The Future of Agentic AI Automated Technical Documentation is Already Here

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

Part 2 of 3: The real deal: Agentic AI documentation automation – Case studies

MICHAEL IANTOSCA
SENIOR DIRECTOR OF KNOWLEDGE PLATFORMS AND ENGINEERING
AVALARA

This is a three-part paper:

PART 1: Agentic AI documentation: How we got here
PART 2: The real deal: Agentic AI documentation automation – Case studies
PART 3: The near future: What comes next and the impact on technical documentation careers

The Real Deal: Agentic 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 agentic 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 agentic 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.

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.

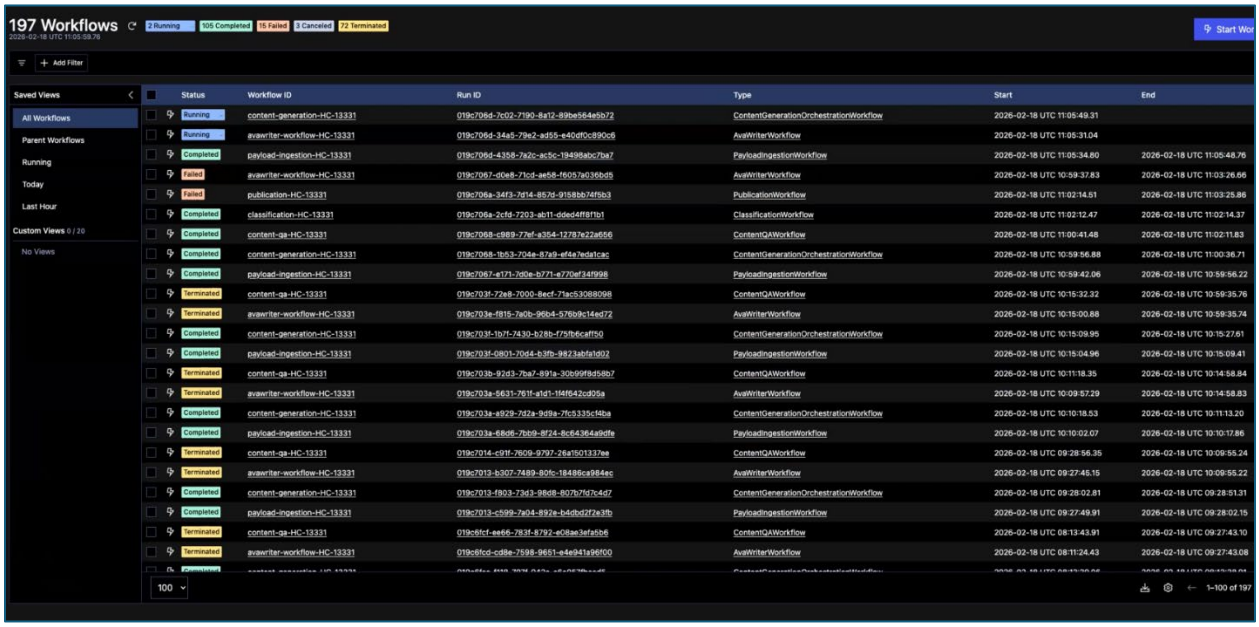

Figure 1. Individual agents that comprise the overall agentic workflow.

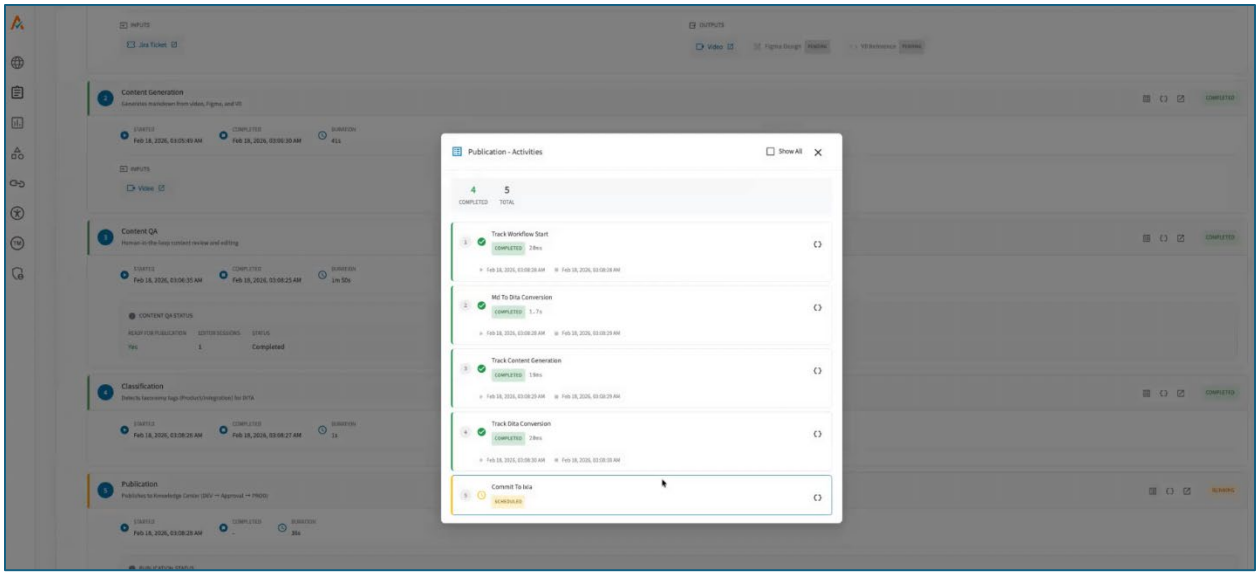

Figure 2. An active workflow during execution

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.

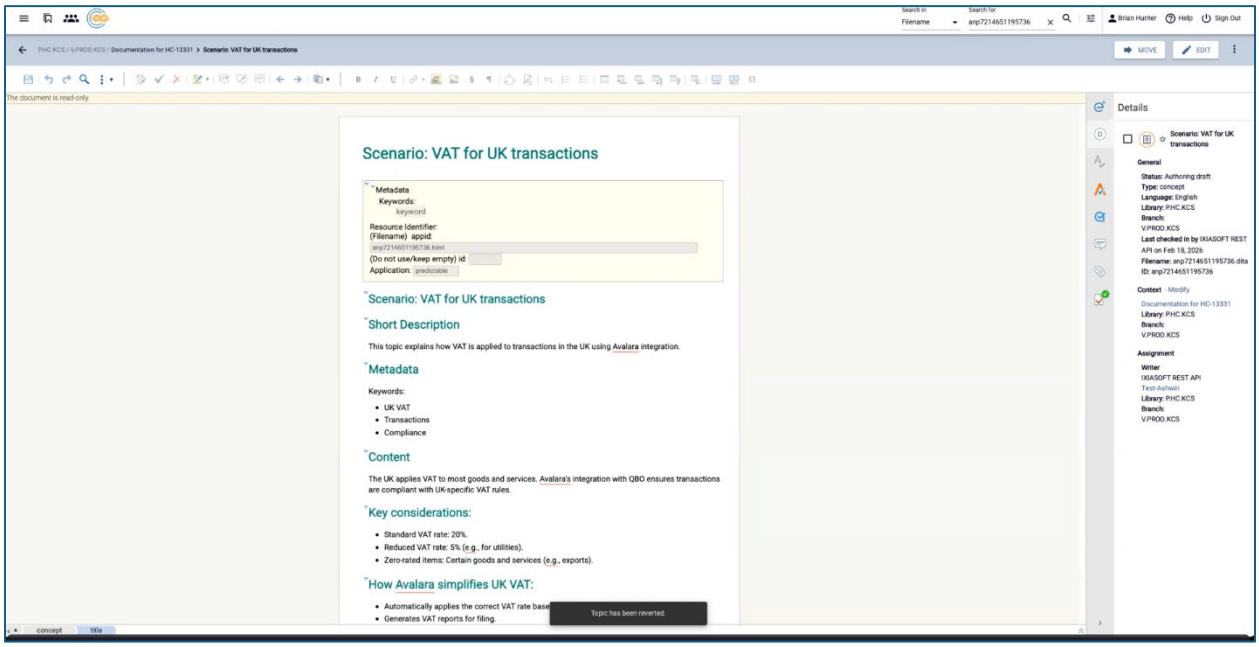

Figure 3. Document automatically converted to DITA XML and stored in the CMS for publishing.

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.

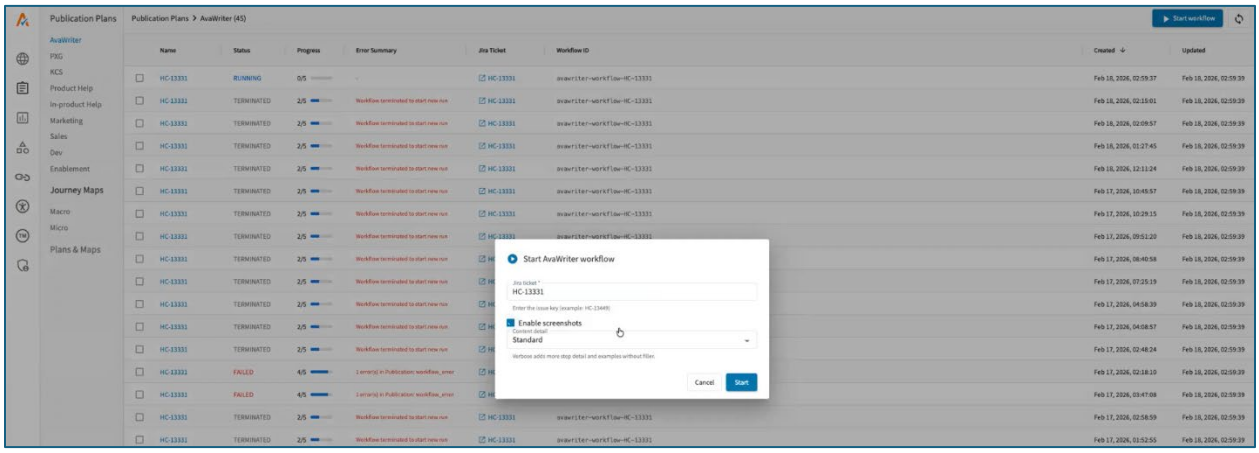

Figure 4. A central operations management center

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 agentic 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 Agentic 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.

Next:

PART 3: The near future: What comes next and the impact on technical documentation careers