THOUGHT FOR THE DAY

It’s the End of the Road for Docs-as-Code

The agentic documentation age has exposed the limits of markdown and Docs-as-Code. Agentic AI is forcing technical documentation organizations to confront a fact that structured-content practitioners have understood for decades: *machine intelligence depends on information intelligence.*

Michael Iantosca
Senior Director of Knowledge Platforms and Engineering
Avalara

Preface

I'm going to give y'all the brutal truth about what we discovered while building and deploying what we believe is one of the first end-to-end enterprise agentic documentation workflows – if not the first of its kind operating at this level. What follows is not based on theory, architectural diagrams, conference-room speculation, LinkedIn debates or what might someday be *possible*. It comes from hands-on development, deployment, failure, iteration, and learning what actually works when agentic documentation moves from concept to production. There will be those who argue that other architectures can reach the same destination. Perhaps they can. But possibility is not proof. A theoretical path is not the same thing as a functioning enterprise model.

What I'm sharing here is what we learned by actually building one.

The semantic layer is no longer optional

With agentic AI documentation workflows, we have entered the age of **semantic content infrastructure**. It's the new **Information Architecture**. Across technology, organizations are no longer talking about a managed semantic layer as a nice-to-have enhancement. For any AI system expected to deliver accurate, traceable, context-aware results at enterprise scale, it is rapidly becoming a requirement. That shift should force a serious reassessment of the formats and workflows the documentation industry has spent the last decade or so treating as modern by default.

For years, advocates of structured content have made the same argument: the intelligence in the data matters. A large language model cannot reliably recover semantics that were never encoded. Vector embeddings cannot manufacture deterministic relationships from structurally ambiguous content. Retrieval-augmented generation can improve access to relevant passages, but it does not transform a pile of loosely structured files into a governed *and extensible* knowledge model. If an organization wants provenance, traceability, explicit relationships, granular context, reuse, and deterministic processing, those capabilities must exist somewhere in the information architecture before the model is asked to reason over it.

That is precisely what SGML, XML, DITA, schemas, and structured content models were designed to provide. They were never merely fussy authoring syntaxes. Their purpose was to encode what information is, how it is contained, what it relates to, and how machines can process it predictably. The uncomfortable realization for the industry is that many of the ideas now being marketed as essential components of an AI-ready semantic layer are not new at all. What changed is that the surrounding systems have finally become capable of exploiting them.

We did not outgrow structure – we traded it away

Many organizations, particularly larger enterprises, retained highly structured authoring models because the operational benefits were obvious: reuse, conditional processing, validation, modularity, metadata, controlled vocabularies, and predictable transformation. Others moved toward lighter-weight formats that were easier for humans to type and easier to place into developer-centric toolchains. Markdown was attractive for understandable reasons – it was simple, readable, inexpensive, and compatible with the Git ecosystem.

But simplicity was not free. Markdown achieved much of that simplicity by discarding semantic richness. The trade was effectively this: remove explicit structure and meaning from the source, then rely on convention, filenames, folder structures, front matter, prose patterns, and downstream code to

reconstruct what the source no longer says explicitly. That was tolerable when the primary goal was publishing pages for humans. It is a far weaker bargain when the consumer is an autonomous system expected to reason, act, explain itself, and prove where its answers came from.

The defense of Markdown often rests on the claim that LLMs read it well. They do. That is not the same thing as saying Markdown is an ideal source representation for an agentic knowledge architecture. LLM readability is a very low bar. The real question is whether the source can reliably drive preprocessing, entity extraction, relationship formation, digital-twin state models, knowledge graphs, policy enforcement, dependency analysis, provenance, and deterministic transformations without forcing the system to infer structure that could have been explicit from the start.

Docs-as-Code confused proximity with collaboration

Docs-as-Code made another assumption that deserves scrutiny. By placing documentation beside software source code in Git, the model promised that developers would naturally participate in documentation because the files were now in familiar territory. Sometimes that happened. Often it did not. Putting a Markdown file next to source code never guaranteed that an engineer would maintain it, contribute, understand its information architecture, recognize downstream dependencies, or know what else needed to change when a feature changed.

A Git repository is not a collaboration strategy. A pull request is not an information architecture. And Markdown is not a semantic model. Those are tools and storage conventions that were elevated into a philosophy because they fit the software-development culture of the period. The fact that Docs-as-Code worked well for some teams does not make it the natural end state for enterprise knowledge operations.

Agentic workflows expose the weakness of the original collaboration premise. For the first time, documentation participation does not have to depend on a developer remembering to open a file. Agents can detect code changes, API changes, UI changes, requirement changes, test changes, and downstream content dependencies. They can identify affected documentation, initiate updates, route reviews, validate references, enforce policy, and synchronize related content. Collaboration can become a property of the workflow rather than a favor we repeatedly ask humans to perform.

The old objection to XML has collapsed

One of the strongest historical objections to XML and DITA was authoring complexity. Writers had to understand elements, attributes, schemas, validation rules, nesting, and content models. The author was required to express intent explicitly by placing information inside meaningful containers. Even when visual XML editors improved, many organizations had already committed to lightweight source formats and inexpensive tooling, and the industry increasingly treated explicit structure as unnecessary friction.

Agentic authoring changes that calculation. Humans no longer need to interact directly with XML tags – but they should not need to interact directly with Markdown syntax, reStructuredText, AsciiDoc, or any other serialization format either. In a mature agentic environment, the authoring experience should be visual, conversational, and intent-driven. Agents should handle source serialization, tagging, metadata, schema conformance, relationships, repository operations, synchronization, and validation behind the scenes.

Once the human no longer bears the cost of the syntax, choosing a semantically weak representation because it is easier to type becomes a bizarre architectural decision. It amounts to optimizing tomorrow's machine infrastructure around yesterday's human inconvenience. If the source is increasingly written, normalized, classified, tagged, and maintained by agents, then the rational choice is to preserve as much machine-usable meaning as possible under the hood.

Markdown does not become semantic because you bolt metadata onto it

Can a digital twin or knowledge graph be built from Markdown? Of course. Using iiRDS It can also be built from PDFs, database rows, transcripts, and scraped web pages. The fact that something can be processed does not make it an optimal source format. The meaningful comparison is how much ambiguity, enrichment, inference, custom code, and human intervention are required before the information becomes suitable for deterministic machine processing.

The typical answer with Markdown is to add the semantics back later. Organizations attach YAML front matter to thousands of files, establish naming conventions, invent relationship keys, define content-type identifiers, create block conventions, add custom parsers, maintain dependency maps, and impose rules describing what particular pieces of text are supposed to mean. Yet topic-level metadata still does little for semantic containment inside the topic. The moment the system needs machine-actionable granularity below the page level, it has to introduce more delimiters, more conventions, more embedded structures, or another parsing layer.

At some point the contradiction becomes impossible to ignore: the organization rejected structured markup because it was supposedly too complicated, then spent years engineering conventions to recover the structure and semantics it deliberately removed. Push that process far enough and you have effectively XML-ified Markdown – only with weaker validation, inconsistent semantics, less reliable containment, no industry standard, and a collection of organization-specific rules that every downstream system must learn. That is not simplicity. It is displaced complexity.

Git can stay underneath – it does not need to define the experience

The same distinction applies to Git. Git may remain an entirely reasonable storage or version-control layer in an agentic architecture. What no longer follows is that writers, reviewers, subject-matter experts, and knowledge managers should organize their work around branches, commits, merge conflicts, repository layouts, and pull requests simply because those concepts are natural to software engineers.

Agentic systems increasingly automate repository interaction. They can retrieve content, make coordinated changes, validate them, preserve history, reconcile dependencies, synchronize variants, and submit changes through whatever underlying controls an organization requires. Once that layer is automated, Git becomes infrastructure rather than user experience. Databases remain underneath applications too, but we do not argue that every business user should work directly in SQL because the database happens to be there.

This is where defenders of Git-centric documentation often confuse implementation with architecture. Retaining Git somewhere in the stack is not evidence that Docs-as-Code remains the correct operating model. If agents abstract repository mechanics away from the people creating and governing knowledge, then one of the central reasons for organizing documentation around Git has disappeared.

The LLM should not be asked to reconstruct reality

Another misconception needs to be eliminated: the argument for structured source is not that raw XML should simply be dumped into an LLM instead of Markdown. An LLM shouldn't ever see either. That would miss the point. In an advanced agentic architecture, source content feeds a preprocessing and semantic layer before the LLM is asked to generate anything, fast, at scale, and without token expenditure.

Schema-based content can contribute directly to digital twins, knowledge graphs, ontologies, entity networks, dependency models, applicability rules, and other deterministic representations. Those structures establish the world in which the model operates – products, components, procedures, states, versions, relationships, constraints, and provenance. Retrieval can then work from explicit context rather than lexical similarity alone, and reasoning can combine symbolic relationships with probabilistic language capabilities.

The LLM should not be forced to reconstruct that world every time it receives a prompt. It should operate against a governed representation of the world that already exists. That architecture reduces how often the system must guess, gives agents a much stronger basis for multi-hop reasoning and action, and reserves probabilistic generation for the tasks where LLMs actually excel.

The resistance is increasingly about inertia, not architecture

There will continue to be vigorous defenses of Markdown, Git, and Docs-as-Code. Some of those defenses are technically legitimate – existing ecosystems are mature, developer adoption is high, migration is expensive, and not every documentation operation needs a sophisticated semantic architecture. Small teams in particular may make perfectly rational tradeoffs in favor of lightweight tooling.

But those practical considerations should not be confused with a claim of architectural superiority. Many organizations have invested deeply in Markdown pipelines, static-site generators, Git workflows, CI/CD automation, custom extensions, and homegrown metadata conventions. People know these systems, have built careers around them, and have accumulated substantial infrastructure around them. That investment deserves respect. It does not deserve immunity from reassessment.

Sunk investment is not an information architecture. Familiarity is not technical superiority. Developer convenience is not semantic richness. Popularity does not prove that a format designed largely for lightweight human-authored text is the best representation for autonomous systems that require granular context, provenance, traceability, explicit relationships, deterministic processing, and governed machine reasoning.

Docs-as-Code was a transition – not the destination

The agentic documentation era demands architectures optimized for a different set of constraints: human-level or better accuracy, deterministic processing where determinism matters, granular context, semantic intelligence, provenance, traceability, relationship awareness, automated governance, speed, cost containment, and far less dependence on hallucination-prone probabilistic inference. Achieving that means putting intelligence into the information infrastructure instead of repeatedly asking an LLM to infer intelligence from semantically impoverished files.

Markdown had its moment. Git-centric documentation had its moment. Docs-as-Code had its moment. Each solved genuine problems and moved the discipline forward. But defending them as the architectural ideal for agentic documentation because they were successful in the previous era is a failure to distinguish historical usefulness from future fitness.

The central question has changed. We are no longer designing primarily around how easily a human can type the source. We are designing around how much reliable intelligence machines can derive from information before an LLM ever has to guess. Once that becomes the design criterion, the case for semantically rich, schema-governed source becomes not nostalgic, but strikingly modern.

> **The future is not "XML instead of AI." It is structured, governed semantics underneath AI – so the model has less to infer, less to hallucinate, and far more to know.**

Michael Iantosca | Senior Director of Knowledge Platforms and Engineering, Avalara