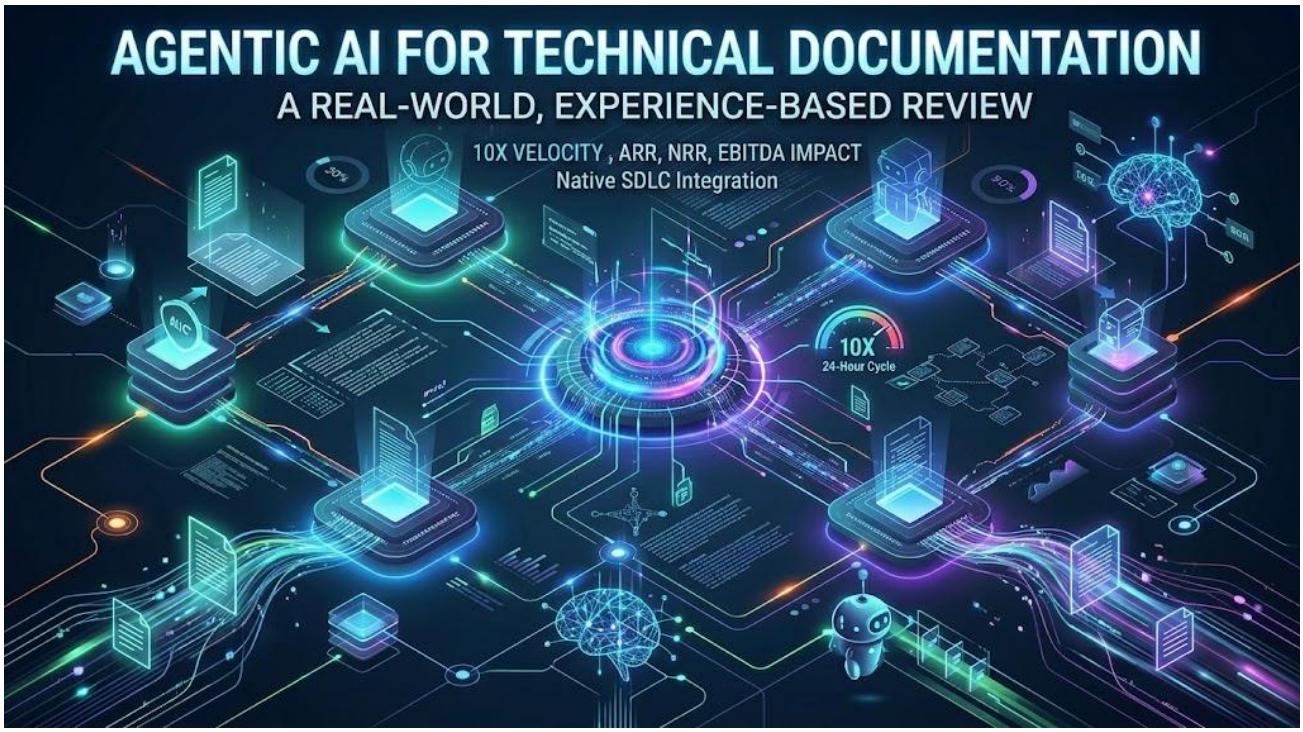
Agentic AI for Technical Documentation

A real-world, experience-based review

PART TWO:

Challenges, Approaches, and Solutions for Designing and Implementing Agentic Documentation

Michael Iantosca
Senior Director of Knowledge Platforms and Engineering
Avalara



Now that we understand the **why** of agentic documentation, let's dive into the **how** – and, perhaps more importantly, how **not** to approach this new era being ushered in by agentic AI for technical documentation.

Challenge #1: Proper Design

As junior technical writers, two of the most basic things we're taught are:

1. Know your audience.
2. Document your perplexities.

Ironically, the broad and early pressure to experiment with and "do AI" – whatever that is supposed to mean – causes teams to forget these basic principles.

Damn the torpedoes. Full speed ahead.

Big mistake.

As career documentation professionals, we know a lot. Too much, in fact.

If your organization documents its processes, those documents are more than likely high-level descriptions – the **Big Rocks**, as I like to call them. What they don't reflect are the dozens, and actually hundreds, of details of our processes and best practices that we learn over time and integrate into our memory, most of which are never documented.

When designing agentic AI solutions, that can be catastropic.

Those details are critically necessary to design successful agentic processes. Take them for granted and success will be elusive. At best, you'll create a great deal of unnecessary rework – assuming you're even given a second chance.

This activity requires the knowledge, both technical and institutional, of documentation professionals.

Worse, automated technical documentation can sound deceptively simple to a software engineer who is unfamiliar with the discipline and sees an opportunity for a quick automation win.

As I've said repeatedly, this is an endeavor documentation professionals must be relentlessly assertive about and must lead. Otherwise, Johnny Programmer, who slept at a Holiday Inn Express last night, is going to convince himself that access to an LLM suddenly makes him the Steven Spielberg of documentation products.

And allow me to address something I'm seeing far too often: individual writers or teams building fragmented, personally maintained automations – perhaps in n8n or Zapier, or as ad hoc agents around frontier models – without treating them as production infrastructure.

Many of these efforts are sincere and useful for learning or personal productivity. The problem begins when the organization becomes dependent on them without ownership, support, testing, or lifecycle management. I've lost count of the times a team has asked my documentation platform engineering team to take over a personally developed tool after its creator moved on. My answer is usually simple: that is not a sustainable operating model [expletives deleted].

In the long run, personally developed tools can fragment and undermine what needs to become a deliberately designed end-to-end agentic documentation supply chain – even if that supply chain is built and deployed incrementally.

How we approached agentic documentation design

My team had the foresight to bring together a wide range of relevant professionals in one location over several days to develop a comprehensive agentic design and strategy. The group included experienced documentation professionals, writers, editors, information architects, content strategists, technical support, user-experience design, engineering – both documentation platform and product engineering – localization, operations, terminology, taxonomy, and ontology expertise, just for starters. It was a big room with a lot of people.

We began with a strawman proposal and walked away with a comprehensive design and strategy after three days spanning people, process, and technology. It was an ambitious plan to be developed incrementally but holistically:

- Agentic documentation workflows ultimately consisting of dozens of agents – mostly stateful, managed workflows, but also stateless autonomous agents where appropriate.
- Combine agents and agentic workflows with traditional non-AI deterministic programming and knowledge management where appropriate.
- Develop several foundation agentic solutions that would quickly return measurable value:

1. A refactor agent to simplify existing documentation
2. A "Net-new" agentic workflow
3. A "Net-Update" agentic workflow
4. Agentic operations management
5. Other document type extensions

These initiatives also included automated integration with the software development lifecycle (SDLC). Let's take the implementation experience one project at a time and examine the issues we encountered and how we solved them.

Agentic Project #1: Documentation Refactor Agent (Stateless)

The first agent my team built was designed to refactor an entire documentation website.

Years of platform migrations had left the site increasingly difficult to navigate. Content had been repeatedly ported forward, resulting in documents nested six or seven levels deep, inconsistent structures, and even circular references. For anyone new to this kind of work, a warning is warranted: automating changes across a massive documentation corpus all at once requires exceptional care, extensive testing, and well-defined fallback and recovery plans. Read: Danger!

Interestingly, my documentation platform engineering team did not initiate the first version of the refactor agent. A vendor experienced in building agentic workflows with n8n had been commissioned to develop it without my team's direct involvement or oversight. That was the first mistake.

The vendor was highly capable at building agentic workflows, but they did not understand our content, information architecture, source-management model, dependencies, reuse strategy, cCMS, or publishing pipelines. Most importantly, they were not working directly with the authoritative structured source.

Instead, they attempted to ingest the published HTML and round-trip it back into XML. That approach would have discarded or damaged many of the relationships embedded in the original XML source – including metadata, reuse dependencies, references, and cCMS-specific structures. What should have been an XML-to-XML transformation was becoming an HTML-to-XML reconstruction.

It was a disaster in the making.

When we discovered what was happening, we intervened post-haste – read: controlled panic.

We ultimately recoded most of the agent through close collaboration between the documentation platform engineering team and the technical writers who understood the content, structure, dependencies, and organizational quirks – warts and all.

The redesigned solution also exposed an important architectural lesson. Lightweight orchestration tools such as n8n were extremely useful for coordinating the workflow, but they were not sufficient for every operation. We augmented the agent with heavyweight Python code wherever deterministic behavior, precise transformations, validation, or dependency management were required.

In other words, we did not ask the LLM or orchestration layer to do work that conventional software could perform more reliably.

After extensive testing and accommodation for numerous exceptions, the properly architected refactor agent performed exceedingly well. It operated as a one-shot autonomous agent across a large content corpus and required relatively little cleanup by the technical writers afterward.

A refactoring effort of this magnitude and complexity would previously have been impractical because of the human resources required. The agent did not merely make the work faster – it made a previously unrealistic and unthinkable project achievable.

Lessons Learned

- **Start with the authoritative source.** Never reconstruct structured documentation from published output when the original structured source is available.
- **Domain expertise is part of the architecture.** Agent developers must work directly with the writers, platform engineers, and others who understand the content and its hidden dependencies.
- **Know your dependencies before automating.** Reuse, references, metadata, publishing pipelines, and cCMS behavior can make seemingly simple transformations extraordinarily complex.
- **Use AI where AI adds value – and deterministic code where certainty matters.** Agentic workflows often work best as hybrids of LLMs, orchestration tools, and conventional software such as Python.
- **Test for exceptions, not just the happy path.** Large documentation repositories contain years of inconsistencies and edge cases that will eventually surface.
- **Build recovery into the design.** Corpus-wide autonomous changes require backups, validation, rollback mechanisms, and clearly defined failure states.
- **Do not confuse agentic expertise with documentation expertise.** Knowing how to build an agent does not mean knowing how to safely transform a complex documentation ecosystem.

Agentic Project #2: Agentic Workflow for New Documentation

The next documentation initiative was far more ambitious – build an interactive, stateful agentic workflow that spans the end-to-end process for creating, reviewing, managing, and publishing net-new technical documentation. We'll discuss updating existing documentation in Agentic Project #3 – a far more challenging endeavor.

This was the project intended to address the business challenges described earlier. At a high level, the workflow needed to:

- Ingest a wide range of source knowledge collateral.
- Interpret that source knowledge and generate initial drafts of complete, multitopic user guides for new products and features – with a target of at least 80% technically usable and high-quality content before human review.
- Provide a simple, interactive authoring interface for review, correction, and QA.
- Automate editorial quality checks and optimization.
- Automate content management and publishing steps where practical.

- Integrate documentation work directly into the software development lifecycle (SDLC).

Easy, right?

Changing the Upstream Process

The workflow was only part of the change. Leadership also established new process expectations. The technical writing staff had already been significantly reduced before these agents and workflows were developed, so the operating model could not depend on writers continuing to chase down source information manually.

The directive was to move responsibility for supplying the inbound knowledge required for documentation upstream to the product development teams. That was not an optional request.

Obtaining accurate source knowledge is often one of the most time-consuming parts of technical documentation. Writing itself is only one activity among many – writers also interview SMEs, locate specifications, reconcile conflicting information, validate behavior, structure content, review builds, manage revisions, and coordinate publication. If an agentic workflow is going to automate content generation at scale, it must first solve the knowledge-acquisition problem.

How the Net-New Workflow Works In Production

1. Automatically create documentation work in the SDLC

We worked with the team designing the agentic SDLC so that documentation tickets – Jira in our case – are created automatically for every new product or product update that requires customer-facing documentation.

This deliberately eliminated the informal path in which someone could simply send a note or message to a technical writer and expect documentation to appear. In the previous model, writers often had to discover upcoming changes, chase product owners for details, and sometimes create the documentation tickets themselves.

In the new model, the ticket is created automatically, assigned to the responsible product owner or development team, and remains their responsibility until the required source collateral has been supplied and validated. The release process can then enforce documentation completion as a release criterion rather than treating documentation as an afterthought.

Make no mistake, this is epochal and fundamental paradigm shift for the ages.

2. Require minimum viable source collateral

The documentation workflow does not begin simply because a ticket exists. It launches only when a minimum viable set of defined source collateral is present, referenced, and accessible to the workflow.

That required a ticket-validation agent. For many documentation scenarios, we require a product or feature demonstration video plus selected metadata and supporting material. Depending on the work, the knowledge package can also include Figma designs, UI prototypes, a PRFAQ, code or API artifacts, specifications, Word or PDF documents, Confluence pages, and other approved sources.

The important point is that the agent does not begin from an empty prompt. It begins from an explicit, inspectable body of evidence.

3. Assess the quality and completeness of the collateral

Presence alone is not enough. A folder full of files can still be technically incomplete.

A second agent evaluates the inbound collateral against defined quality and completeness criteria. DITA task, concept, and reference information models provide a useful starting point because they formalize common structures for procedural, conceptual, and reference information. However, they are not by themselves a complete quality rubric. We augment those structures with product-specific criteria, required metadata, expected evidence, and organization-specific rules.

The result is a scored assessment of whether the available knowledge is sufficient to begin generation – and, where it is not, what is missing.

4. Start the stateful workflow and distill the source knowledge

When the documentation ticket reaches the required state and its collateral passes validation, the stateful documentation workflow begins automatically.

The workflow ingests the approved sources. Specialized agents and deterministic services parse, interpret, normalize, and distill each artifact into usable knowledge. Because the workflow is stateful, intermediate results can be persisted, inspected, retried, and recovered rather than forcing the process to start over whenever a step fails or when a human needs to intervene.

5. Generate and organize the first draft

The distilled knowledge is then used by the agentic workflow to generate a first draft. A single documentation request may produce many topics of different types – concepts, tasks, references, Q&A, troubleshooting, or other content structures – each with its own content type-specific structure and model.

The workflow organizes those topics into a logical hierarchy and sequence. In our DITA environment, that means generating the topic structure and a DITA map that defines how the collection is assembled and navigated.

In production, this has reduced the elapsed time required to produce a substantial first draft from an average measured in weeks to roughly twenty minutes for suitable use cases. That does not mean the documentation is finished in twenty minutes – it means the workflow can create a reviewable, structured starting point at a speed that was previously impossible.

6. Bring in the first human-in-the-loop reviewer

This is the first formal human-in-the-loop (HITL) stage. The workflow pauses and presents the generated content through a simplified review interface.

The reviewer can edit, regenerate, add source artifacts, fill gaps, reject inaccurate material, and resolve discrepancies. In our production experience, reviewers commonly retain roughly 80% or more of the generated draft for appropriate use cases, while the remaining content still requires human correction, completion, or judgment.

For net-new content, we use a customized Markdown-based editor. For update workflows in which the authoritative source is DITA, we expose a text-focused editing experience so the reviewer can work with the content without needing to interact directly with XML markup.

7. Require technical certification from the product SME

Who performs the first review can vary by organization. In our implementation, we require a product SME from the development team to perform the initial technical review and edit. In another organization, that role could remain with a technical writer. That’s an organizational choice.

Requiring a product SME has two major advantages. First, if the inbound collateral is deficient, the team that supplied it encounters the consequences immediately and can correct the source knowledge. Second, the SME must explicitly certify technical accuracy and completeness before the workflow can advance.

We make that accountability visible in the interface. Approval is captured at the topic level and for the overall collection, and the workflow records those approvals as operational metrics. The goal is not to remove the writer from quality assurance – it is to put technical accountability with the people who own the product knowledge.

8. Continuously publish the draft to staging

Once a reviewable draft exists, the workflow publishes it to the staging instance of our documentation website – what we, and many organizations, call the Knowledge Center.

The staging version is kept synchronized as the content evolves. That provides reviewers with a realistic representation of the eventual customer experience rather than limiting review to an authoring interface.

9. Perform automated editorial quality assurance

After the SME certifies technical accuracy and completeness, the workflow performs automated editorial quality checks.

We initially experimented with using an LLM for this stage. LLMs can be very effective for rewriting, explanation, and context-sensitive editorial suggestions, but generative output is probabilistic and often varies from run to run. For enforceable terminology, style, grammar, and content-governance rules, we needed a more controlled deterministic AI editorial quality layer.

We therefore integrated Markup AI, formerly Acrolinx into a workflow agent – which provides configurable content guidance, scoring, automated checks, APIs, and quality-gating capabilities. The workflow can evaluate topics against our editorial standards, capture scores, surface issues in the UI, and feed those measurements into operational dashboards.

The architectural lesson was important – generative AI and deterministic or rule-governed quality controls are complementary, not interchangeable.

10. Automatically classify the content

Another agent classifies the generated collection against a curated taxonomy.

Those classifications are not decorative metadata. They support downstream search, filtering, personalization, analytics, delivery, reuse, generation of knowledge graphs and digital twins, and other forms of automated processing. The quality of this classification layer becomes increasingly important as the content supply chain becomes more machine-driven.

11. Transfer control to a documentation professional for final QA

At this stage, workflow control and the interface move to an experienced documentation professional.

The documentation professional performs final QA across the collection. If additional editorial refinement is needed, they can make it directly. If the content is technically incomplete or inaccurate, they can either correct the issue or return the workflow state to the SME for remediation (which is also captured and tracked in the analytics).

This creates an explicit feedback loop rather than allowing weak content to flow downstream simply because an earlier agent completed successfully.

12. Convert and commit the approved content into the content-management environment

After final approval, the workflow enters the publishing and post-production phase. Depending on the operation, tasks may be performed by deterministic services, agents, or humans.

In our environment, approved drafts are converted into DITA source format and committed into the component content management system (cCMS). This is a critical boundary – the generated content becomes managed source content and must conform to the structures, metadata, and governance rules of the production repository.

13. Allow expert intervention where the cCMS still requires judgment

An authorized and skilled documentation professional can intervene before external publication whenever the content needs additional work in the cCMS. In our case, that environment is IXIASOFT.

For a completely new guide, the workflow may require little additional structural work. For a new feature being inserted into an existing guide, however, a documentation professional may still need to determine where the new topics belong in an existing user guide, how they interact with existing reuse and navigation structures, and whether additional updates are required elsewhere in the collection.

That is an area in which future agents may eventually automate more of the decision-making, but it should not be automated until the required rules and context can be expressed reliably.

14. Publish the approved documentation

Finally, an authorized documentation professional promotes the completed content from staging to the live publishing environment.

The workflow therefore remains highly automated without eliminating governance at the point where customer-facing content becomes authoritative.

15. Capture and track everything in a persistent system-of-record (SOR)

It is critical that the agentic workflow create and maintain a persistent system-of-record for every job and every document it processes.

These records form an operational inventory that persists for the life of the content. That lifespan may be brief – for example, a temporary KCS article needed only until a product issue is resolved – or it may extend for years for core product documentation.

Each record contains the metadata required to identify, manage, track, and govern the content throughout its lifecycle. It can also capture workflow state, ownership, approvals, quality scores, timestamps, exceptions, change history, and other metrics needed for both operational management and auditability.

In our implementation, the SOR supports fully interactive dashboards for agentic jobs, document inventory, workflow status, and analytics. This operational layer gives us visibility into what is being created or changed, where each item is in the workflow, who or what is responsible for the next action, and how the system is performing over time.

This is what we mean by **agentic documentation operations management**. And, like the content workflows themselves, much of this operational management can also be automated – using deterministic methods where repeatability and control are required, and agentic methods where interpretation, analysis, or adaptive action adds value.

The Architecture Behind the Workflow

What is described above isn't theory; it's deployed in production use. It is the core workflow, but it is not the endpoint. Additional capabilities can and will be layered onto the same architecture – for example, an adaptive interview agent that asks targeted questions when the inbound knowledge validation agent detects gaps, a Docs-as-Test agent, accessibility checks, legal and trademark review, code-change detection, and localization support to name a few.

The workflow also captures operational metrics throughout the process. Those measurements feed interactive dashboards that track throughput, quality, approvals, rework, cycle time, content scores, exceptions, and other end-to-end signals. Together, they form an operations control layer for managing the content supply chain – a subject substantial enough to warrant its own discussion.

Stateful Orchestration

A workflow of this length and complexity requires durable state. We use Temporal as a serious agentic orchestration backbone. Temporal is designed for durable execution – workflow state and progress can survive process failures, allowing long-running operations to resume instead of restarting from the beginning. That makes it well suited to workflows that must wait for external systems, pause for human review, retry failed activities, and recover safely.

Other architectures can use different components. LangGraph, for example, provides persistence, interrupts, and human-in-the-loop patterns for agent graphs. Tools such as n8n and Zapier are excellent for integration automation and can participate in larger agentic systems, but a long-running, highly governed documentation workflow may require additional persistence, recovery, versioning, and application-state design around them.

The important distinction is not lightweight versus heavyweight for its own sake. It is whether the orchestration layer provides the durability, observability, recoverability, governance, and human-intervention semantics required by the business process.

MCP and Separation of Agent Assets from Application Code

Model Context Protocol (MCP) also plays an important role in the architecture. MCP provides a standardized mechanism for exposing tools, resources, and prompt templates to AI applications. In our implementation, that separation helps keep agent-facing resources and capabilities from being hard-wired into the workflow code.

That architectural boundary is particularly valuable for documentation teams. Templates, prompts, reference resources, and agent instructions can evolve at a different pace from compiled application logic. With appropriate governance, documentation professionals can own and improve those content-specific assets while platform engineers maintain the orchestration, integrations, security, and production runtime.

MCP does not itself choose an LLM or replace model-routing infrastructure. Rather, it standardizes how an AI application can discover and use external capabilities. Model selection and routing remain architectural concerns that can be implemented separately or exposed through tools.

Lessons Learned

- **Automating writing is not enough.** The larger opportunity is to automate the knowledge flow, governance, review, and publishing system surrounding the writing.

- **Move knowledge accountability upstream.** Agentic generation performs best when product teams are responsible for supplying complete, accessible source evidence rather than forcing writers to reconstruct product knowledge after the fact.
- **Validate inputs before generation.** The presence of files is not evidence of completeness. A production workflow needs explicit minimum-input requirements and automated quality assessment.
- **Treat human review as part of the architecture.** HITL is not a failure of automation. It is how accountability, judgment, exception handling, and technical certification are built into the system.
- **Separate probabilistic generation from enforceable controls.** LLMs are valuable for synthesis and transformation, while deterministic code and governed quality systems are better suited to rules that must be repeatable and auditable.
- **Use durable orchestration for long-running work.** Any workflow that spans systems, approvals, retries, and hours or days of elapsed time needs persistence, checkpointing, recovery, and observability.
- **Preserve structured-content governance.** Generated content becomes production content only after it conforms to the structures, metadata, reuse rules, and repository requirements of the authoritative content-management system.
- **Measure the workflow end to end.** Cycle time, content quality, approval behavior, rework, exceptions, and automation rates turn an agentic workflow from an experiment into an operational system.
- **Keep content expertise in the loop.** The goal is not to remove documentation professionals. It is to shift their effort from chasing information and performing repetitive production work toward architecture, quality, governance, exception handling, and continuous improvement.

What Comes Next

This workflow addresses net-new content. Updating existing documentation is substantially more complex because the workflow must reason about what already exists – including topic boundaries, reuse, dependencies, navigation, historical context, and the risk of unintended change.

We'll discuss that next.

Agentic Project #3: Agentic Workflow for Updating Existing Documentation

Whoo-hoo. We now had a powerful, orchestrated, stateful, multi-agent workflow capable of creating, reviewing, managing, and publishing net-new documentation at scale. Pop the champagne corks!

Not so fast.

Any seasoned documentation professional will tell you that creating entirely new documentation is only part of the job. In most organizations, updating existing content is the far more common day-to-day activity because products, services, interfaces, APIs, regulations, and operating procedures are constantly changing.

We knew we could reuse much of the Net-New workflow as the foundation for Net-Update. The two workflows share many of the same capabilities – ticket orchestration, source-collateral ingestion, input validation, human-in-the-loop review, editorial quality checks, taxonomy classification, content management, staging, metrics, and publishing.

But updating existing documentation introduced a set of challenges that simply do not exist when generating a new guide from scratch. Those challenges turned out to be among the hardest problems in the entire agentic content supply chain.

Branching the Workflow

The first change was straightforward. We added an indicator to the work item – Jira in our environment – that distinguishes a request for net-new documentation from a request to update existing content. That signal determines which workflow branch, agents, and actions are invoked. That can potentially be automated in the futire.

The inbound knowledge process remains largely the same. The workflow still requires sufficient source collateral, and the validation and assessment agents still determine whether the minimum required inputs are present and technically adequate before automation begins.

We are still evaluating how much the assessment criteria should differ between Net-New and Net-Update. That distinction may become increasingly important as Net-Update workflows are triggered automatically by product or code-change detection. A change event may tell us that something changed, but not necessarily whether the available evidence is sufficient to update customer-facing documentation safely.

The Wall

I have euphemistically branded the next phase “The Wall.” It represents one of the central challenges in agentifying updates across an existing large documentation corpus.

The core problem is deceptively simple: How do you identify, quickly and reliably, the exact source content objects that are affected by a product change – and then agentically update only the portions that actually need to change?

What Did Not Work

1. Searching the Published Documentation Website

Searching the documentation website is useful for finding published information, but it is a poor foundation for an automated source-update workflow.

The published site is normally an output representation – often HTML generated from Markdown, XML, AsciiDoc, reStructuredText, or another source format. The workflow, however, must locate the authoritative *source* content objects that will actually be revised and committed back into the content-management system.

Published output rarely preserves the persistent storage identifiers, source-level metadata, reuse relationships, version context, inclusion dependencies, or other information needed to map a rendered page cleanly back to its source. Even when such mappings can be engineered, treating the presentation layer as the primary retrieval layer adds unnecessary ambiguity and potential information loss.

For Net-Update, we therefore needed fast, source-addressable retrieval – not merely website search.

2. Asking an LLM to Search the Entire Corpus

The next idea sounds reasonable on its face: give a powerful frontier model the documentation and ask it to determine what must change.

In practice, that approach did not scale for us. During testing, asking an LLM to analyze even a single very large user guide required hours of processing. In the experiments described in my related paper, some runs extended to roughly four hours, while token consumption became expensive enough that light testing exhausted our allocated budget in only a couple of days.

If you choose that route and I can guarantee a midnight call from your CFO with lots of colorful language.

More importantly, the problem was not only latency and cost. LLM-based retrieval is probabilistic. When we repeatedly asked the model to determine which topics were affected, the candidate set could vary from run to run. That may be acceptable for exploratory search. It is not acceptable when an automated production workflow needs repeatable impact analysis and an auditable explanation of why a source object was selected.

As they say here in the South where I live*, that dog won't hunt*.

If you want to read a review of the whole debacle, check out this entertaining companion paper, Meeting the Snake in the Garden of Eden.

3. Letting the LLM Rewrite an Entire Source Topic

Suppose we somehow solve the retrieval problem and identify the right source topic. Why not simply send the entire topic to ChatGPT, Claude, Gemini, or another model and ask it to make the update?

Because prompting alone cannot guarantee that every unrelated portion of the source will remain byte-for-byte or semantically unchanged. Generative models optimize the output they produce. When given a broad editing surface, they often rewrite wording, normalize style, alter examples, restructure passages, or make other changes that were not part of the requested update.

Some of those changes are obvious. Others can be extremely subtle – precisely the kind most likely to survive human QA and later surface as a customer support problem, or worse.

I see the same tendency in a completely different domain. As a hobbyist songwriter, I use AI to create commercial performances of songs I have written in their entirety. Ask a generative system to change one small element in a performance and it enthusiastically "improves" several others along the way almost every time. In music, that is maddening. In technical documentation, it can be dangerous. As a shameless plug to get more plays, check out my original songs on Spotify or iTunes – just ask it to "*Play Michael Iantosca Songs*".

The lesson was clear: the LLM should not be given an entire topic if only one subsection, step, warning, command, or UI label needs to change.

So, Do We Give Up on Agentic Net-Update?

No – but the architecture has to change.

A reliable Net-Update workflow needs a mechanism that can identify the affected source objects before generation, preserve source identity and dependencies, isolate the smallest meaningful content fragment that requires revision, send only that bounded fragment to the LLM, and then merge the approved change safely back into the authoritative source.

If your content model uses advanced reuse by reference, inclusion, conref, transclusion, shared variables, or similar mechanisms, the workflow must also understand those relationships. Changing one source fragment may affect many publications, products, versions, or delivery contexts.

I told you this wasn't trivial!

The Solution in a Nutshell

I explore the problem in detail in my paper, "Agentic Technical Documentation: Meeting the Snake in the Garden of Eden." The central architectural lesson is that an LLM alone is not enough for enterprise-scale Net-Update. The workflow first needs a governed, source-addressable semantic layer that can perform impact analysis before the LLM ever begins editing.

- **Build a synchronized digital twin of the source corpus**. Represent the documentation as a machine-readable knowledge graph that preserves source identity, structure, metadata, references, relationships, version context, and links back to the authoritative content-management system. The graph is not the source of truth – it is an enriched representation of it. The cCMS remains authoritative.
- **Use the graph for deterministic impact analysis.** Convert the product-change signal into structured facts, then query known relationships among products, components, topics, publications, versions, roles, and content elements. This sharply narrows the candidate set without asking an LLM to read the entire corpus.
- **Add semantic retrieval where it improves recall.** A graph-derived vector index can act as meaning-based radar, finding additional topics or fragments that use different terminology or have incomplete metadata. Graph-derived vector RAG retrieval suggests candidates; the graph verifies their product, version, provenance, lifecycle, and publication context.
- **Retrieve at the smallest useful information unit.** Do not automatically hand the LLM a 500-page guide – or even a complete topic – when a single procedure step, warning, short description, or UI label is the actual change target.
- **Bound the LLM's job to revision, not discovery.** Once the graph and retrieval layer identify the approved target, the LLM receives only the relevant source fragment plus the evidence describing the product change. The model writes; it does not decide the entire impact surface.

- **Patch the approved change back into authoritative source.** The workflow preserves the source object and its surrounding structure, reinserts only the approved revision, validates the result, and leaves unrelated content untouched.
- **Preserve traceability and human review.** Every selected fragment can be traced to its source, product context, publication, version, provenance, and change signal. Human reviewers can see not only what the system changed, but why that content was selected.

The rule of thumb is simple: the vector layer may suggest, the knowledge graph should verify, and the LLM should edit only the final selected content.

That sharply narrows the LLM's role. Instead of using an expensive probabilistic frontier model as the search engine, impact-analysis engine, dependency resolver, and editor all at once, we use deterministic and semantic preprocessing to narrow the problem first. The LLM is then applied only where probabilistic language generation adds value.

Why the Knowledge Graph Matters

For our architecture, the foundation is a synchronized knowledge-graph digital twin of the documentation estate. With structured content such as DITA XML, the graph can preserve internal structure and source relationships at a level conventional chunk-based retrieval often loses. For heterogeneous content estates, iiRDS can add a standardized semantic layer describing what information is about, who it serves, and the product, task, lifecycle, version, and delivery contexts in which it applies.

The result is not merely better search. It is a machine-readable impact map of the documentation environment – one that can connect product changes to affected source objects and, where the model is detailed enough, to the specific sections or elements inside those objects.

Traditional naïve RAG vectors do not preserve the full context of our documentation corpus. A knowledge-graph digital twin-derived vector of the source content does!

Drop mic.

That is what turns Net-Update from a clever LLM demonstration into an industrial-strength documentation workflow.

Lessons Learned

- **Net-Update is fundamentally an impact-analysis problem.** Writing the replacement text is often the easy part. The hard part is determining exactly what must change and what must not.
- **Published HTML is not the authoritative source.** Production update workflows need persistent, source-addressable identifiers and direct relationships back to the managed source objects.
- **Do not use the LLM as the corpus-wide search engine.** At enterprise scale, that approach can be slow, costly, inconsistent, and difficult to audit.
- **Probabilistic retrieval and deterministic impact analysis are different things.** Semantic similarity is excellent for finding possibilities; governed graph relationships and constraints are better suited to confirming applicability.
- **Minimize the editing surface.** The smaller and more precisely bounded the fragment sent to the LLM, the lower the risk of unrelated rewriting and the easier the review.
- **Reuse dependencies must be first-class data.** In component content, changing one object can affect many publications. The workflow must understand those relationships before applying an update.
- **Keep the source of truth separate from the semantic twin.** The graph mirrors and enriches the corpus, but the cCMS or source repository remains authoritative.

- **Traceability is mandatory.** A production system should be able to explain why content was selected, what evidence triggered the change, what was modified, and where the revised source was committed.
- **Use each technology for what it does best.** Graphs provide explicit relationships and validation, vector retrieval improves semantic discovery, deterministic code protects structure, and LLMs perform bounded language transformation.

Why Structured Source Matters

Having our source content in DITA made this architecture about as close to automatic as you are likely to get.

Could you build the same kind of digital-twin knowledge graph from Markdown, AsciiDoc, reStructuredText, or another source format? Absolutely. Those formats can be parsed into structured representations, enriched with metadata, represented in a graph – including with semantic vocabularies such as iiRDS – and used to generate a semantic or vector index.

But there is a catch.

DITA (or any XML/DOM format for that matter) gives us something particularly valuable for Net-Update workflows – explicit, addressable structure below the topic level. DITA topics are composed of semantically identified XML elements, and DITA itself is designed around modular topics, metadata, linking, and content reuse. Elements can be independently identified and referenced, and DITA provides native mechanisms such as conref and conkeyref for fragment-level reuse.

That means our digital twin can represent not merely a topic as a node, but the individual sections, steps, warnings, examples, commands, UI elements, and other meaningful structures within that topic – while maintaining their relationships to the authoritative source. Machine processing needs that.

This is why my mantra about writing for AI has consistently been "*Write for humans; structure for machines*."

Other formats can certainly support fine-grained retrieval, but the required boundaries and semantics may need to be inferred from syntax, conventions, headings, custom metadata, AST structures, or additional preprocessing. The issue is therefore not that Net-Update is impossible without DITA. It is that DITA gives us much of the structure, semantic typing, metadata, reuse, and addressability we need before we write a single line of agentic infrastructure. DITA maps also explicitly organize topics and relationships, while topic and map metadata can carry product, audience, platform, and other contextual information valuable to a knowledge graph.

For organizations contemplating a major transformation of their documentation architecture, those advantages should factor into the decision about whether enriching an existing lightweight format or migrating some or all of the source to DITA offers the better long-term return.

For a deeper discussion of those advantages, see my paper DITA Wins!.

The Payoff

When the digital twin and its associated semantic retrieval layer are built correctly, the advantages become substantial.

1. **Impact discovery becomes wicked fast**. Instead of asking an LLM to read hundreds of pages, we narrow the search across the graph and a vector index generated from graph-derived representations. That search can

operate across not merely one user guide, but across hundreds, thousands, or tens of thousands of source documents and content objects.

2. **The process becomes far more deterministic and repeatable**. This distinction matters. Vector similarity itself is not deterministic impact analysis – it is a candidate-discovery mechanism. The deterministic part comes from using graph relationships, identifiers, metadata, product and version constraints, and other governed rules to validate the candidates. The vector layer can suggest; the graph can verify.
3. **The economics change dramatically**. Graph queries and conventional retrieval do not require us to send the entire documentation corpus through an LLM. A vector index does require embedding computation, which may itself have infrastructure or API costs, but corpus retrieval need not consume generative-LLM inference tokens. By the time an LLM is invoked, we can often provide only the relatively small fragment that actually requires revision. That can reduce generation-token consumption by orders of magnitude compared with repeatedly submitting complete guides or topics.

Your CFO, and you, can sleep through the night.

4. **Dramatically reduce unintended LLM changes**. The workflow does not hand the model an entire source file when only one subsection needs to change. It isolates the smallest practical editing surface, submits that fragment for revision, and patches the approved result back into the authoritative source. Everything outside that boundary can remain untouched.
5. **Every retrieved object remains addressable back to authoritative source**. The graph is a digital twin – not a replacement repository. Its nodes retain source identity and provenance so the workflow can retrieve the correct object from the cCMS or source repository and return an approved update to the correct location.
6. **Content reuse becomes part of impact analysis instead of an unpleasant surprise**. DITA provides explicit mechanisms for reuse at the topic and element level, including content references. Those relationships can be represented in the graph, allowing the workflow to determine where a shared fragment originates and where it is consumed.

   That becomes equally important during human QA. The workflow operator needs to know when the content being reviewed is reused elsewhere – and what other publications, products, or contexts might be affected if that shared source is changed.

6. **Avoid treating documentation as anonymous chunks**. A conventional vector-RAG implementation often divides source documents into chunks optimized primarily for retrieval size – completely discarding its context - then folks wonder why a chatbot question about one product returns an answer alright – about a different product! Unless considerable additional engineering is applied, chunking can weaken or lose important document-level relationships such as topic hierarchy, source identity, metadata, links, reuse dependencies, publication context, and semantic element types.

   A digital-twin, graph-informed retrieval model can preserve those relationships rather than forcing the retrieval system to reconstruct them after chunking.

8. **The digital twin can stay synchronized with the corpus**. Once the ingestion pipeline is established, additions, modifications, and deletions in the authoritative source can be propagated into the graph and its associated indexes incrementally. This does not happen magically – synchronization logic, change detection, and index maintenance still have to be engineered – but structured XML with stable identifiers makes that process considerably easier to automate.

   The important difference is architectural: we are maintaining a living digital twin, not periodically rebuilding a disconnected pile of embeddings.

9. **The graph can become semantically richer than the source itself**. This may be one of the most valuable long-term benefits. Automated classifiers can derive taxonomy, relationships, concepts, entities, product associations, task classifications, and other metadata and add them to the graph without forcing all of that

enrichment back into the authoring source.

That allows the semantic model to evolve continuously while keeping the source appropriate for authors and the cCMS.

10. **Domain ontologies can be linked directly to the documentation twin**. Once the corpus exists as a knowledge graph, it can be connected to domain-specific ontologies representing products, components, capabilities, personas, tasks, lifecycle states, business concepts, APIs, and other enterprise knowledge.

    Now the system knows considerably more than the words that appear in a topic. The graph can be queried directly with SPARQL, while SHACL can validate structural and semantic constraints. Where deeper inference is required, an RDF/OWL reasoner or an explicit rules layer can derive additional facts from the ontology and graph. Those relationships and rules are inspectable and traceable – fundamentally different from relying on an LLM alone to infer applicability probabilistically.

    It can begin to understand what the topic is about, what it relates to, where it applies, what depends on it, and why a particular product change may require that content to change.

And that is the larger point.

The knowledge graph and vector index are not merely a more sophisticated search engine. Together, they provide the semantic and structural substrate that makes industrial-strength Net-Update automation practical – allowing us to find the right source, understand its relationships and dependencies, isolate the precise content that needs attention, and only then bring an LLM into the process.

What Comes Next

Once you can reliably identify and isolate the affected source fragments, the remainder of the Net-Update workflow can rejoin many of the same stages used in Net-New – human review, technical certification, editorial optimization, taxonomy classification, source validation, staging, final QA, and publication.

But crossing The Wall is the prerequisite. Without reliable source-level impact analysis and bounded editing, automating updates across a large documentation corpus is not industrial-strength automation – it is automated risk.

Have a better mouse trap? Tell me about it, I'm all mouse ears!

Selected References and Further Reading

Official standards and sources used to validate selected factual and architectural statements in this paper:

- OASIS – DITA Version 1.3 specification
- iiRDS Consortium – intelligent information Request and Delivery Standard
- W3C – SPARQL 1.1 Query Language
- W3C – Shapes Constraint Language (SHACL)

- Model Context Protocol – specification and documentation
- Temporal – durable execution documentation
- LangGraph – persistence and human-in-the-loop documentation
- Markup AI – Acrolinx is now Markup AI

Continue to Part Three

Part Three: Agentic Documentation – People – examines the human side of the transformation: how documentation roles change when content production becomes agentic, which responsibilities become more important, and why experienced documentation professionals remain essential as designers, governors, evaluators, and knowledge architects.