

Formalizing Document Assurance

A Topological Framework for Verification, Validation, and Human Accountability

When AI assists in creating systems engineering documentation, who bears responsibility for the result?

PRESENTER

Michael Zargham, PhD

*CEO and Chief Engineer,
Dynamical Systems Group*

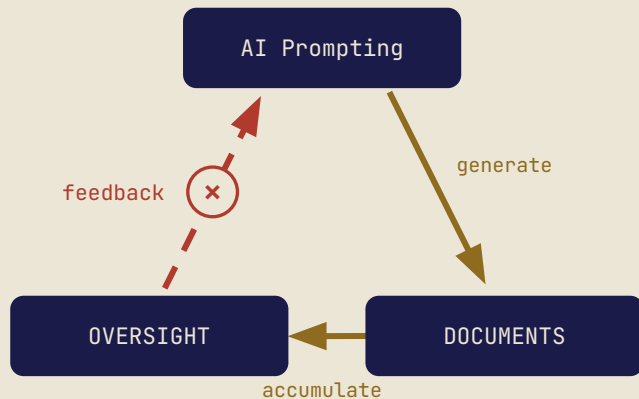
INCOSE International Symposium · 2026

DSG is an
R&D firm
and
systems integrator
operationalizing
emerging tech.

Professional Services,
not a product company.

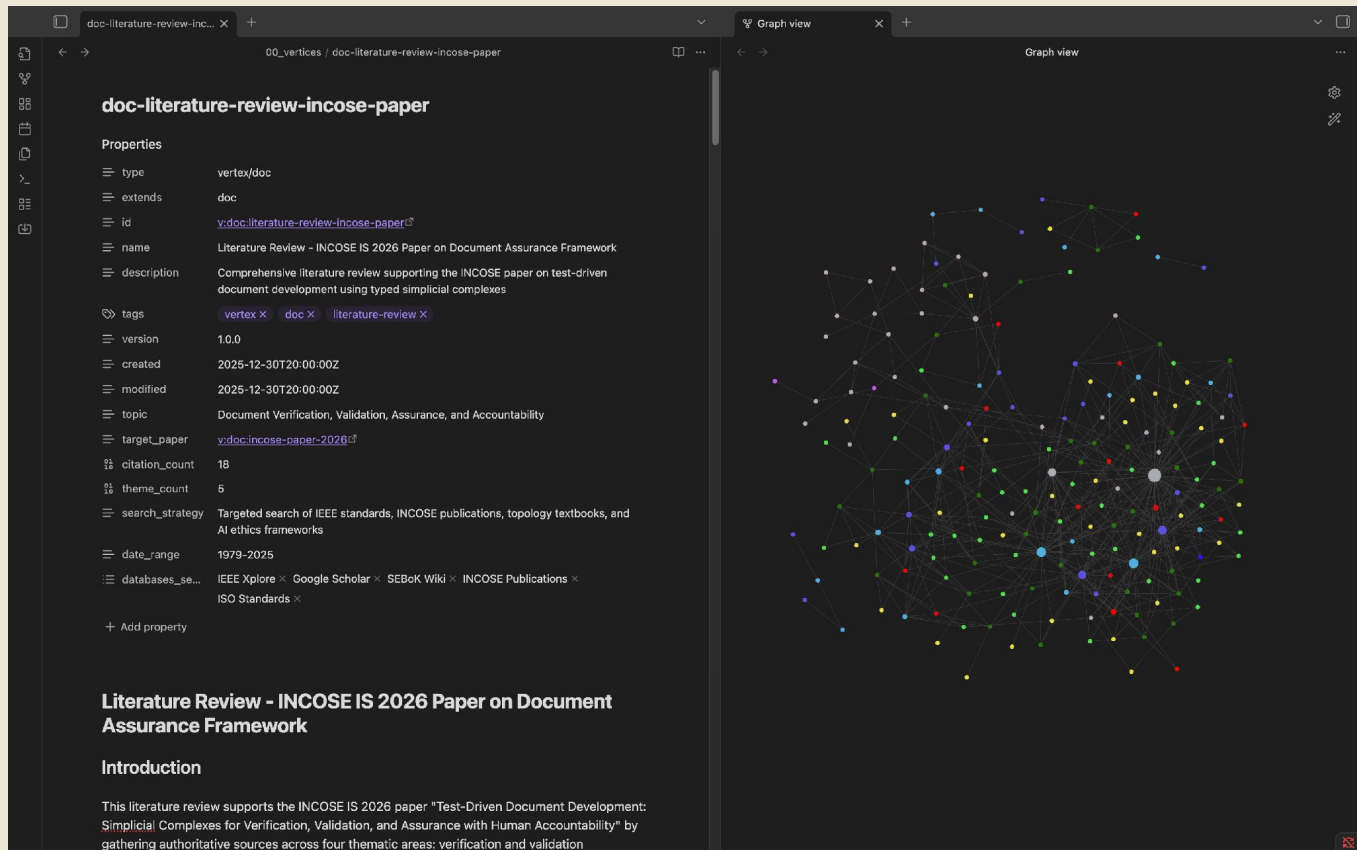
Generation Velocity goes up, Oversight must keep up.

- **Velocity.** LLM lets a team draft documents faster.
- **The catch.** Those documents refine and reference each other; a constellation, not a stack.
- **The question.** How do I trust the whole web when AI helped?
- **The gap.** Throughput scaled; the human oversight did not.



the corrective loop is weak at best

We need accountable oversight that scales with AI-assisted drafting.



The screenshot displays a web application interface with two main panels. The left panel, titled "doc-literature-review-incose-paper", shows the properties of a document. The right panel, titled "Graph view", displays a complex network graph.

doc-literature-review-incose-paper

Properties

- type: vertex/doc
- extends: doc
- id: [y.doc.literature-review-incose-paper](#)
- name: Literature Review - INCOSE IS 2026 Paper on Document Assurance Framework
- description: Comprehensive literature review supporting the INCOSE paper on test-driven document development using typed simplicial complexes
- tags: [vertex](#) [doc](#) [literature-review](#)
- version: 1.0.0
- created: 2025-12-30T20:00:00Z
- modified: 2025-12-30T20:00:00Z
- topic: Document Verification, Validation, Assurance, and Accountability
- target_paper: [y.doc.incose-paper-2026](#)
- citation_count: 18
- theme_count: 5
- search_strategy: Targeted search of IEEE standards, INCOSE publications, topology textbooks, and AI ethics frameworks
- date_range: 1979-2025
- databases_se...: [IEEE Xplore](#) [Google Scholar](#) [SEBoK Wikd](#) [INCOSE Publications](#) [ISO Standards](#)

Literature Review - INCOSE IS 2026 Paper on Document Assurance Framework

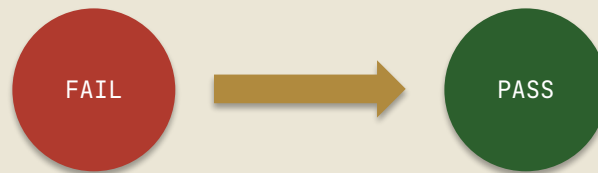
Introduction

This literature review supports the INCOSE IS 2026 paper "Test-Driven Document Development: Simplicial Complexes for Verification, Validation, and Assurance with Human Accountability" by gathering authoritative sources across four thematic areas: verification and validation

The right panel, "Graph view", shows a complex network graph with numerous nodes (colored dots) and edges (lines) connecting them, representing a large dataset or a complex system.

Test-driven development

- Write the check before / alongside the artifact.
- The artifact is not “done” until the check passes.
- **Engineers** already trust code this way — red, then green.
- **The move.** What if documents had tests too? — but documents need two kinds of test.



the check exists before the artifact does

Verification, validation & assurance

VERIFICATION

Are we building the product right?

Boehm, 1984

Conformance to the specified requirements. Machine-checkable, deterministic pass / fail.

IEEE 1012 · ISO/IEC/IEEE 15288

VALIDATION

Are we building the right thing?

Boehm, 1984

Fitness for intended use — satisfies stakeholder needs.

Requires human judgement.

INCOSE SE Handbook, 2023

ASSURANCE

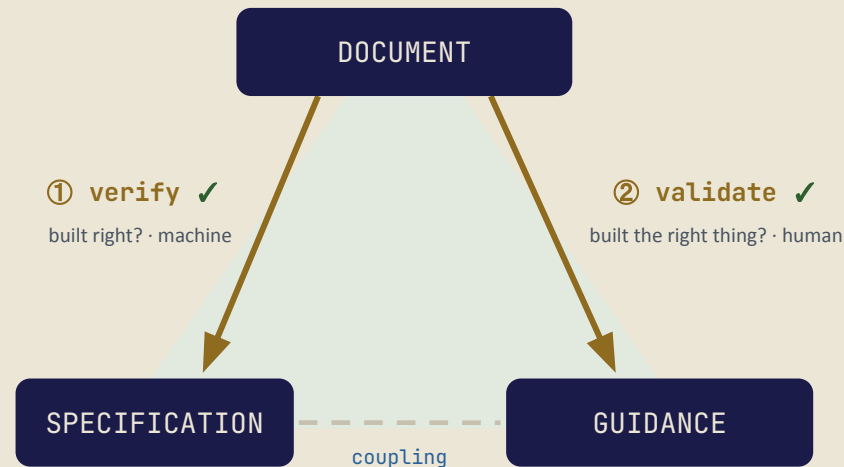
Is quality attested — by whom?

qualified authority

Verification, validation, signed off by a qualified, accountable authority.

verification can be automated; validation cannot.

Ontology — *classes, subclassing, disjointness*



- **Document** — broad class of which Spec and Guidance are subclasses which inherit rules and heuristics but are more specialized
- **Specification** — contains formal machine checkable rules that allow for deterministic pass-fail checks (like tests for software)
- **Verify** — built right? Structural conformance to a spec. *Machine*.
- **Guidance** — contains guidelines, heuristics, rubrics and other review procedures for evaluating content conformant to a spec
- **Validate** — built the right thing? Fidelity to author intent, and in-content sufficiency judgement. *Human*.

Specification and Guidance are disjoint subclasses of Document

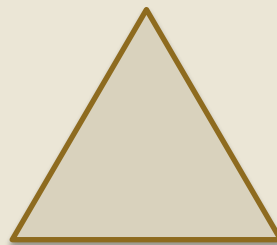
Topology – *vertices, edges, faces*



vertex (0)



edge (1)



face (2)

Vertices, edges and faces provide distinct but related elements.

An edge can only be added if both bounding vertices exist.

A face can only be “filled in” only when its bounding edges close a loop.

Ontology says *what* must be true. Topology says *where* to check.

ONTOLOGY – what must be true

- **Document** is a typed class.
- **Spec / Guidance** are disjoint subclasses of Document.
- **Verify / validate / couple** are typed relations.
- **Assurance** = attested quality.

TOPOLOGY – where to check it

Document → vertex

3 relations → edges

Assurance → a closed 2-face

Audit → a recursively check dependencies are Assured

Document development is loopy — at every scale



TYPE SCALE

spec ↔ guidance ↔ doc co-evolve

*the document type stabilizes
(supports reuse)*



DOC SCALE

draft → verify → validate → revise

a single document evolves in context



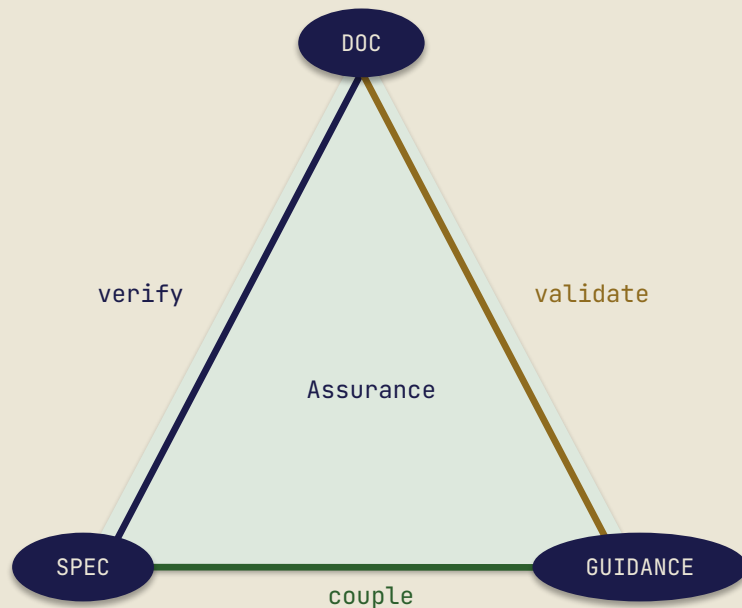
PROJECT SCALE

audit the whole constellation

*Change management tracks cascades
across dependencies*

The same verify / validate pattern recurs at every scale — which is exactly what a recursive structure wants.

The assurance triangle



THREE EDGES CLOSE A FACE

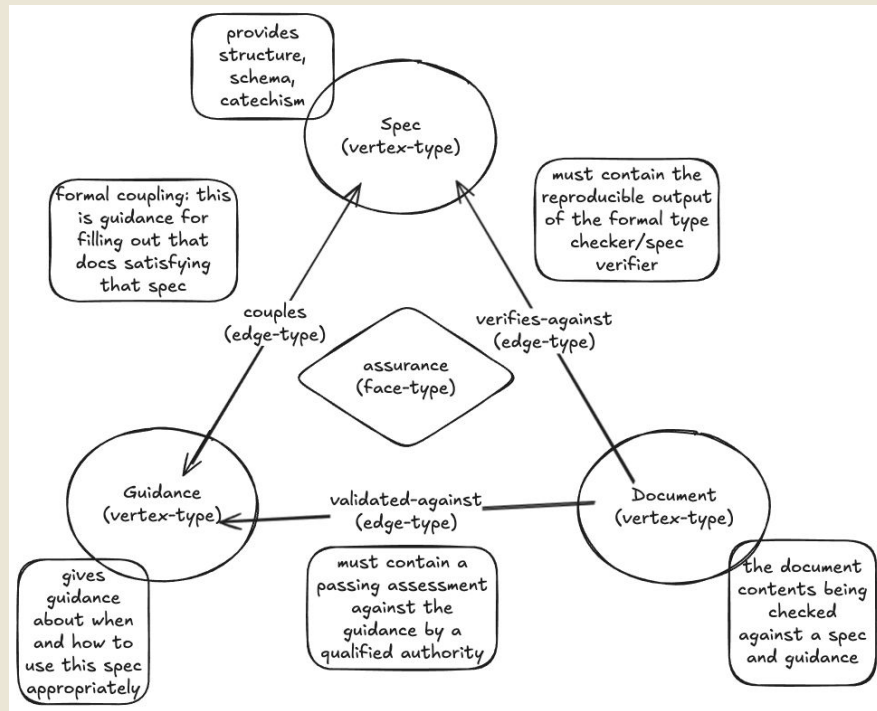
- **Verify** (doc–spec): machine conformance ✓
- **Validate** (doc–guidance): human judgement ✓
- **Couple** (spec–guidance): the matched pair defines the doc type ✓

No coupling → no triangle. You'd be verifying and validating against unrelated standards — the attestation would be meaningless.

Every check is local and well-defined

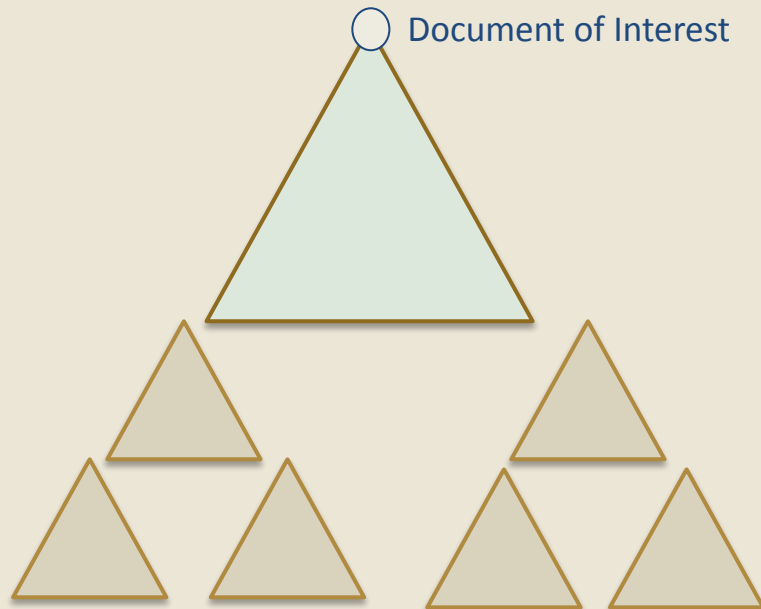
- **Verification** is mechanical — run it.
- **Validation** is one accountable human judgement against explicit guidance.
- **Coupling** is a one-time pairing of spec and guidance.
- You never assess the whole constellation at once — only one triangle at a time.
- You need formal guarantees that the local checks assemble into a valid global check.

*Accountability is recorded, not assumed:
every face carries signed_by + dates.*



What about the *spec and guidance*?

- **The objection.** You validated the doc against a guidance — but who assured the guidance? the spec?
- **The answer.** Backtrack. A spec is itself a document.
- It has its own triangle: verified against a spec-for-spec, validated against a guidance-for-spec, coupled.
- Same for the guidance. Recurse.



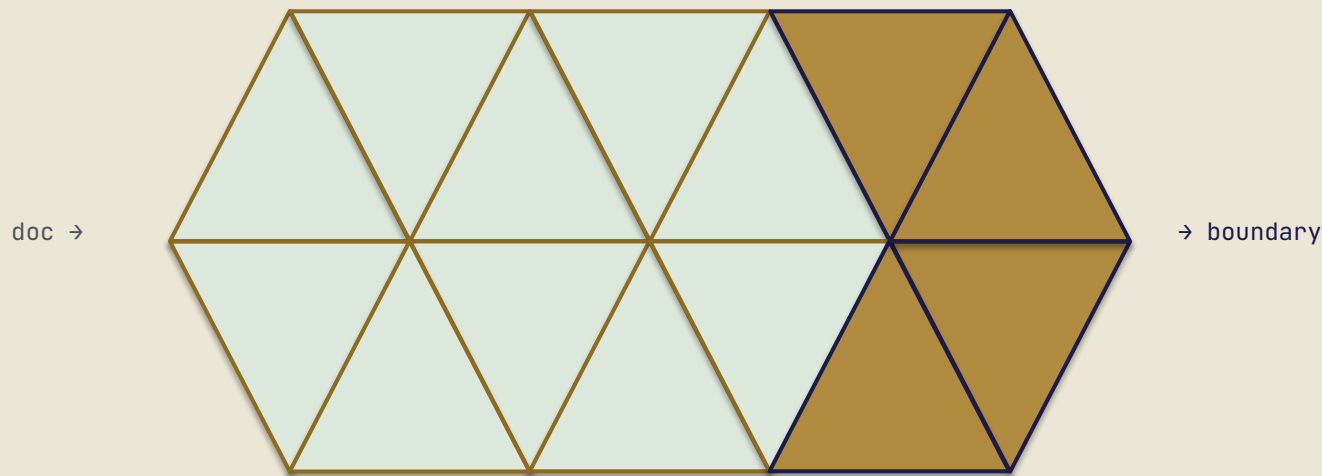
Four self-describing documents axiomatically anchor everything

ID	Document	Coupling	Verification	Validation
SS	spec-for-spec	SS:GS	SS:SS	SS:GS
SG	spec-for-guidance	SG:GG	SG:SS	SG:GG
GS	guidance-for-spec	SS:GS	GS:SG	GS:GG
GG	guidance-for-guidance	SG:GG	GG:SG	GG:GG

Self-describing and mutually anchored; a single root resolves the self-reference paradox. These four are the axioms — all audits trace back to them.

An audit is a trace of tiled triangles to the boundary

Triangles tile by sharing edges (two papers share one doc-type edge). Follow the assurance faces, triangle to triangle, until every path lands on a boundary face.

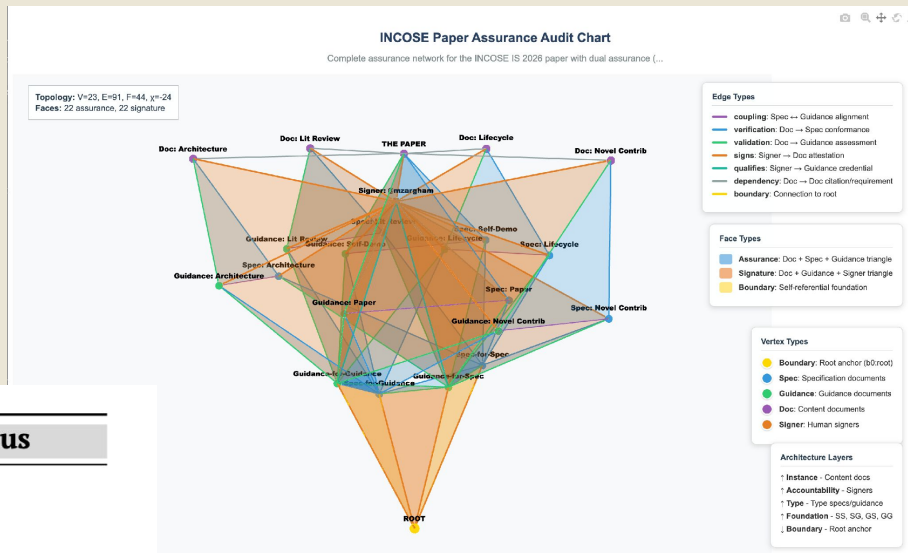


Stylized view of a tiling

The framework assured its own paper

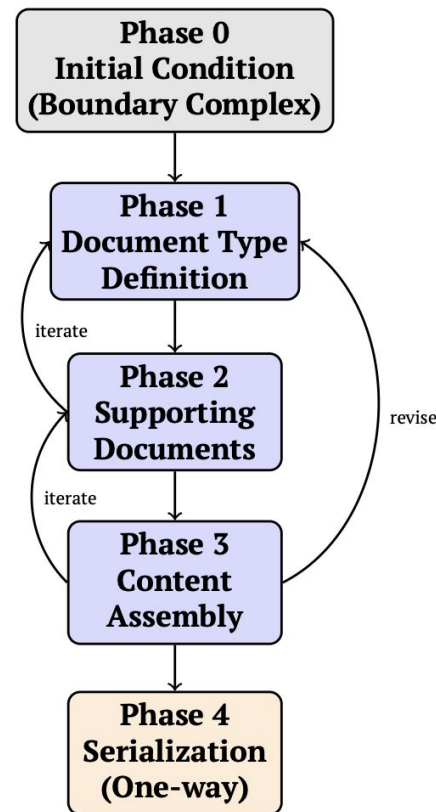
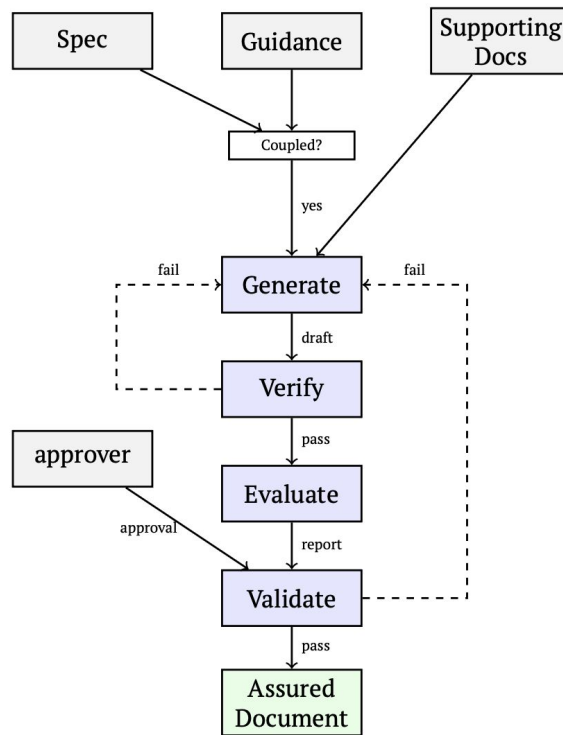
- I defined, implemented and evaluated this paper's contents using AI.
- AI drafts at high velocity; a recursive, well-defined oversight function gates every document.
- It emulates an engineering authority performing an accountable oversight role.
- human judgement stays human and explicit, exactly at the validation edges. Everything else is mechanical stored structurally.

Metric	Value	Status
Total vertices (V)	22	—
Assurance faces (F)	22	—
Approved Validation Coverage	22	—
	100%	PASS
	(22/22)	



What I actually did:

Element	Technology	Rationale
Document Store	Directories, Markdown files, github	Human-readable, version-controlled
Type System	YAML, Python	Human readable, library support
Verification	Python script	Types & structural compliance checks
Validation	human review with obsidian; Claude Code for reports; git commit to sign	LLM assistance for assessments with human accountability for decisions
Audit Tool	Python script	Coverage and topological analysis
Accountability	Git history, approver field	Username as identity, Github Actions enforces rules
UI for Develop and Write	VS Code with Claude Code	Smooth UX for LLM collaboration
UI for Read and Review	Obsidian Vault	UX for navigating, reading and editing



Two implementations

v1 — aaa-docware

- **Python + YAML** — markdown docs with frontmatter.
- **CLI** — aaa verify / build / audit.
- Proves the construction end-to-end.

Bespoke Implementation with all definitions and rules
materialized in python classes.

v2 — knowledgecomplex + docs-as-code

RDF-native typed simplicial complex:

OWL carries the ontology

SHACL provides verification (conformance)

SPARQL the audit trace is a query

Documents are compiled views on data in a quadstore and
history in github.

github.com/DynamicalSystemsGroup/assurances-audits-accountability • github.com/DynamicalSystemsGroup/knowledgecomplex

Three things to take home

- 01 Verification is machine; validation is human judgement — keep them disjoint.
- 02 Assurance is a closed triangle; Audits provide a trace to a small well-defined boundary condition.
- 03 Topological structure provides semantics richness without losing “local” checks to offer “global” coverage guarantees.