judgment calls that technical documentation professionals apply implicitly. A good swath of timeless best practices have been documented in time-tested books such as Developing Quality Technical Information: A Handbook for Writers and Editors.

These and many more behaviors are learned through repeated exposure to edge cases and are executed automatically by experienced practitioners. They cannot be extracted through a small number of interviews or a cursory documentation review - particularly when system designers lack domain training.

As a result, undocumented assumptions are encoded directly into AI systems. These systems are deployed with confidence, but their outputs diverge from real-world expectations. In some cases, failures are subtle; in others, they are highly visible and operationally disruptive.

This pattern has been especially evident in the technical documentation domain, which has been falsely treated as a low-risk target for AI automation. Familiarity with documents does not equate to understanding how quality documentation is authored, reviewed, governed, versioned, and maintained over time. Consumption is not production, and surface-level artifacts obscure the underlying decision systems that produce them.

To address this gap, institutional knowledge must be explicitly captured - not as rigid, procedural workflows, but as durable, governed, declarative representations. Ontologies and knowledge graphs provide the necessary abstraction layer to model concepts, relationships, constraints, and domain rules in a form suitable for AI reasoning.

This modeling effort is the hardest task of designing, implementing, and governing the system. It is also the part most frequently skipped in the interest of speed. Until organizations invest in this work, AI initiatives will continue to underperform - or fail outright - regardless of how advanced the underlying models appear to be.

When decomposing the documentation development lifecycle for agentive AI, an initial analysis typically surfaces dozens of discrete agents required to support the nominal end-to-end workflow. These agents represent only the visible layer of the system. The majority of system complexity resides beneath this surface, embedded in fine-grained decisions, conditional logic, and domain-specific exceptions that emerge only under detailed process examination.

Shift Left: Transformation of Responsibilities

Forget everything you think you know about roles in technical documentation. The field has once again packed up its desk and moved offices – without telling anyone where it went.

Agentic AI workflows significantly diminish, if not outright eliminate, the need for humans to write initial drafts of technical documentation. That’s a bitter pill for those of us who’ve spent entire careers mastering the craft. But bitter doesn’t mean surprising. Computers

have always been about automation as much as capability, and generative AI is simply the next step in that long, familiar march – just happening at warp speed.

Some of us are old enough to remember the trauma of abandoning typewriters for mainframe terminals, then personal computers. We remember building early electronic technology and books with primitive hypertext links. We remember the first web browsers, multimedia CD-ROMs, SGML, XML, DITA, and every other "this will change everything" moment that actually did. Each wave reshaped the profession. We evolved from Technical Writers to Information Developers, then to User Experience (UX) professionals and a constellation of specialized roles orbiting product teams.

This moment is no different.

Just as electronic publishing made typesetters obsolete, agentic AI makes human-written initial drafts optional. And here's the irony: in some organizations, we're witnessing a reversion. Engineers are once again picking up the pen – or at least operating the model.

Decades ago, development cycles were long – years, in some cases. Engineers wrote product design and test specifications, often hundreds of pages each. Technical writers used those documents as a foundation, layering audience analysis, task modeling, and structured documentation plans on top. Objectives, assumptions, dependencies – all the things that made documentation usable rather than merely present.

Then development sped up. Agile arrived. Specs quietly vanished. Information Developers became sleuths – investigative journalists playing *Where's Waldo?* with source knowledge. We learned by using the product, interviewing engineers, reading code, reverse-engineering intent, and filling in the gaps no one else had time to notice. That's been the job for years.

Now the pendulum is swinging again.

In early agentic-AI documentation shops, some organizations have reduced writing teams dramatically – in some cases cutting upward of three-quarters of their technical writing staffs. What happened wasn't magic. Documentation workflows are "shifting left," back to development teams, who already hold most of the source knowledge in their heads. AI simply made that shift operationally possible.

Author's note: This isn't an endorsement. It's an observation.

On one level, it's pure karma. Developers, long relieved of the burden of capturing and documenting design intent, are once again responsible for doing exactly that. I highly doubt they're going to be thrilled, but more than a little excited in a different way.

Personally, I'm taking cover in a virtual fall-out shelter.

On another level, it's a looming disaster if engineers become the sole owners of end-to-end documentation. Writing usable documentation isn't just about assembling sentences – it's about audience awareness, task clarity, information architecture, and ruthless prioritization. Those are not skills you pick up between stand-ups, and they're not fully replaceable by AI – at least not yet.

There are also some inconvenient realities these organizations haven't fully encountered. Engineers are task with and prefer to build, test, and maintain products. Documentation has never been their favorite pastime. Even with AI handling the "writing," experienced documentation professionals know the truth: **the actual writing is only about 10–15% of the job as every study confirms**. The rest is knowledge collection, research, synthesis, validation, QA, and judgment – ensuring accuracy, completeness, usability, and coherence across systems that rarely behave politely.

Attention, engieeers: welcome to our world.

Engineers are also, understandably, far more expensive than technical writers. Their time directly drives revenue (though without usable technical doc, there is no product). And now they're being asked to do what they arguably should have been doing all along: documenting their designs, describing implementations, commenting code thoroughly, recording demos that are actually complete, and producing artifacts clean enough for AI ingestion.

Senior management may love this on a slide deck and mandate it. Engineers? Less so. This transfer of responsibility is not going to be met with applause. It's going to be a slow-burn thriller.

I've got my popcorn ready.

Now add AI-assisted coding to the mix. Place your bets: will product development accelerate so dramatically that documentation demand explodes, or will developer ranks shrink, leaving the remaining engineers with oceans of spare time to document everything – even with agentic AI doing the heavy lifting? My bets are on the former.

However this suspense film ends, one thing is already clear: agentic AI isn't killing the profession. It's mutating it. And in the process, it's spawning entirely new roles for those former technical documentation professionals who know where the bodies – and the missing context – are buried.

The credits haven't rolled yet.

New and Emerging Roles in Agentic AI-Driven Technical Documentation

The shift from document production to agentic, ontology-driven documentation systems has introduced a set of new and materially different roles. These roles reflect a transition from writing and editing toward system design, semantic governance, process automation, and operational oversight. The roles do not imply individual jobs; some may be combined or shared responsibilities.

- **Agentic Workflow Pilot**: Operates stateful agentic AI technical documentation workflows or monitors stateless workflows. Exactly where this role resides is a business decision up for debate.
- **Documentation QA Engineer**: Intervenes with complex content issues that lie beyond the current scope and capabilities of the agentic processes or the skills of the agentic workflow operators. These are more akin to current technical documentation professionals, only fewer of them with highly specialized content and systems skill sets. Over time, this role can diminish with more advanced agentic AI capabilities, but not anytime soon.
- **Knowledge Architect**
  Defines and governs the ontologies and semantic models that establish what the organization knows and how concepts relate.
- **Documentation Platform Engineer**
  Builds and operates the end-to-end documentation infrastructure across CMSs, agents, knowledge graphs, and delivery channels.
- **Agent Workflow Designer**
  Translates editorial judgment and process logic into executable agent behaviors, validation loops, and escalation rules.
- **Semantic QA Lead**
  Ensures factual consistency, ontology alignment, and semantic integrity across all machine-generated content.
- **Context Engineer**
  Controls how knowledge is retrieved, filtered, structured, and delivered based on user intent and task context.
- **Ontology Product Manager**
  Owns the roadmap, scope, and tradeoffs of the organization's meaning model as a managed product.

- **AI Enablement Lead**
  Makes agentic and ontology-driven systems usable, safe, and governable for non-engineering teams.
- **Knowledge Graph Engineer**
  Implements and optimizes the graph infrastructure that supports semantic retrieval, reasoning, and dependency analysis.
- **Retrieval Architect**
  Designs hybrid retrieval strategies that balance precision, recall, trust, and explainability.
- **Content Systems Analyst**
  Measures semantic coverage, reuse, drift, and performance across the documentation knowledge supply chain.
- **Legacy Content Migration Lead**
  Transforms page-based legacy documentation into structured, graph-aligned, AI-ready knowledge.
- **Trust and Provenance Officer**
  Owns source attribution, versioning, lineage, and auditability of machine-mediated knowledge.
- **Documentation Systems Architect**
  Designs the overall topology and integration strategy of the documentation ecosystem as a coherent system.
- **Process Modeler and Domain Modeler**
  Captures tacit domain knowledge, heuristics, and exception logic into formal, machine-usable representations.
- **Agent Safety and Failure Engineer**
  Designs safeguards, rollback strategies, and failure controls to limit automation risk and blast radius.
- **AI Operations Lead for Content**
  Operates agentic documentation systems in production, managing workflows, failures, and releases.
- **Semantic Infrastructure Product Owner**
  Owns shared semantic infrastructure across documentation, support, product, and analytics domains.

- **Documentation Data Engineer**
  Builds and maintains the data pipelines, metrics, and observability required to operate documentation systems at scale.
- **Human Review Strategist**
  Defines where and when human intervention occurs in agentic workflows to balance safety and scalability.

Extended role descriptions, responsibilities, and failure modes are provided in *Appendix A: New and Emerging Roles With Agentic AI*.

At a macro level, what's really happening is skill and role convergence. Professional documentation roles are converging with formal knowledge management roles and also converging with or partnering tightly with content platform and agentic design and engineering roles. At the same time, entirely new content operations and governance models are emerging; these will quickly take center stage.

Case Studies: Agentic AI Workflow Automation for Technical Documentation Emerge

This case study examines the design, implementation, and operational deployment of an industry-first agentic AI system used to automate large portions of a technical documentation supply chain. The initiative demonstrates how agentic workflows - when grounded in structured content, deterministic transformations, and disciplined process modeling - can safely and effectively operate at enterprise scale.

Agentic Program Overview

The AI Product Documentation Automation initiative was established to modernize and automate our documentation ecosystem using agentic AI workflows. The program encompasses two production-grade solutions currently in operation, with a third currently under development.

The initiative was explicitly as a production system with clear success criteria, operational constraints, and long-term extensibility requirements from its inception – not as an experiment or proof-of-concept.

Implemented Solutions

Solution 1 – One-Time Corpus Refactor
An agentic AI workflow designed to refactor Avalara's entire existing technical documentation corpus in a single, large-scale operation.

Solution 2 – Net-New Content Automation
An agentic AI workflow that automates the creation, scoring, refinement, and publication of new technical documentation moving forward.

Solution 3 – Continuous Net-Update Automation
A agentic workflow currently under development that assists and automate updates to existing documentation in response to ongoing product changes, closing the loop on the full documentation lifecycle.

Case Study #1 – One-Time Corpus Refactor

Despite the company's advanced technical documentation platform and architecture, a combination of legacy content that preceded it, coupled with poor information architecture and writing practices resulted in below best-in-class content navigation, organization, findability, and discoverability as those tasks were solely the domain of human authors and governance.

Objectives and Success Criteria

The primary objective of the refactoring workflow was to achieve *best-in-class technical documentation*, defined across four dimensions:

- Accuracy
- Completeness
- Ease of use
- Ease of access

Direct business outcomes (KPIs) targeted by the initiative included improved customer satisfaction, faster onboarding, reduced support costs, increased customer retention, accelerated time-to-value, and long-term operational efficiency.

Findings from Corpus Analysis

A comprehensive analysis of documentation libraries revealed three primary improvement areas:

1. **Content Access**
   Documentation across the entire knowledge portal was difficult to navigate due to

deep hierarchies and inconsistent organization, resulting in poor content discoverability.

2. **Content Quality**
   While technically accurate, content required refinement to improve clarity, concision, consistency, and customer focus. The problem was not correctness, but usability.

3. **Multimedia Enablement**
   Existing documentation lacked sufficient visual and experiential assets. The initiative added "show-me" demo videos and other multimedia elements directly into technical topics.

A critical conclusion from the analysis was that technical accuracy of the existing content was already adequate. The challenge lay in structure, presentation, and semantic usability, not factual correctness.

Constraints and Risks

Refactoring an entire technical documentation corpus programmatically in a single operation is rarely attempted due to extreme risk and danger of data corruption. At the outset of the initiative:

- No known organization in the industry had publicly reported success in performing corpus-wide mass refactoring of every document in a large corpus using agentic AI at scale - and rarely, if ever, using traditional non-AI processes.
- Failure modes included content corruption, semantic data loss, broken dependencies, invalid reuse structures, and irreversible SEO damage.

Compounding these risks was the existing Docs-as-Code architecture.

Docs-as-Code Complexity

The documentation supply chain was built on a highly structured Docs-as-Code model using XML-based DITA encoding. The system includes:

- XML DITA topics
- Advanced semantic tagging
- Extensive reuse models
- Deep cross-document dependencies
- CMS versioning metadata

This content behaves as executable infrastructure rather than prose. Any automation incapable of preserving this structure would fail catastrophically.

DITA XML Preservation Was Non-Negotiable

DITA XML remains the industry standard for machine-driven documentation automation because it is:

- Highly structured
- Semantically explicit
- Programmatically and reliably manipulable

While presentation-centric formats optimize for human authoring convenience, they collapse under scalable automation. This initiative reinforces a central premise: *AI-ready documentation systems require and benefit from more semantic structure, not less.*

Architectural Evolution and Course Correction

Initial Approach

The original architecture attempted to refactor documentation by:

1. Scraping the public Knowledge Center
2. Converting HTML to Markdown
3. Applying AI-driven restructuring
4. Scoring and publishing results

This approach failed for a fundamental reason: the public site represents *compiled output*, not the authoritative source XML content. The result was irreversible semantic loss and an unstable pipeline.

Final Architecture

The team pivoted to refactoring *source content directly*:

- XML/DITA → XML/DITA
- Zero semantic loss
- Full preservation of reuse, tagging, and dependencies

This decision enabled deterministic transformations, predictable outcomes, and long-term AI-ready extensible content.

System Architecture

The agentic refactoring workflow is built primarily on:

- n8n for orchestration
- Python for transformation logic

The system is stateless, meaning, the workflow, comprised of multiple agents plus deterministic code, is fully automated end-to-end and requires no human intervention once initiated to refactor >10,000 documents at a time.

Implementation and Execution

To meet the mandated delivery deadline in under two months, teams executed intensively throughout the design, development, and test windows. Execution highlights included:

- Iterative design and validation cycles
- Daily scrums and continuous testing
- Refinement of scope and scoring criteria
- Enforcement of a minimum content quality score of 80
- Large-scale integration of multimedia assets

Fewer than 5% of topics required manual remediation, validating the effectiveness of the agentic approach. The key to success wasn't the technology; it was the extraordinary and intense teaming of professional documentation experts with the engineering team who also had depth of understand of the discipline.

Workflow Phases and Responsibilities

The refactoring workflow consists of four primary phases:

1. **Ingestion**
   Source content was pulled directly from the component content management system (our cCMS, IXIASOFT) as DITA XML and stored in n8n data tables for processing.

2. **Scoring and Rewriting**
   Content was scored using the Acrolinx, running Acrolinx via its API against company-specific style guidelines. Topics scoring below 80 were automatically rewritten and re-scored until thresholds were met or escalation criteria triggered human review.

3. **Structural Refactoring**
   Table-of-contents hierarchies were flattened and optimized to a maximum depth of four layers. Python-based workflows rebuilt DITA maps while preserving all identifiers, tags, and dependencies.

4. **Publication**
   Refactored topics and maps were published back into the CMS and published to the end-point delivery channels, including the product Knowledge Center portal through validated API pipelines with version control safeguards, error handling, and staged reviews.

Outcomes and Significance

This initiative demonstrates that agentic AI systems, when built on structured content, deterministic transformations, and deep domain modeling, can safely automate mission-critical documentation processes at scale.

More importantly, it illustrates a broader principle: *agentic AI is not about replacing writers with models, but about engineering systems that operationalize expertise*. The success of this system was driven not by larger models or longer prompts, but by disciplined structure, governance, and process fidelity.

The future of technical documentation automation is not hypothetical. It is already in production.

Case Study #2 – Net-New Agentic AI Product Documentation Automation

Where the one-time corpus refactor agentic workflow addressed historical debt, the net-new agentic AI product documentation automation workflow establishes a fundamentally new operating model for how technical documentation is created going forward. Rather than correcting legacy artifacts after the fact, this initiative embeds documentation generation directly into the software development lifecycle (SDLC), transforming documentation from a downstream, labor-intensive activity into a continuous, intelligence-driven system.

Problem Context

Traditional technical documentation workflows position documentation as an endpoint activity - initiated late in the SDLC, dependent on handoffs, and constrained by incomplete or degraded source knowledge. This model scales poorly, introduces latency into product launches, and places disproportionate pressure on specialized technical writer roles to reconstruct intent after design and implementation decisions have already been made.

As the product portfolio expanded in both scope and velocity, this downstream documentation model became increasingly misaligned with modern development practices. The challenge was not simply speed, but fidelity: authoritative product knowledge already existed early in the lifecycle, but it was fragmented across tools, formats, and teams.

Strategic Objective

The primary objective of the net-new agentic documentation workflow is to *highly automate and accelerate the creation of best-in-class technical documentation for newly developed products, features, and services*, shifting documentation activities to the right within the SDLC. This shift:

- Transfers ownership of technical knowledge creation closer to product teams
- Reduces dependency on dedicated first-draft technical writer labor
- Captures authoritative knowledge at its point of origin
- Reinvents the documentation content supply chain as a continuous system rather than a series of manual interventions

Rather than retrofitting content after release, the workflow establishes documentation as a first-class, automated capability that evolves alongside the product itself.

Source Knowledge Capture

The net-new workflow is designed to ingest and synthesize authoritative knowledge artifacts generated early in product conception and design. Primary inputs include:

- PRFAQs, when available
- Design assets such as Figma files
- Recorded product walkthroughs and demos
- Agent-generated UI models
- Product source code
- Design and technical documents across multiple repositories and formats

This is not an exhaustive or definitive list. These artifacts are intended to serve as structured inputs to the agentic system, enabling documentation to be generated from source truth rather than assembled after the fact. That said, this is a safe space to acknowledge, quietly or out loud, that reconstruction is how documentation so often happens. Many of us have been there. When design documentation is thin, outdated, or simply nonexistent (as it frequently is), technical writers fall back on the practices that keep the work moving forward: asking honest questions, reading between the lines, and piecing together the narrative as best they can. No shame, no judgment - just a clear-eyed recognition of reality, and a better path forward.

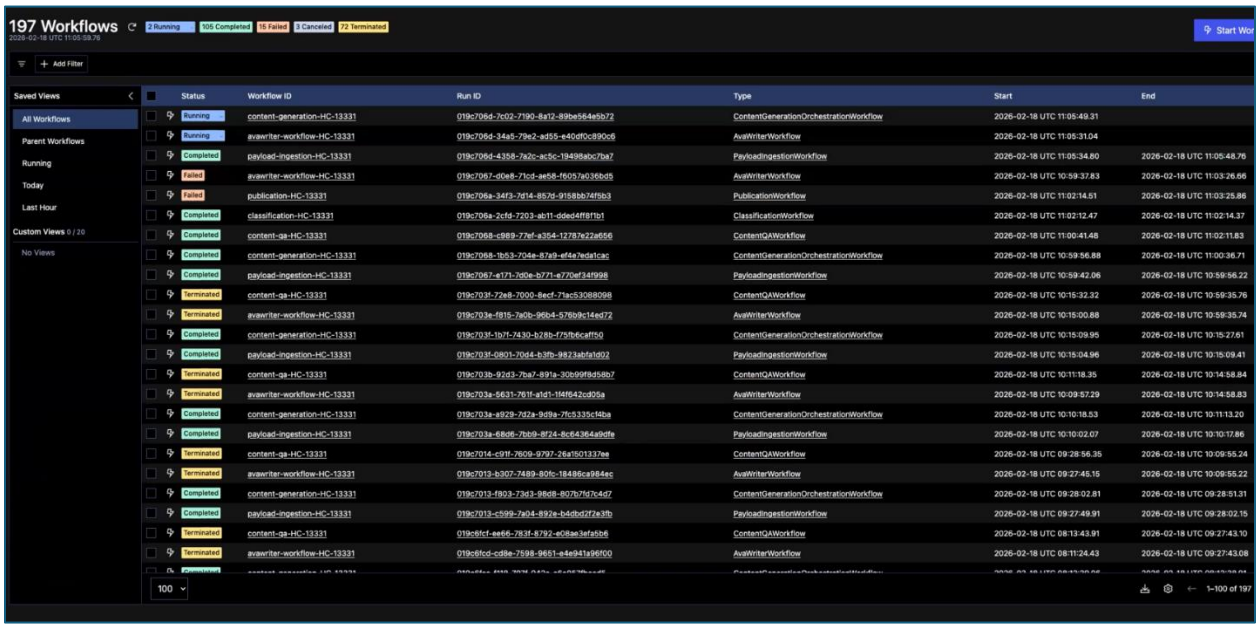


Figure 1. Individual agents that comprise the overall agentive workflow.

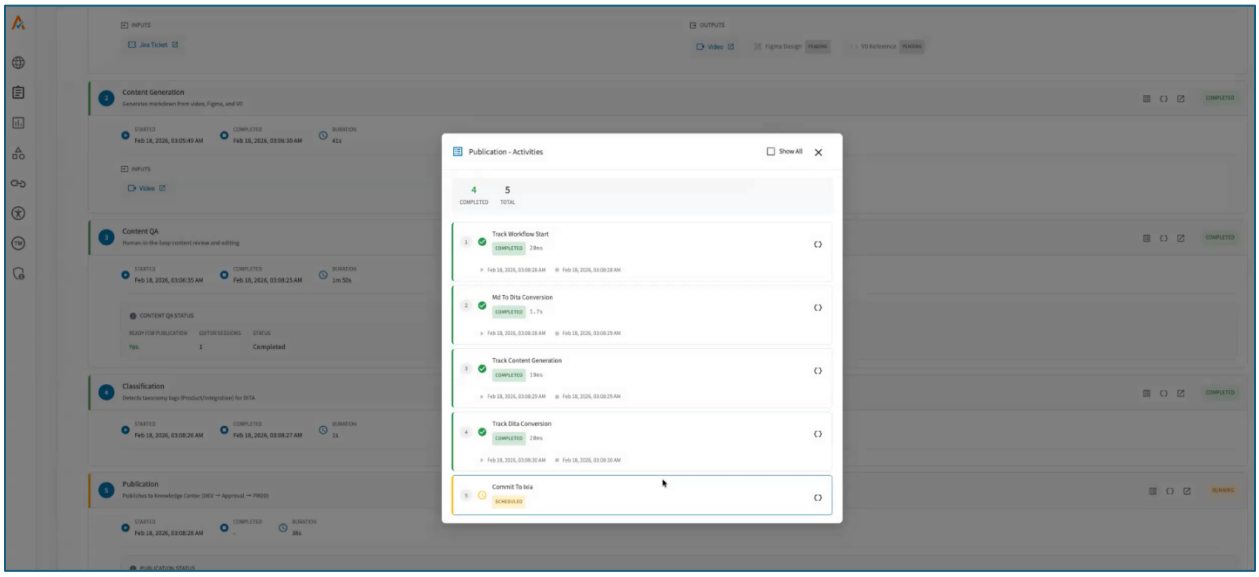


Figure 2. An active workflow during execution

End-to-End Workflow Design

The net-new documentation system operates as a *stateful, agentic workflow*, enabling both automation and controlled human intervention.

1. **Workflow Initiation**
   The process is initiated via a standardized Jira request containing or linking to all required knowledge assets. Future iterations automate this trigger directly from PRFAQs and agents, further reducing manual coordination.

2. **Automated Draft Generation**
   Agentic workflows ingest the supplied knowledge assets and generate initial technical documentation drafts. Output ranges from individual articles to fully structured manuals composed of dozens or hundreds of interlinked topics.

3. **AI-Driven Quality Optimization**
   Specialized agents apply computational linguistics and content scoring to evaluate quality against defined standards, optimizing clarity, completeness, and usability.

4. **Stateful Human Review and Refinement**
   Product development owners, engineers, SMEs, and reviewers interact with the system through a controlled interface, regenerating sections, refining content, and iterating until publishable quality is achieved. The choice of who pilots the workflow, whether that be a highly-skilled technical documentation professional or product developers, is an organizational and business choice.

5. **Exception Handling**
   For non-standard or complex scenarios, human operators can temporarily exit the agentic workflow to leverage external tools, then re-enter the workflow system without losing state or context. Since this is a stateful solution, human intervention can occur directly in the main workflow, and in more complex cases, the workflow can exit, permit highly-skilled technical documentation professionals to use advanced tools and systems (such as powerful authoring and content management systems and tools), then return and resume the agentic workflow to completion and publish.

6. **Automated Publishing and Distribution**
   Approved content is automatically stored in the cCMS and published to development and production environments, including the primary help knowledge

center, portal, in-product help, support and developer portals, knowledge graphs, and conversational AI systems to name a few as end-point delivery channels.

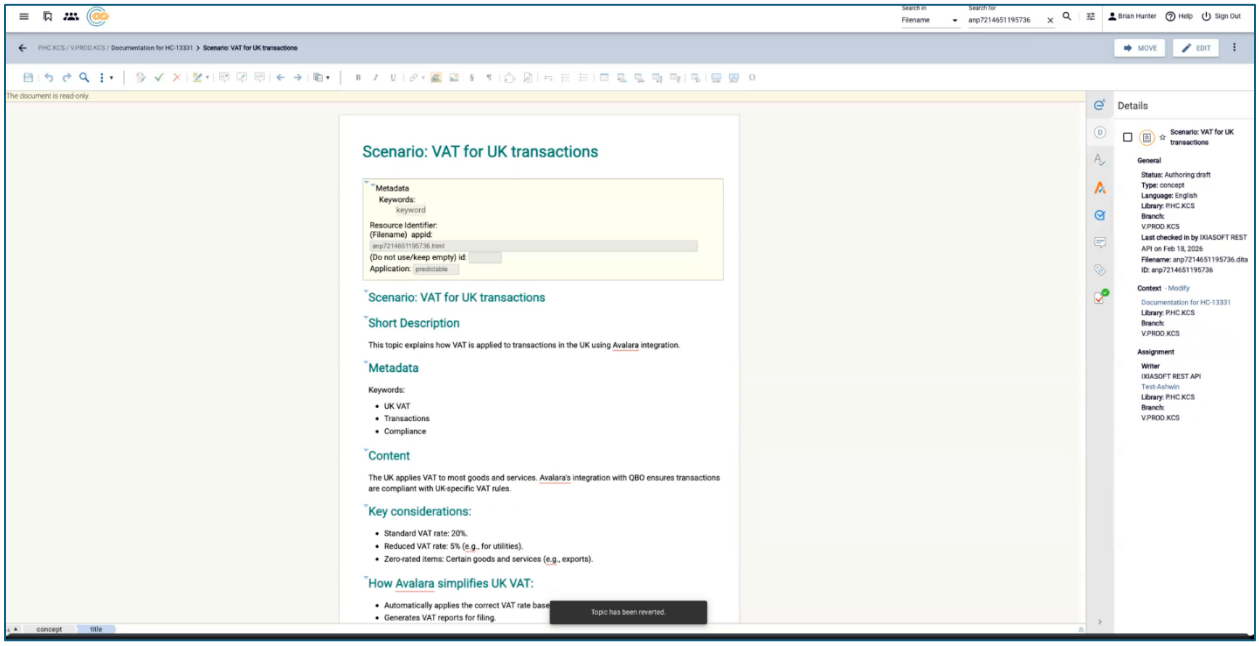


Figure 3. Document automatically converted to DITA XML and stored in the CMS for publishing.

Governance, Analytics, and Lifecycle Management

Operational visibility and long-term governance are provided through a centralized operations management center and portal, which serves as the system of record for documentation production, analytics, and lifecycle management. The operations management portal, much like an interactive type of Mission Control for documentation planning and management enables teams to:

- Creating and maintaining a living, actively continuously record of every document
- Monitor documentation performance over time
- Track content usage and effectiveness
- Support future agentic workflows for targeted updates and maintenance

This persistent *system of record* (SoR) is foundational for transitioning from net-new generation to continuous accuracy management.

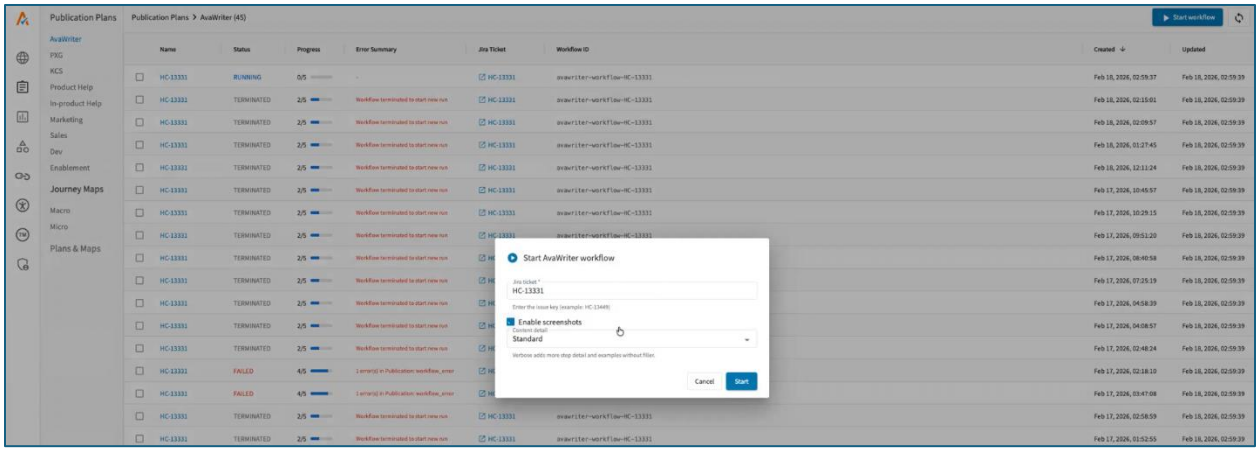


Figure 4. A central operations management center

Architecture

The net-new agentic workflow is built on a modern, stateful architecture:

- **Temporal** for agentic orchestration and workflow state management. Other options include stateful agentic architecture including LangGraph. CrewAI, and others.
- **Python** for transformation logic and AI integration

Agentic workflows are integrated with the operations management center, which provides the user interface, governance controls, and analytics layer. The operations management center itself is built on an Angular frontend, Node Express backend, and PostgreSQL database. This architecture supports long-running workflows, human-in-the-loop interactions, integration with content plans, content analytics and KPIs constructed from multiple sources, content journey maps, other agentive workflows, and future extensibility.

Implementation and Delivery

A proof of concept was initiated and completed within weeks, demonstrating multi-topic content generation from product demo videos. Following validation, production development work quickly followed.

Development of the net-new workflow continued through an intense two-month schedule under aggressive delivery constraints. PoC functionality was migrated into true agents orchestrated by Temporal, and the system was hardened for production use. While more advanced sub-agents and infrastructure work were deferred due to resource prioritization on the refactor project, a functional end-to-end solution - from asset ingestion through CCMS and Knowledge Center deployment - was delivered quickly and on schedule.

Outcomes and Significance

The Net-New Agentic AI Product Documentation Automation efforts represents a decisive

break from legacy documentation models. By embedding documentation creation directly into the SDLC and grounding it in authoritative source artifacts, the system reduces latency, improves fidelity, and establishes a scalable foundation for long-term knowledge delivery.

More broadly, this case study reinforces a central conclusion of this paper: *the real value of AI in technical documentation lies not in isolated tools, but in agentic process automation grounded in structure, governance, and system-level design*.

Foundational Agents

Foundational agents form the core of the net-new agentic documentation workflow. These agents are stateful, governed, and domain-aware, and they replace traditional manual documentation processes with executable, automated systems.

1. Workflow Orchestration Agent

Acts as the controlling meta-agent for the system. Maintains workflow state, coordinates agent execution, handles retries and escalation, enforces policy, and balances automation with human oversight.

2. Workflow Initiation and Intake Agent

Transforms structured intake requests into validated workflow payloads. Verifies completeness of inputs, establishes the system-of-record instance, and prevents downstream generation on incomplete or low-confidence inputs.

3. Content-Type-Specific Ingestion Agents

Each content modality is ingested by a specialized agent to preserve structure and intent.

3a. Document Ingestion Agent

Parses PRFAQs, specifications, and design documents. Preserves hierarchy, references, tables, and provenance.

3b. Video Extraction Agent

Transcribes demo recordings, extracts screenshots, and aligns narration with UI state changes.

3c. Design Asset Ingestion Agent

Processes Figma and UX assets to extract UI structure, flows, labels, and user intent.

3d. Code and API Ingestion Agent

Extracts APIs, schemas, parameters, defaults, and test-based examples. Flags knowledge gaps that require SME input.

4. SME Knowledge Capture Agent

Automates structured interviews with subject matter experts. Captures tacit knowledge, summarizes insights, and feeds structured results back into the workflow payload.

5. Input Validation and Payload QA Agent

Performs pre-generation validation. Ensures clarity, terminology alignment, persona coverage, and task completeness. Blocks generation when confidence thresholds are not met.

6. Knowledge Graph Alignment and Reuse Agent

Retrieves and aligns reusable components from the knowledge graph. Ensures semantic consistency, prevents duplication, and supplies authoritative references to downstream agents.

7. Core Generation Agent

Transforms validated knowledge into documentation outputs. Composed of draft generation, style and tone alignment, persona formatting, and governance validation sub-agents.

8. Media Enrichment Agent

Identifies where visual assets are required and generates or integrates diagrams, screenshots, and videos to improve usability and comprehension.

9. Automated Editorial QA Agent

Applies editorial quality checks including clarity, style, inclusivity, and SEO. Scores content quality and initiates regeneration loops when thresholds are not met.

10. Technical Accuracy Review Agent

Manages SME review only when required. Flags low-confidence assertions, summarizes review requirements, and tracks acceptance or correction.

11. Content Management and DITA Preservation Agent

Converts outputs into valid DITA XML, preserves identifiers and reuse structures, manages versioning, and stores content safely in the component CMS.

12. Publishing and Distribution Agent

Automates delivery across portals, in-product help, APIs, and conversational systems. Manages staged releases and rollback.

13. Analytics and Observability Agent

Tracks velocity, quality, reuse, and performance metrics. Feeds insights back into orchestration logic for continuous improvement.

14. Lifecycle and Knowledge Drift Agent

Monitors content freshness, detects drift from product reality, triggers update workflows, and prevents silent decay.

Future Agents

These agents described above represent core capabilities for an agentic documentation workflow. Some have yet to be fully matured and further tuned. The agents that follow below are candidates to build on the foundational system and emerge as automation maturity increases.

Knowledge and Semantics Layer

Ontology Management Agent, Taxonomy Auto-Classification Agent, Knowledge Graph Reasoning Agent.

Optimization and Intelligence

Prompt Optimization Agent, A/B Content Testing Agent, Personalization Agent, Intent Detection Agent, Docs-as-Tests agent.

Governance and Trust

Trust and Provenance Agent, Legal and Trademark Compliance Agent, Accessibility Compliance Agent.

Experience and Journey

Micro Journey Mapping Agent, Macro Journey Orchestration Agent, Scenario Simulation Agent.

Operations and Scale

Role Assignment Agent, Agent Safety and Failure Agent, AI Content Operations Agent.

This is not a finite inventory of potential agents. More are certain to emerge.

Case Study #3 – Net-Update Agentic AI Product Documentation Automation: - Where the Amateurs Wash Out

This is the part of the journey where enthusiasm meets physics - and loses.

Generating net-new, multi-topic user guides for brand-new products or major features is a legitimate and increasingly solvable use case. It's flashy. It demos well. It looks great on slides. But it's not where most of the real technical documentation work occurs.

The overwhelming majority of technical documentation effort isn't about creating something entirely new; that happens. It's most frequently about *changing something that already exists* - often something large, old, interconnected, and opinionated.

Now picture the moment of truth: a product change ships, and the documentation team receives the familiar Jira ticket. Vague title. Minimal context. Something like *"Update docs for new behavior."* No scope. No impact analysis. No indication of how far the ripple travels.

What happens next is not "update a topic."
It's a forensic investigation.

The change might touch a single paragraph, or it might require precise, surgical edits across multiple topics, multiple guides, and multiple product libraries that just happen to share concepts, APIs, or behaviors. And because offerings are almost never isolated - stacked products, layered services, bundled solutions - changes tend to cascade in ways that are only obvious *after* something breaks.

In human-driven processes, experienced documentation planners, information architects, and writers compensate through hard-earned intuition. They remember where the bodies are buried. They remember which guide quietly reuses that concept. They remember the one topic that says the same thing… differently.

And yes, if structured reuse was diligently implemented, this gets easier.
Yay, DITA. Gold star.

But let's be honest. Even in disciplined shops, not every writer complied. Not every reuse opportunity was recognized. Not every exception aged gracefully.

So, here's the real question, the one most AI conversations conveniently skip:

How does a system identify every library, map, topic, and element that needs to be updated - or at least reviewed- when a change occurs?

Let's start by clearing away the usual suspects.

Conventional search?
Viable only if you enjoy running dozens of queries, waiting minutes or hours, and still missing half the dependencies Even the same content is often written differently across multiple documents. Conventional search can't detect those

Large language models or custom GPTs?
No. There isn't a model alive with a context window large enough to load even a couple of complete user guides. For reference, I recently used LLMs to analyze my 50-chapter, 400-page fiction novel and had to load it ten short chapters at a time. Context degradation was obvious, and brutal. More tokens didn't help; they made it worse.

Vector databases?
Absolutely not. Once you shred a corpus into equal-sized chunks, you've discarded the very things that matter: structure, identity, hierarchy, explicit relationships, reuse, metadata; everything required to trace changes back to authoritative source topics. You can try to reconstruct meaning after retrieval, but at that point you're playing semantic Whac-A-Mole.

So no.
Search won't save you.
Models won't save you.
Vectors won't save you.

This is where the professionals reach for the only tool class that actually survives contact with reality: **ontologies and knowledge graphs** are needed to do the heavy lifting. Vector models can be employed to inform, but they are utterly incapable of being the source of ground truth.

A documentation corpus generated from highly structured source content enables fast, full-context queries while preserving every original relationship, identifier, dependency, and semantic signal. This is precisely the problem space that knowledge graphs were designed to solve, and why highly structured componentized object-oriented document formats such as XML remains unmatched for building, maintaining, and exploiting documentation knowledge graphs at scale.

It's also why we developed the *Document Object Model (DOM) Graph RAG* open model: to enable surgical, deterministic operations over large documentation systems without collapsing context or losing provenance.

There isn't room here to unpack the full DOM graph model; that's what the linked paper is for. What matters for this discussion is the takeaway: If you want agentic AI to make precise, safe updates across a living documentation system, you need structure that remembers what humans forget - and preserves what probabilistic systems destroy.

Everything else is just vibes and retries.

Think you have better mousetrap? I'm all mouse ears.

Human in the Loop for Net-Update Agentive Workflow

Finding and retrieving the correct topic files is table stakes. Once you have the correct subset, then the workflow has to identify and process the specific sub-elements in those topics. Good luck with non-DOM formats – they lack componentization and element containment – clearly delineated boundaries of what to target and modify. Take a task topic as a classic example. We know the major component of such topics:

- **Title** – What you're doing
- **Shortdesc** – Why you care
- **Prereq/Context** – Are you ready?
- **Steps** – Do the thing
- **Result/Postreq** – What happens next

These are the minimum components; there are many more required and optional ones depending on one's specific content model.

How does the system know where the boundaries are without the kind of containment provided in formats such as XML, JSON, and other DOM formats? How does the solutions "know" a preceding lead-in paragraph is a dependency with the steps and not to be excluded from update? They can't unless humans diligently and consistently annotate the source, which isn't enforced programmatically and error-prone. This is the reality that those who argue for author-friendly, simplistic presentation-oriented formats ignore to their strategic demise. Neither deterministic code nor AI can reliably tell. It is rank ignorance by otherwise highly intelligent developers.

The DIFFerence Between Net-New and Net-Update Agentic Workflows

With net-new agentic processes we're not concerned with the high-risk of making fully automated changes using AI. With net-update agentic processes we must still compensate for incorrect identification and hallucinating. The written language is a squirrely beast.

A stateful workflow is required to provide the workflow operator the opportunity to verify the proposed changes made by the LLM. For this one option is to provide an intermediate authoring step in the workflow where the before and after content is presented (DIFFing) to the human workflow operator and afforded the opportunity to accept, reject, modify, or otherwise disposition the suggested changes for optional downstream intervention in exception cases.

And let's not ignore the stark reality of product development: No matter how intent SDLC process are or may become insisting design be locked down at the design phase, testing, and use, result in many changes, additions and deletions late in the cycle – sometime right up to deployment and of course, afterwards too. Net-update workflows need to be designed to accommodate the unexpected.

So, What's on the Near Horizon?

The need for semantically rich, deterministic AI governance is not news to the cohort of content and knowledge management professionals worldwide. Some of us have been screaming from the rooftops to deaf developer ears about it since the turn of the decade.

Only now are development leaders asking about ontologies and knowledge graphs. It is about time! The chorus is slowly getting louder by the day and building toward an eventual crescendo that will become part of the permanent AI stack, not a passing fancy.

Let us examine a few approaches now in their early and emergent stages.

Context Graphs for Agentic Decision-Making and Governance

There is a great deal of excitement about ontologies and context graphs lately. Let us unpack what they are and how they are used in practical applications, especially in agentic AI solutions.

Context graphs are not only post-hoc explanatory artifacts. They are also powerful mechanisms for the automated making of decisions in live, dynamic processes. When used inside agentic systems, they provide both concrete and abstract frameworks for dynamic governance in AI, whether the system is a fully autonomous agent or an agentic workflow composed of multiple tools and decision points.

Over the past year, I referred to these structures as "process" or "governance" graphs for lack of a stable, shared term. "Context graph" captures the idea just as well, and the emergence of a standard term is valuable because it gives the field a common conceptual anchor.

Context graphs in agentic systems are not primarily about "things." They are about process.

When building a context graph to drive decision-making, entities can, and often should, represent tasks, states, goals, and constraints, not just real-world objects. For example:

- Entity X → a task or state

- Entity Y → another task or state
- Relationship → X must precede Y

This does not hard-code a workflow. Instead, it encodes constraints that the agent can reason over. The agent remains free to plan, replan, recover, or parallelize actions as conditions change.

The core design principle is simple:

Express what must be true, not what must be done next. If an agent may need to reason about something, inspect it, adapt around it, or justify it, it belongs in the context graph.

In this sense, agentic systems work best when the graph captures semantics and constraints, not scripts.

In an agentic process, a context graph is not merely a knowledge graph of static facts. It is closer to a combined model of situational state, constraints, and affordances that the agent uses to decide what to do next.

Because of that, entities in the graph can represent:

- States
- Tasks
- Goals
- Sub-processes
- Preconditions
- Decisions
- Resources
- Constraints
- Policies or rules

Process abstractions are first-class citizens, not second-class hacks. As a concrete example, a contextual ontology manifested as an operational knowledge graph might include process entities such as:

- Validate_Input
- Acquire_Permissions
- Execute_Action
- Confirm_Result
- Goal_Achieved
- Blocked_State

Edges between these entities can encode relationships such as:

- must_precede
- requires
- enables
- blocks
- invalidates
- satisfies
- violates_policy

A relationship like “Y is invalid unless X has occurred” is declarative, not procedural. It defines legality, not sequence. This distinction matters. The graph defines constraints.

The agent decides behavior

SHACL integrates naturally into this model by providing explicit, machine-checkable constraints over the context graph, including structure, dependencies, ordering rules, and policy guardrails.

What SHACL does not do is decision-making. It does not plan, sequence actions, or choose what to do next. Its role is to ensure the graph never enters an illegal or inconsistent state.

The right mental model is this:

SHACL is the constitution of an agentic system; it defines what must never be violated. The agent, planner or reasoner, is the government. It decides what happens next.

Used together, context graphs and SHACL provide strong guardrails while preserving agent flexibility. The result is an agentic system that is inspectable, adaptable, and governed by explicit semantics rather than brittle control flow.

Context Graph Layers in Practical Use: Rules Layer vs Audit Layer

There is frequent confusion about whether context graphs represent rules or the memory of how rules were executed. They are both, and they must be separated into layers to be useful/ Context graphs are not just about remembering decisions. They are about governing them.

Layer 1: Rules (what should be true)

This layer captures intent. It includes policies, constraints, procedures, ordering rules, and guardrails. It defines what is allowed, required, or forbidden under normal circumstances.

This layer is intentionally small, stable, and deliberate. Think of it as the constitution of the system: human-approved rules that define how decisions are supposed to work.

Layer 2: Records (what actually happened)

This layer captures reality after execution for auditing and for adjusting Layer 1. Each time a decision is made, an action is taken, or a rule is overridden, the system records it. This layer is append-only and factual. It preserves what happened and why it was reasonable at the time.

Rules define intent. Records show reality. Audits decide whether intent should change.

Smarter Knowledge Graphs: From Labels to Semantic Layers

As organizations adopt AI-driven documentation and agentic workflows, it is no longer enough to rely on simple labels or tags applied to content. In traditional systems, labels functioned much like sticky notes. They helped humans find things, but they did not explain meaning to machines. This distinction becomes critical when AI systems are expected to reason, automate decisions, or safely operate at scale.

The article by Elsa Sklavounou, When Labels Become Semantic Layers highlights a key shift: labels stop being decorative metadata and become part of a deeper semantic layer. Instead of saying only what something is called, semantic layers describe what something means, how it relates to other concepts, and under what conditions it is valid or authoritative. For humans, this feels subtle. For machines, it is the difference between guessing and knowing.

In the context of technical documentation, this shift directly supports this paper's central argument that precision, trust, and automation depend on explicit meaning. A label such as "API" or "Configuration" is useful for navigation, but it does not tell an agent whether the content is normative, optional, deprecated, or version-bound. Semantic layers, expressed through taxonomies, ontologies, and context graphs, make those distinctions explicit and enforceable.

This is why semantic structure must be treated as infrastructure rather than editorial polish. When labels evolve into semantic layers, they become part of the system's decision logic. Agents can validate assumptions, detect conflicts, respect constraints, and explain why an action was taken. Without this layer, AI systems may appear confident while operating on incomplete or incorrect assumptions.

In short, labels help humans browse. Semantic layers allow AI systems to operate safely. As agentic documentation workflows move from experimentation to production, this transition is no longer optional. It is a prerequisite for scalable, trustworthy automation.

Context, In Context

You can't scroll LinkedIn, Medium.com, or many other useful sites without hearing the term "context". Unfortunately, definitions are all over the map.
Why? Well because it requires... context!

Let's examine the notion in the context of technical documentation and AI.

The term "context" has rapidly become another ingredient in the modern AI word salad. It gets tossed around freely alongside "agents," "reasoning," and "semantics," yet is rarely defined with any precision. In practice, "context" is often reduced to whatever text happens to fit into a prompt window or a vector chunk. That is not context. That is proximity.

Proper context is structured, governed, referential, and computationally meaningful. And this is where DITA enters the conversation, not as a silver bullet on its own, but as the most effective high-structure document format ever designed for creating real, durable context when combined with taxonomy, schema, ontology, and knowledge graphs.

To be clear at the outset: DITA alone is not context. DITA plus domain taxonomy, semantic metadata, ontology, and graph-based knowledge representation is context.

What DITA Actually Contributes

DITA provides something most AI pipelines quietly lack: an explicit, enforced semantic structure. Its elements are not arbitrary containers but descriptive semantic constructs such as concept, task, reference, step, prerequisite, result, and condition. These element names already encode meaning before a single model is invoked.

Unlike flat documents or markdown blobs, DITA structures content into stable, typed objects. Those objects persist across systems, versions, and delivery channels. This is the foundation upon which everything else becomes possible.

Semantic Metadata as First-Class Context

DITA topics and elements can carry rich semantic metadata: audience, product, version, role, region, lifecycle state, applicability, and domain-specific attributes. This metadata is not decorative. It is machine-addressable, queryable, and deterministic.

When taxonomy is applied to that metadata, content is no longer merely labeled. It is classified within a governed semantic system. Taxonomy labels introduce shared meaning

across documents, domains, and time. They allow systems to answer questions like “what kind of thing is this?” without guessing.

Ontology and Knowledge Graphs Turn Structure into Context

When DITA structures and taxonomies are aligned to an ontology, the result is not better documentation. It is a knowledge model.

DITA topics and elements map naturally to graph nodes and relationships. Tasks reference concepts. Concepts constrain references. Versions supersede prior versions. Products depend on components. Preconditions gate procedures. These relationships exist implicitly in the documents and explicitly in the ontology.

Persisted in an RDF or property graph database, this content becomes a queryable knowledge graph that preserves both explicit and implicit document relationships. Context is no longer inferred statistically. It is retrieved deterministically.

Why Graph-Based Retrieval Matters

Graph databases allow object-oriented retrieval of content that respects structure, meaning, and relationship. This is fundamentally different from vector-based retrieval.

Vector RAG retrieves text based on similarity. Graph-based retrieval retrieves knowledge based on meaning, role, and relationship. It preserves the integrity of the document model instead of shredding it into chunks.

With DITA-based graphs:

- Content identity is deterministic.
- Relationships are preserved end-to-end.
- Provenance, versioning, and applicability remain intact.
- Retrieval is explainable

Vector chunking, by contrast, discards structure, breaks semantic boundaries, and reassembles context probabilistically. It works until it doesn’t, and when it fails, it fails silently.

Deterministic Identification and Reasoning

DITA-based content objects have stable identifiers. Those identifiers survive storage, transformation, delivery, and reuse. This allows precise citation, traceability, and reasoning.

When combined with RDF graph databases, this enables real inference. Not simulated reasoning inside an LLM, but actual rule-based and description-logic inference using

established engines. Systems can infer applicability, compliance, dependency, and impact without token-burning guesswork.

Advanced NER without Guessing

Named Entity Recognition becomes dramatically more accurate when applied to semantically structured document objects enriched with metadata and taxonomy. Entities are not extracted from raw text alone. They are resolved using:

- Semantic element roles
- Taxonomic classifications
- Ontological relationships

This produces higher precision, lower ambiguity, and far better grounding for downstream AI systems.

Cost and Performance Advantages

A knowledge graph processes context without calling an LLM. Queries, inference, filtering, and traversal happen outside the model. The LLM is invoked only when language generation is actually needed.

This dramatically reduces token consumption, latency, and cost. Instead of flooding models with entire documents or massive chunks, systems retrieve exactly the context required, already structured and validated.

This is not just cheaper. It is architecturally sound.

Why LLM-Only and Vector-Only Approaches Fall Short

Direct LLM usage treats context as text. Vector RAG treats context as similarity. Neither treats context as meaning.

They lack:

- Deterministic structure
- Governed semantics
- Persistent identity
- Explicit relationships
- Formal inference

They approximate context statistically. DITA-based semantic systems model it explicitly.

So, What Is Context?

Context is not more text. It is not larger prompts. It is not higher-dimensional embeddings.

Context is structured meaning, governed classification, explicit relationships, and computable knowledge.

DITA, when combined with taxonomy, schema, ontology, and knowledge graphs, delivers exactly that.

Call it unfashionable if you like. But unlike word salad, it actually works.

It's Still About the Semantic Stack

Agentic AI systems do not fail because models are insufficient. They fail because meaning is insufficiently defined, governed, and operationalized. As organizations rush to deploy agents and agentic workflows, a recurring pattern emerges: probabilistic systems are asked to compensate for the absence of a formal semantic foundation. This is not a tooling problem. It is a semantic maturity problem.

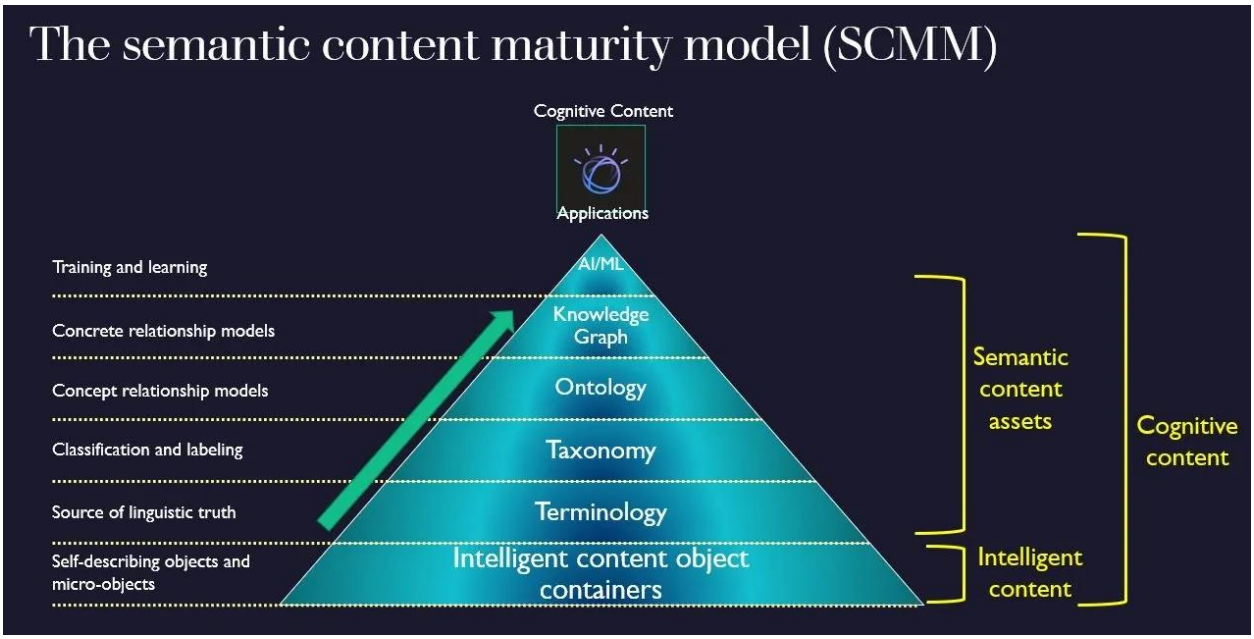
The semantic stack - taxonomy, ontology, and knowledge graphs - provides the structural backbone required for agentic AI to operate safely, deterministically, and at scale. Taxonomies establish controlled classification and shared vocabulary. Ontologies formalize meaning, constraints, and relationships. Knowledge graphs persist these semantics as executable infrastructure that agents can query, reason over, and validate against.

Without this stack, agentic workflows are forced to rely on statistical similarity, oversized context windows, and recursive prompting strategies. These approaches create the illusion of intelligence while amplifying error, cost, and governance risk. In contrast, semantic systems reduce model load, constrain behavior, and enable deterministic retrieval and inference outside the language model itself.

This mirrors the progression described in my Semantic Content Maturity Model. Note the date of the model; it's another "*I told you so*." to the AI industry.

Organizations evolve from unmanaged content, to structured content, to semantically governed knowledge capable of supporting automation and reasoning. Agentic AI represents the final stage of that maturity curve, but it cannot be reached by skipping the foundational layers.

Agentic systems require more than access to content. They require an explicit understanding of what content means, how it relates, what constraints apply, and which sources are authoritative. That understanding does not emerge from scale alone. It must be engineered.



Michael Iantosca © 3/23/2002

Figure 5: The Semantic Stack as the Foundation for Agentic AI

Major Takeaways (a.k.a. Things We Should've Realized by Now)

Agentic AI is not a "someday" concept; it is already here, quietly unpacking its bags and asking for Wi-Fi. Yes, the agentic documentation workflows being designed, built, deployed, and put into production today are, let us call them, primitive. But that is fine. This is the Model T phase, not the Tesla Autopilot era. Better models, smarter agents, and less duct tape are coming, whether we are ready or not.

Right now, most technical documentation teams are still dabbling with individual AI tools, cute experiments, really. That is child's play compared to what is next. The real blocker is not imagination; it is resources. Most organizations simply do not have the internal development muscle to build end-to-end agentic systems, because this cannot be bought off the shelf. We are talking about automating the entire content supply chain, not sprinkling AI fairy dust on a single tool in isolation. And let us be honest: most documentation vendors only see their own narrow slice of the ecosystem.

Papers like this should arm documentation leaders with the arguments they need to go to leadership and say, "We need real investment, not another pilot."

One important point not emphasized earlier in this paper is that the agentic workflows described are intentionally designed as orchestrated and largely linear models. That is not the only viable approach. I am keenly aware of, and fascinated by, peers who are pursuing a different path, creating collections of non-linear autonomous agents to solve the same classes of problems. Neither approach is inherently wrong; each has strengths and trade-offs. For example, with fully autonomous agents, accountability becomes a harder question when something goes awry.

Nevertheless, both approaches are compelling, and the right answer may be orchestrated, autonomous, or a hybrid combination, depending on the application. At this stage, we do not yet know which models will dominate. A useful analogy is that we are flying a fighter jet, building it at the same time, in the middle of a dogfight, while the ammunition is still being designed on the ground. AI technology stacks are evolving even as we build on top of them.

The smartest thing all of us in the professional technical documentation and adjacent communities can do is share, compare, and learn from one another.

Meanwhile, documentation tools and systems vendors need to wake up and choose violence, the productive kind. Their platforms still matter, even in an agentic world, but only if they collaborate or fundamentally evolve. Otherwise, they should enjoy their future as niche plugins. What is happening today is largely the bolting-on of taxonomy management systems and knowledge graph databases to legacy platforms. That approach has a shelf life.

At some point, a startup with no legacy baggage will build a component content management and delivery platform natively grounded in a universal content ontology, DITA-informed, RDF or OWL-based, or hybrid graph-driven, and designed for AI from day one, and it will fundamentally disrupt the current vendor landscape.

Which brings us to the uncomfortable truth.

Technical documentation professionals need to step up and claim ownership of next-generation agentic content workflows and the broader content supply chain. This is how the discipline expands its influence and relevance, not how it fades politely into obsolescence. If we do not, well-meaning but underprepared software teams will build brittle knowledge structures and accumulate content debt at massive scale.

Once that happens, the rest of us will be left with shovels, digging our way out for years.

.

Appendix A: New and Emerging Roles With Agentic AI

Knowledge Architect

Designs and governs the semantic model that defines what the organization knows.

Responsibilities include defining domain ontologies, concept boundaries, relationships, constraints, and authoritative sources. This role ensures that meaning is explicit, governed, and evolvable over time. Without this role, AI systems retrieve fluent but conceptually inconsistent or incorrect knowledge.

Documentation Platform Engineer

Builds and maintains the end to end documentation pipeline as production infrastructure.

This role integrates content management systems, knowledge graphs, agent orchestration frameworks, validation services, and delivery channels. The focus is deterministic transformation, observability, resilience, and scale. Without this role, AI capabilities remain brittle augmentations rather than reliable systems.

Agent Workflow Designer

Encodes human editorial judgment into executable agent behavior.

Responsibilities include decomposing documentation processes into agents, defining thresholds, escalation logic, validation loops, and stopping conditions. This role determines where autonomy is safe and where human intervention is required. Absent this role, agentic systems act unpredictably or destructively.

Semantic QA Lead

Owns correctness at the semantic level rather than surface quality.

This role ensures factual consistency, ontology alignment, concept reuse integrity, and semantic drift detection across all generated content. Semantic QA defines what correctness means in machine enforceable terms. Without it, systems slowly degrade while remaining superficially fluent.

Context Engineer

Controls how knowledge is retrieved, ranked, transformed, and delivered based on intent.

Responsibilities include designing hybrid retrieval strategies, enforcing authority and scope constraints, and shaping model ready context structures. This role prevents context overload and semantic contamination. Without it, more context reliably produces worse answers.

Ontology Product Manager

Owns the roadmap, scope, and tradeoffs of the organization's meaning model.

This role balances modeling rigor against delivery needs, manages breaking changes, and aligns stakeholders around shared semantics. Ontology is treated as a product, not an academic artifact. Without this role, ontologies stagnate or fragment.

AI Enablement Lead

Makes ontology driven and agentic systems usable and safe for non engineering teams.

Responsibilities include training, workflow design, permission models, and governance enforcement. This role prevents shadow AI practices while accelerating adoption. Without it, systems are either ignored or misused.

Knowledge Graph Engineer

Implements and optimizes the graph infrastructure that powers semantic retrieval and reasoning.

This role designs schemas, ingestion pipelines, query patterns, and performance optimizations. It ensures graphs operate reliably at enterprise scale. Without it, knowledge graphs remain proofs of concept.

Retrieval Architect

Designs how knowledge is found and assembled.

Responsibilities include balancing precision and recall, combining symbolic and probabilistic retrieval, and enforcing trust boundaries. This role eliminates blind vector chunking. Without it, retrieval either misses critical knowledge or floods the system with noise.

Content Systems Analyst

Measures semantic health and performance across the knowledge supply chain.

This role tracks coverage, reuse, drift, agent effectiveness, and lifecycle performance. Analytics provide feedback loops for continuous improvement. Without this role, degradation goes unnoticed until trust collapses.

Legacy Content Migration Lead

Transforms page based legacy content into structured, graph aligned knowledge.

Responsibilities include preserving identifiers, provenance, reuse models, and semantic integrity during large scale transformation. This role prevents legacy debt from poisoning modern systems.

Trust and Provenance Officer

Owns source attribution, versioning, auditability, and lineage of machine mediated knowledge.

This role enables explainability, compliance, and institutional trust. Without it, AI systems fail in regulated or mission critical environments.

Documentation Systems Architect

Designs the overall topology of the documentation ecosystem.

This role ensures authoring, agents, graphs, QA, analytics, and delivery systems compose into a coherent whole. Without it, organizations build isolated subsystems that do not scale together.

Process Modeler and Domain Modeler

Captures tacit domain knowledge and decision logic into formal representations.

This role extracts heuristics, exception handling, and micro decisions from subject matter experts. Without it, agentic systems encode incorrect assumptions.

Agent Safety and Failure Engineer

Designs safeguards against catastrophic automation failures.

Responsibilities include blast radius control, rollback strategies, edge case modeling, and failure testing. Without this role, errors propagate at machine speed.

AI Operations Lead for Content

Runs agentic documentation systems in production.

This role monitors workflows, manages long running agents, handles partial failures, and coordinates releases. Without it, operational reliability collapses.

Semantic Infrastructure Product Owner

Owns meaning as shared enterprise infrastructure beyond documentation.

This role aligns documentation, support, product, and analytics around a unified knowledge model. Without it, knowledge silos persist.

Documentation Data Engineer

Builds and maintains data pipelines that support documentation systems.

Responsibilities include normalization, storage optimization, observability, and metrics. Without this role, systems become opaque and fragile.

Human Review Strategist

Determines where and how humans intervene in agentic workflows.

This role defines escalation thresholds and prevents over humanization that destroys scale. Without it, either automation fails or humans become bottlenecks.