SYSTEM-AS-CODE FEST 2026 · H1

# meMO

## Medical Device Architecture Modeling as Code for Ontology-Backed Safety Assurance

A SysML v2-native ontology and toolchain that keeps a medical device's safety evidence **aligned as the design changes** — typed, computable, and architecture-backed.

Ontology | Methodology | Tools | Architect

MEMO:: 0.2.0 · OPEN SOURCE · SYSML V2 · ISO 14971 · IEC 62304 · ISO/IEC/IEEE 42010

**SOMESH KASHYAP** · SENIOR PRINCIPAL SOFTWARE SYSTEMS ARCHITECT · MEDTRONIC

linkedin.com/in/someshkashyap

01 ABOUT ME

# Seasoned in taming safety-critical medical devices. Some days they listen.

## Somesh Kashyap

Senior Principal Software Systems Architect · Medtronic

linkedin.com/in/someshkashyap

**15+**

**years in medical devices**

software & systems architecture

**7+**

**clinical areas**

UKR · TKA · PFA · FAI · cardiac ablation and mapping · abdominal surgery · soft-tissue surgery · robotic-assisted surgery

**4**

**medical devices**

orthopedics · general surgery · electrophysiology

**Navio**

ORTHOPEDICS

BLUE BELT

**CORI**

ORTHOPEDICS

Smith+Nephew

**LUNA***

GENERAL SURGERY

ASENSUS

**Affera Mapping & Ablation**

ELECTROPHYSIOLOGY

Medtronic

* LUNA Surgical System is not approved by the FDA and is not available for sale.

02 OVERVIEW

# How the talk is organized.

The deck moves from the safety-evidence problem to meMO, then into the ontology, methodology, GPCA example, and adoption guidance.

| | | | |
|---|---|---|---|
| 01 | Safety evidence | Why traced documents still drift when the architecture and design change. | PROBLEM |
| 02 | Introducing meMO | What meMO is: the missing semantic layer for medical-device assurance. | FRAMING |
| 03 | Ontology | Packages, core types, architecture layers, semantic links, and closure constraints. | REUSABLE LIBRARY |
| 04 | Methodology | Profiles, viewpoints, workflow gates, usage rules, and project binding. | HOW TO APPLY IT |
| 05 | GPCA example | A public-domain infusion pump thread from clinical need to evidence view. | CONCRETE INSTANCE |
| 06 | Adoption suggestions | How to pilot, extend, validate, and grow without expanding the core vocabulary. | PRACTICAL ROLLOUT |

PART ONE

# Safety evidence drifts as the design changes.

Medical devices became software-intensive systems. Their safety case is spread across documents that are traced but not always meaningful — and it loses alignment as the design evolves.

01

03 DEVICE COMPLEXITY

# Medical devices are becoming complex.

Connected, configurable, software-defined cyber-physical systems. Safety now depends on how the whole system behaves, not on any single requirement.

## As complexity accumulates, how do we ensure safety under change?

**SAFETY DEPENDS ON**
**behavior**
not only requirements text

**ASSUMPTIONS LIVE IN**
**interfaces**
not only block diagrams

**RISK CONTROLS CROSS**
**teams**
software · systems · clinical · V&V

**EVIDENCE MUST FOLLOW**
**change**
or it goes stale


**So the question is: as complexity grows, how do we keep safety assured as the design keeps changing?**

04 FRAGMENTED EVIDENCE

# The artifacts are linked, but the links lack meaning.

Specialized teams and a document-heavy process. Artifacts are linked, but the links often stop at IDs: they do not capture the design behavior being claimed, the architecture element that implements a control, or the conditions a test actually exercised.

ARTIFACT ISLAND

## Requirements

Traced to tests, but the link does not carry the safety claim it is meant to support.

User needs | System reqs | Software reqs

**Architecture gap:** the test is linked to a requirement, not to the behavior or design it verifies.

ARTIFACT ISLAND

## Risk

Hazards and controls live in separate files; controls are not anchored to the design.

Hazards | Harms | Controls

**Architecture gap:** the control is named, but not anchored to the design feature that implements it.

ARTIFACT ISLAND

## Verification

Evidence proves something — but not always the claim in today's released baseline.

Tests | Results | DHF

**Architecture gap:** the test can pass without exercising the behavior that fails under load.

**The problem isn't too few documents. It's that the links between them can't be checked for meaning or completeness.**

05 INDUSTRY CONTEXT

# Safety-critical domains lean on architecture.

Aerospace and automotive treat architecture as the backbone of the safety case, and they write that discipline into their standards. Medical has strong standards too, but less of the architecture model.

ARCHITECTURE-LED

## Aerospace

Architecture & allocation chains tie function, safety analysis, and verification evidence together.

ARP4754A · ARP4761 · DO-178C

PLATFORM-LED

## Automotive

Platforms & contracts standardize safety mechanisms, interfaces, and reuse across product lines.

ISO 26262 · AUTOSAR · ISO 21434

PROCESS-LED

## Medical devices

Strong process, risk, electrical safety, and cybersecurity standards — but architecture maturity still varies.

ISO 14971 · IEC 62304 · IEC 60601 · IEC 81001-5-1 · ISO 13485

**IEC 62304 requires software architecture, but it does not provide a shared model connecting system design, risk controls, cybersecurity, and verification evidence.**

06 THE GAP

# Why medical architecture stays weak.

The standards ask for architecture, risk, cybersecurity, and verification. The gap is one shared model that all of those activities can use.

TEAM REALITY
**Cross-functional slices**
systems · software · risk · cyber · V&V · quality

PROCESS GRAVITY
**Document-heavy evidence**
DHF, specs, risk files, protocols, reports

PRODUCT HISTORY
**Code-first startup inheritance**
acquired products often carry architecture in code and people

ADOPTION BARRIER
**Tool cost + model setup**
even with tools, ontology and team workflow are missing

!

THE GAP

## Architecture becomes a picture

Useful for review, weak at proving the documented architecture matches what was implemented.

RISK
**Controls float**
not anchored to design features

CYBER
**Threats sit apart**
not tied to behavior and interfaces

VERIFICATION
**Tests miss behavior**
wrong path, wrong load, wrong assumption

IMPLEMENTATION
**Architecture drifts**
code and architecture docs do not stay aligned

**Missing: a shared ontology and low-cost architecture model that connects design behavior, implementation, risk, cyber, V&V, and evidence.**

07 CODE-FIRST DEBT

# Code-first is fast, until assurance needs architecture.

Code is necessary, but it should not become the primary architecture artifact. Medical devices are cyber-physical systems: safety depends on software, hardware, interfaces, timing, sensors, actuators, and change impact.

CODE-FIRST PROVIDES

## Fast path to working software

**Immediate feedback**
engineers can build, run, test, and learn quickly

**Concrete behavior**
the implementation becomes executable proof that something works

**Low ceremony**
small teams can move without a heavy modeling process up front

**Demo momentum**
progress is visible to product, clinical, and leadership stakeholders

THE ASSURANCE QUESTION

## It works, but is it safe?

**Hard to see**
safety boundaries, control loops, hardware assumptions, failure paths, risk-control placement

**Lives in people**
design assumptions remain in developer knowledge

**Code visibility**
not every assurance role can inspect or interpret source code

**Traceability**
links to code do not prove architecture completeness

**Verification**
unit tests can pass while scenarios fail under timing, users, hardware, or faults

**Emergent behavior**
complex interactions can create hazards no single code path reveals

**Interface failures**
wrong assumptions about hardware/software interfaces, units, timing, ownership, or data freshness can break safety

**Change**
engineers can change code without seeing impacted requirements or risk controls

ARCHITECTURE/DESIGN-FIRST PROVIDES

## A reviewable safety argument

**Visible rationale**
why the cyber-physical design is safe, not only what the code does

**Shared model**
systems, software, hardware, risk, cyber, V&V, quality, and regulatory can inspect the same architecture

**Hardware analysis**
sensors, actuators, electrical assumptions, and hardware safety mechanisms stay connected to software behavior

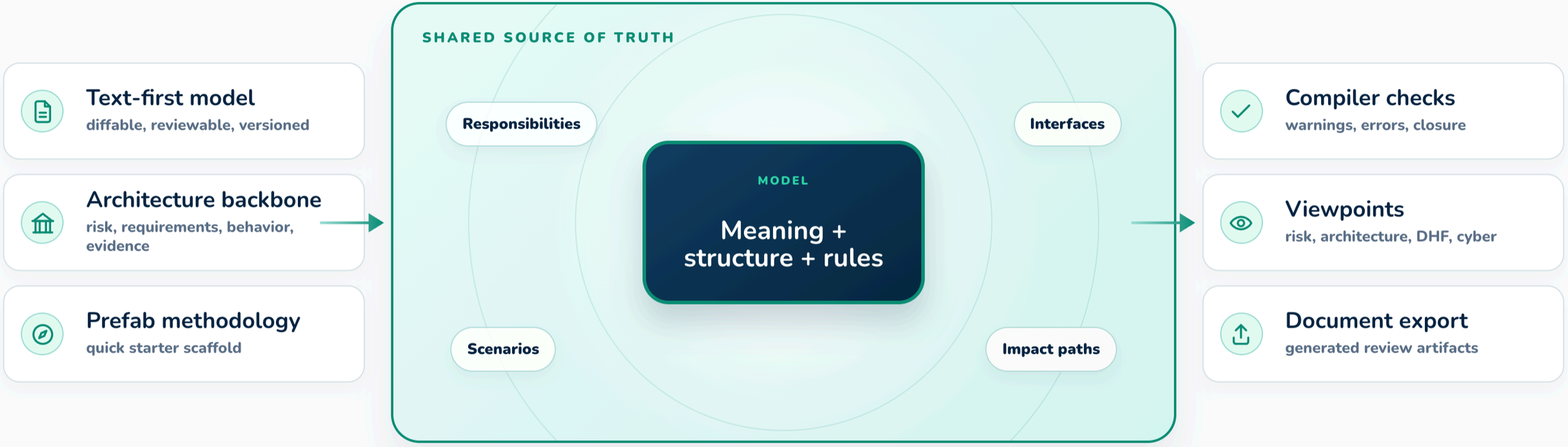
**Explicit links**
behavior, hardware/software interfaces, risk controls, tests, and evidence connect through architecture

**Change impact**
review scope follows design dependencies instead of expert memory

**The goal is not less code.** **It is better assurance around the code: architecture-first or model-informed development makes safety assumptions visible, reviewable, and testable.**

# What is needed is a shared model.

The missing layer is a low-cost, versionable, architecture-backed model that connects engineering meaning across teams.




**The goal:** MBSE benefits with code-first adoption and lower setup friction.

PART TWO

# Introducing meMO.

Medical Engineering Modeling Ontology: a SysML v2-native medical-device assurance layer.

02

09 WHAT MEMO IS

# meMO is custom domain modeling for medical assurance.

It specializes SysML v2 with medical-device domain types, required attributes, semantic relationships, constraints, and generated assurance views.

REFERENCE IMPLEMENTATION `https://github.com/memoarchitect/memo`

SYSML V2 PROVIDES

## Language substrate

Packages, parts, requirements, actions, interfaces, relationships, and typed model structure.

- **Model structure**
- **Behavior**
- **Interfaces**
- **Requirements**

MEMO SPECIALIZES IT

## Medical-device domain model

Hazards, harms, risk controls, safety threads, verification obligations, DHF evidence, and closure rules become first-class model types.

- **Domain type library**
- **Required attributes**
- **Semantic link definitions**
- **Closure constraints**

ENGINEERING RESULT

## Computable assurance model

Design review questions become model queries, validation errors, impact analysis, trace views, and generated evidence artifacts.

- **Validate**
- **Trace**
- **Analyze impact**
- **Generate views**

# meMO defines the missing semantic layer.

A domain ontology for safety-critical medical devices: purpose-built for regulated systems where architecture, risk, verification, and evidence must stay connected.

### Typed elements

Well-defined medical-device artifacts with clear attributes and lifecycle state.

### Typed relationships

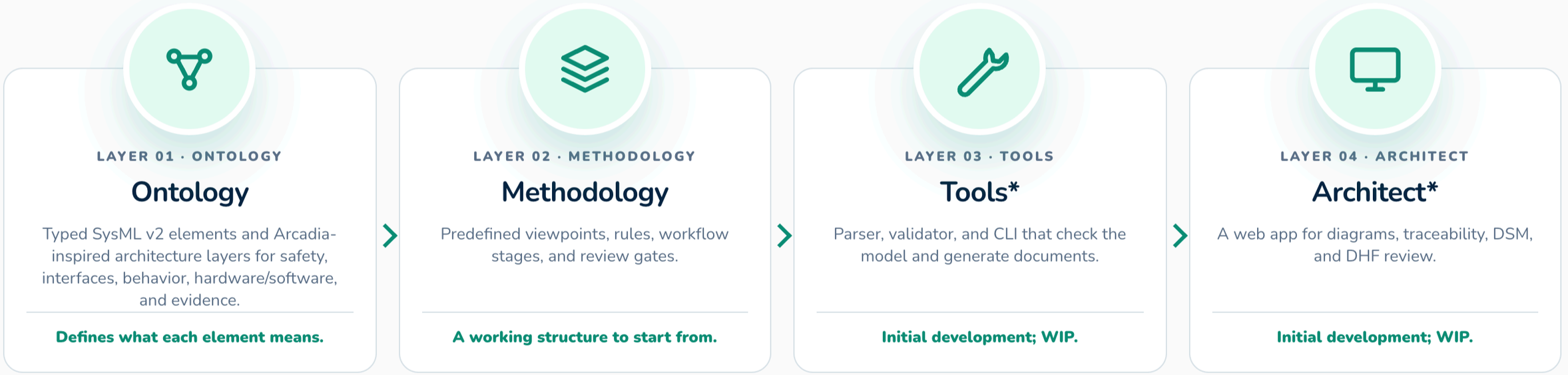
Meaningful connections that express intent, roles, direction, and constraints.

### Closure rules

Logical checks that complete the chain and enforce consistency.

# Four layers, adopt what you need.

meMO is a stack, not a single tool. Today will focus mostly on the ontology and methodology; the tools and Architect are not yet ready for public release.




*** Tools and Architect are in initial stages of development and are work in progress.**

# Ontology — a typed medical vocabulary.

A minimal, medically grounded SysML v2 vocabulary, published as an open spec in one `memo::` namespace, version 0.2.0.

### TYPED ELEMENTS

Every safety-relevant thing is a typed part with regulator-relevant attributes.

- Intended Use
- Hazard
- Harm
- RiskControl
- Requirement
- Architecture
- Behavior
- VerificationCase
- Evidence

### TYPED LINKS & RULES

- **Semantic links** carry roles, not free-form arrows.
- **Closure rules** are checked by walking required semantic links and flagging missing paths.
- **18 architecture layers**, Arcadia-inspired, with risk and cybersecurity as peers.

**The meMO ontology defines the medical-device concepts, attributes, semantic links, and closure rules used to make safety threads computable.**

# Methodology — how teams apply the ontology.

A vocabulary alone does not tell a team what to model first, which reviews to support, or when a model is complete. The methodology layer turns the ontology into repeatable modeling practice.

### 1 Scope the model

Pick the product slice, lifecycle stage, and safety class. This keeps the first model useful instead of trying to represent everything.

### 2 Choose viewpoints

Define who needs the view: architect, safety engineer, V&V, cybersecurity, regulatory, or leadership.

### 3 Use rules and gates

Make review expectations explicit: required elements, required relationships, exit criteria, and missing-evidence checks.

### 4 Bind it to a project

A project can adopt a profile, add domain patterns, and generate review views without changing the released ontology.

**In meMO, methodology means modeling practice:** profiles, viewpoints, workflow steps, quality gates, patterns, and project bindings.

# Tools — parse, check, generate.

A Langium-based model engine and CLI parse text-first SysML v2 into a semantic graph, then run domain rules and closure checks.

**model engine + CLI**

```
model engine
  parser · semantic graph · rule engine
  impact analysis · DSM · DHF · importers

memo CLI
  validate · dev · build · export · import
  ontology · dhf · generate · req

$ cd examples/gpca-pump && memo dev
  → live model + validation at localhost
```

**WHAT THE TOOLS DO**

- Parse text-first SysML v2 into a semantic graph.
- Run native KerML closure & consistency checks — errors, warnings, completeness.
- Import / export: Enterprise Architect, Cameo, OWL, CSV.
- Generate DHF artifacts; validate in CI before review or release.

**Run `memo validate` in CI, and each change produces a defined re-review scope.**

# Architect is an optional visual workbench over the same model.

It helps reviewers navigate. It does not become a second source of truth.

ADOPT ARCHITECT WHEN VISUAL NAVIGATION HELPS

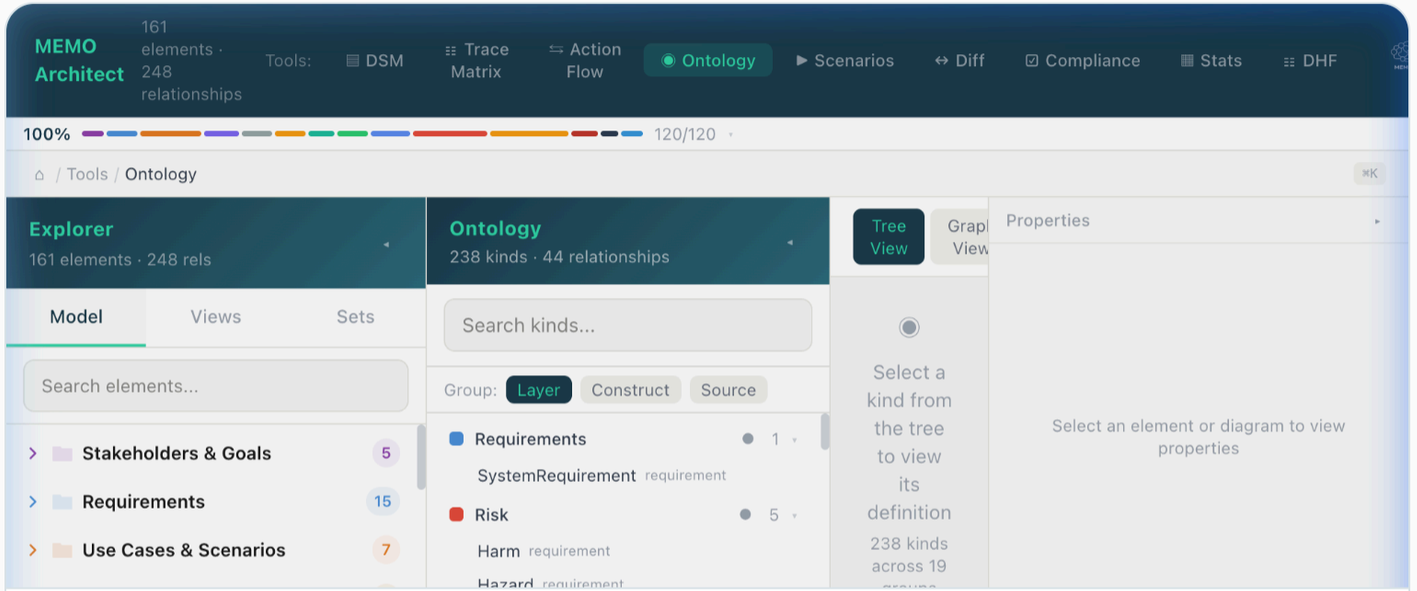
## Thin workbench, shared semantics

**Explore**
browse layers and kinds

**Inspect**
follow typed relationships

**Review**
see scenarios and consistency

**Project**
navigate stakeholder views




**Current prototype:** the ontology workbench reads the same inspectable model used by the CLI.

**Architect is optional. SysML source, packages, and generated artifacts remain usable without the web UI.**

SECTION THREE

# Ontology: the reusable model library.

Start with memo::, its packages, architecture layers, domain types, and typed semantic links.

03

# Keep the mental model small.

A stable core, architecture and process around it, then project extensions and reference examples.

| | | |
|---|---|---|
| **Core** | Traceable elements, documented elements, evidence elements, enums, and typed semantic links. | stable semantics |
| **Architecture** | Arcadia-inspired layers for context, behavior, interfaces, hardware, software, risk, cyber, requirements, and assurance. | model the system |
| **Methodology** | Viewpoints, rules, workflow stages, quality gates, document views, and project bindings. | apply the method |
| **Extensions** | Device-specific modes, interfaces, cybersecurity, usability, AI/ML, and organization-specific profiles. | add domain depth |
| **Examples** | GPCA-style worked trace threads that show definitions becoming concrete project instances. | learn by reference |

# How memo:: is organized.

`memo::medical_device_library` is the public import surface. Product models import one library, then use focused packages underneath it.

| | | |
|---|---|---|
| `memo::core` | *Shared foundation* | identity, traceability, documented/evidence elements, controlled values, and typed semantic relationships. |
| `memo::architecture` | *What the device is* | Arcadia-inspired layers for context, requirements, behavior, interfaces, hardware, software, risk, cyber, assurance, analysis, and decisions. |
| `memo::methodology` | *How to apply the ontology* | profiles, patterns, rules, workflow steps, quality gates, viewpoints, and project bindings. |
| `memo::viewpoints` | *Who needs to see what* | Stakeholder-oriented concerns such as architecture, safety, cybersecurity, verification, and regulatory review. |
| `memo::views` | *Concrete projections* | Diagram and document-backed views generated from the same model source. |
| `memo::compliance` | *Regulated outputs* | Controlled artifacts, change, postmarket, and ISO 14971 risk-management-file views. |
| `memo::examples::gpca` | *Reference product model* | A worked infusion-pump example used to validate and teach the modeling style. |

# memo:: builds bottom-up.

Read the namespace from the base upward: core semantics first, architecture plus device extensions next, viewpoints and generated views above that, methodology to apply the model, and GPCA as a concrete example.

**memo::examples::gpca**

**Concrete instance.** A worked infusion-pump model that instantiates the reusable library as requirements, architecture, risk, verification, trace, and views.

**memo::methodology**

**How teams apply it.** Profiles, rules, workflow steps, gates, patterns, and project bindings select the right subset for a review.

**memo::viewpoints**

`memo::viewpoints`
audience and stage intent: what each role needs to inspect.

`memo::views`
diagram and document-backed views generated from that intent.

**memo::architecture**

Need → behavior
`context · requirements · functions · behavior`

Structure
`logical · software · hardware · interfaces`

Assurance
`risk · cybersecurity · assurance · analysis · decisions`

`+ extensions`
device-specific modes, interfaces, profiles, and domain kinds live beside architecture, not inside core.

**memo::core**

**Foundation.** Traceable elements, documented/evidence elements, controlled values, enumerations, and typed semantic relationships.

# Typed links connect the layers.

Layers don't connect by free-form arrows. Every relation is a native SysML v2 `connection def` specializing `SemanticLink` — its name is the verb, its typed ends carry the roles, a `linkStatus` carries state, and navigation is bidirectional.

| DOMAIN | CONNECTION KINDS |
|---|---|
| **Requirement** | `DerivesFrom` · `SatisfiedBy` · `DerivesInto` |
| **Function** | `AllocatedTo` — function ↔ allocated element |
| **Interface** | `RealizesInterface` · `CrossesTrustBoundary` · `BindsToInterface` |
| **Verification** | `VerifiedBy` · `ProducesEvidence` · `Validates` · `IncludedIn` |
| **Risk** | `MitigatesHazard` · `TracesRisk` · `AssessedAgainst` |
| **Cyber** | `ThreatenedBy` · `Exploits` · `MitigatesVulnerability` · `ImpactsSafety` |

SECTION FOUR

# Methodology: how teams apply the ontology.

Resolve scope, select viewpoints, bind rules to lifecycle stage, and keep project extensions outside the core library.

04

# Methodology profiles constrain the choices.

A team does not pick any rule or view ad hoc. A resolved methodology profile defines the allowed scope, viewpoints, usage rules, gates, and project-specific extensions for the current review.

1. **Set safety classification** Use the product slice, safety class, lifecycle stage, and review purpose to resolve the method.
2. **Select allowed viewpoints** Use the views the profile enables for architecture, safety, cybersecurity, V&V, or regulatory review.
3. **Apply required rules** Usage rules and closure checks say which model content must exist before the gate can pass.
4. **Bind to the project** The project adopts one resolved profile so every review uses the same expectations.
5. **Extend outside core** Add device-specific states, interfaces, controls, and views in project packages.

**Methodology is controlled flexibility:** **adapt the practice through profiles and project packages, not by changing the core ontology.**

SECTION FIVE

# GPCA example: a concrete safety thread.

Now the reusable types become project instances in a public-domain infusion pump example.

05

# Meet GPCA — memo::examples::gpca

A public-domain Generic Patient-Controlled Analgesia infusion-pump benchmark. Useful as a teaching and validation case, but the public reference material is not actively maintained.

## WHY GPCA

- **Public University of Minnesota CriSys example**
- **FDA infusion-pump software-safety research context**
- **Small enough to inspect end to end**
- **Rich enough for architecture, interfaces, behavior, risk, verification, and evidence**

## SOURCES

**UMN CriSys GPCA**
`https://crisys.cs.umn.edu/gpca.shtml`

**FDA infusion pump software safety research**
`https://www.fda.gov/medical-devices/infusion-pumps/infusion-pump-software-safety-research-fda`

## HOW WE USE IT

- **Reference material, not a maintained product baseline**
- **Treated here as an IEC 62304 Class C style example**

## memo::examples::gpca::*

| | |
|---|---|
| `requirements` | needSafeTherapy · reqSingleMode · reqLockout |
| `architecture` | hardware assemblies + software components |
| `interfaces` | logical interface + hardware/software data bus |
| `behavior` | mode machine · alarm monitoring · infusion management |
| `risk` | hazard chain · lockout RiskControl · residual evaluation |
| `verification` | VerificationCase · TestArtifact · Evidence |
| `trace + views` | SemanticLink thread · RMF · diagrams · generated review views |

# One GPCA-style closed thread.

Small enough to follow. Complete enough to prove the semantic backbone.

**Plain scenario.** **A patient presses the dose button for pain relief. The pump may deliver one extra dose, called a bolus, but only if the lockout timer says enough time has passed; otherwise the pump must block the dose to prevent overdose.**

| | |
|---|---|
| Use case / need | `needSafeTherapy` |
| Requirement | `reqLockout` |
| Architecture | `infusionMgr` |
| Behavior | `lockout prevents bolus` |
| Risk control | `prevent overdose during lockout` |
| Verification | `vcLockout` |
| Evidence | `evidenceLockout` |
| Document view | `rmfView` |

### Use case / need

The clinical intent: patient-controlled therapy must stay safe during normal use and fault handling.

### Requirement

A typed obligation derived from the safety scenario, with class and source metadata.

### Architecture

The responsible software item or block that implements the design decision.

### Behavior

The mode, guard, or flow that must hold when the scenario executes.

### Risk control

The ISO 14971 control that prevents or reduces the hazardous situation.

### Verification

The case and acceptance criteria that check the control under the scenario.

### Evidence

The test result or artifact tied to the reviewed baseline.

### Document view

The generated RMF, V&V, or review view produced from the same thread.

**Why this works.** **Change impact is computed from typed relationships across architecture, behavior, risk, verification, evidence, and document views.**

# We compile code. We should compile the safety argument.

With typed elements and typed links, review questions can become executable checks: missing mitigations, missing verification, missing risk evaluation, and incomplete evidence paths.

terminal — closure & consistency

```
$ memo validate

CR-MED-001  Hazard must have ≥1 risk control   (ISO 14971)
            Missing mitigation: hazAirInLine

CR-MED-003  Risk control must be verified      (ISO 14971 §7.4)
            Missing verification: rcDoorOpenAlarm

CR-MED-004  Pre-mitigation risk must be assessed
            against a risk matrix              (ISO 14971)

Result: 2 errors · 1 warning · thread HZ-001 closed
```

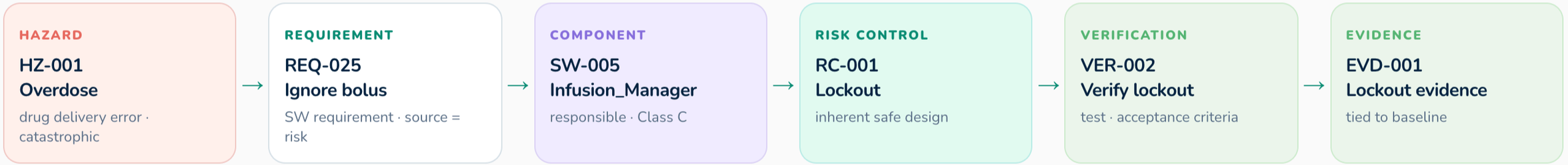
**WHY IT MATTERS**

- Find gaps **before** design review or audit.
- Make closure visible to software, systems, risk, and V&V at once.
- Run the rules in CI before merge or release.
- Turn each change into a deterministic re-review scope.

**This does not prove safety automatically. It makes review gaps visible early enough for engineers, safety, and V&V to act on them.**

# The layers connect into one closed thread.

The instances from the last three layers, linked end to end. Every node is typed; every edge is a `SemanticLink` with status.




**Each row of a traceability table becomes an object you can inspect and query — the RMF view, V&V matrix, and impact analysis all read this thread.**

SECTION SIX

# Adoption suggestions: start small, keep it typed.

Use the ontology and methodology first; introduce automation and UI support as the project workflow stabilizes.

06

# Start with one scenario-driven thread.

Begin with one clinically meaningful use case. Turn it into a concrete scenario, then walk the same scenario through systems engineering, functional behavior, architecture allocation, safety, cybersecurity, and evidence.


**Continuous cross-cutting activities**

**Software safety work**
classify, derive software requirements, update architecture, verify change impact (IEC 62304 context)

**Risk activities**
identify hazard, hazardous situation, harm, risk control, verification, residual risk (ISO 14971 context)

**Cyber activities**
identify assets, trust boundaries, threat scenarios, mitigations, and safety impact

**V&V activities**
define acceptance criteria, verification cases, validation claims, test artifacts, evidence status

**Review + view creation**
continuously generate scenario views, trace gaps, design rationale, and affected decisions

# Adopt by team context.

meMO should fit the team's current maturity. The first win is shared meaning, not tool replacement.

STARTUP PATTERN

## Keep it lightweight

- Start with intended use and one safety thread.
- Model only key architecture, interfaces, and controls.
- Add verification cases and lightweight review views.

ENTERPRISE PATTERN

## Bridge existing tools

- Pick one change-impact problem.
- Reference existing requirements and risks.
- Connect architecture to evidence before integrating tools.

REVIEW PATTERN

## Give each role a view

- Architecture sees responsibilities and impact paths.
- Safety sees controls and residual risk.
- V&V sees targets, evidence state, and release impact.


THE TAKEAWAY

# Create. Compile. Assure.

Author the model, compile the checks, and keep the safety case assured as the design changes — typed, architecture-backed, and re-checkable.

- Faster changes, lower risk
- Stronger cases for regulators
- Less rework

**MEMO:: 0.2.0 · OPEN SOURCE · SYSML V2 · ISO 14971 · IEC 62304 · ISO/IEC/IEEE 42010**