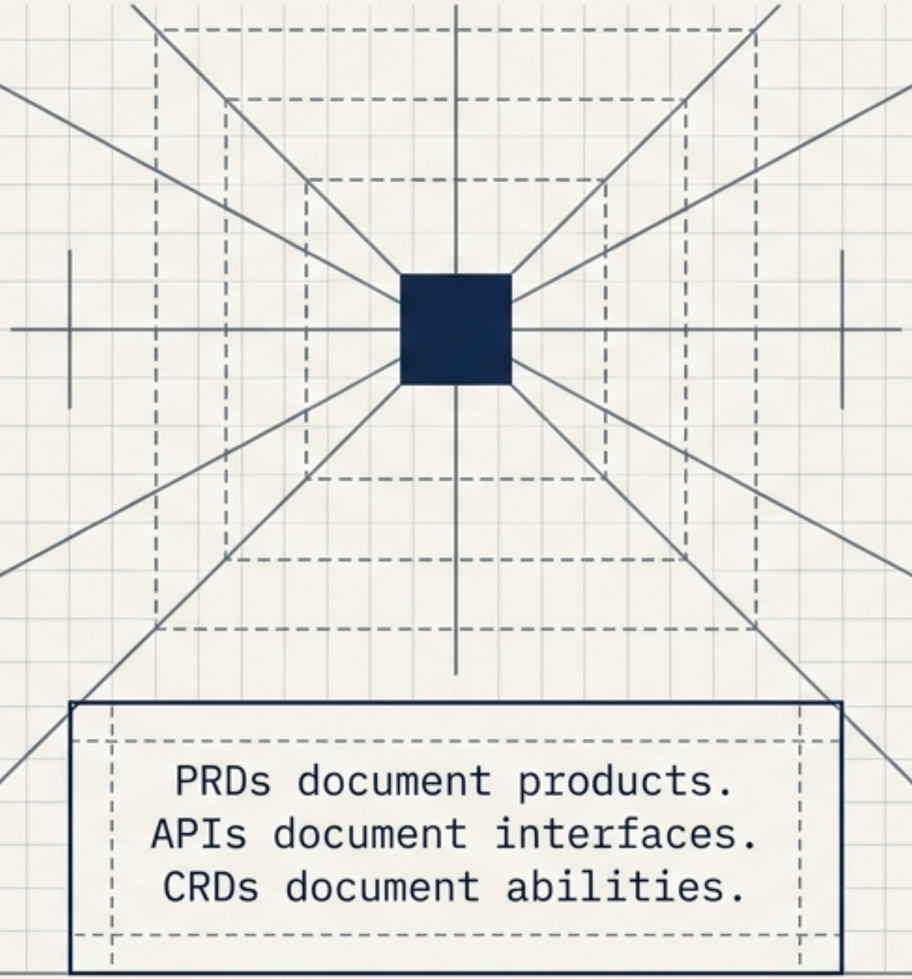
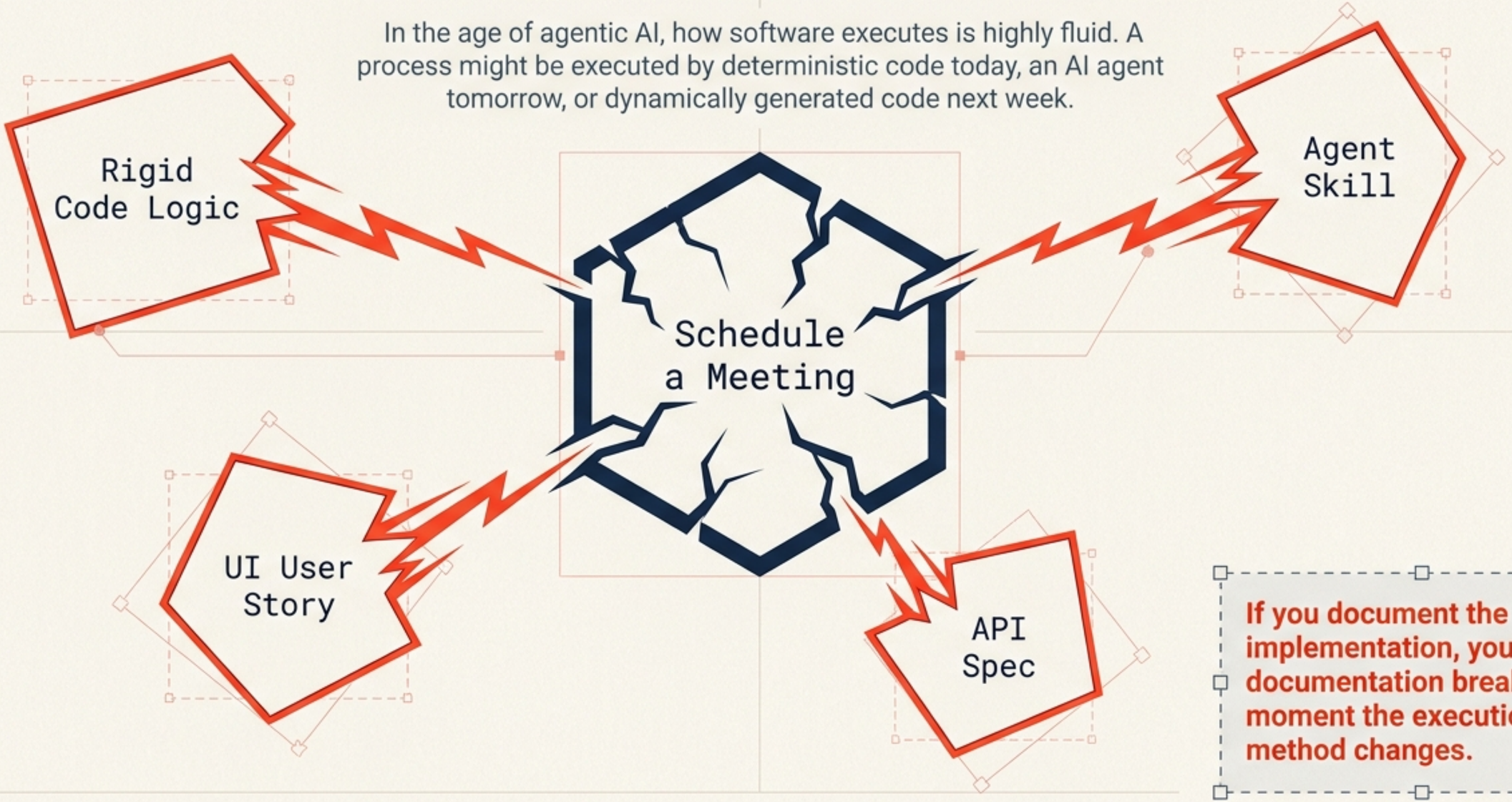
# The Capability Requirements Document (CRD)

## A Technology-Agnostic Architecture for the Agentic Era



# Implementation is no longer the most stable source of truth.

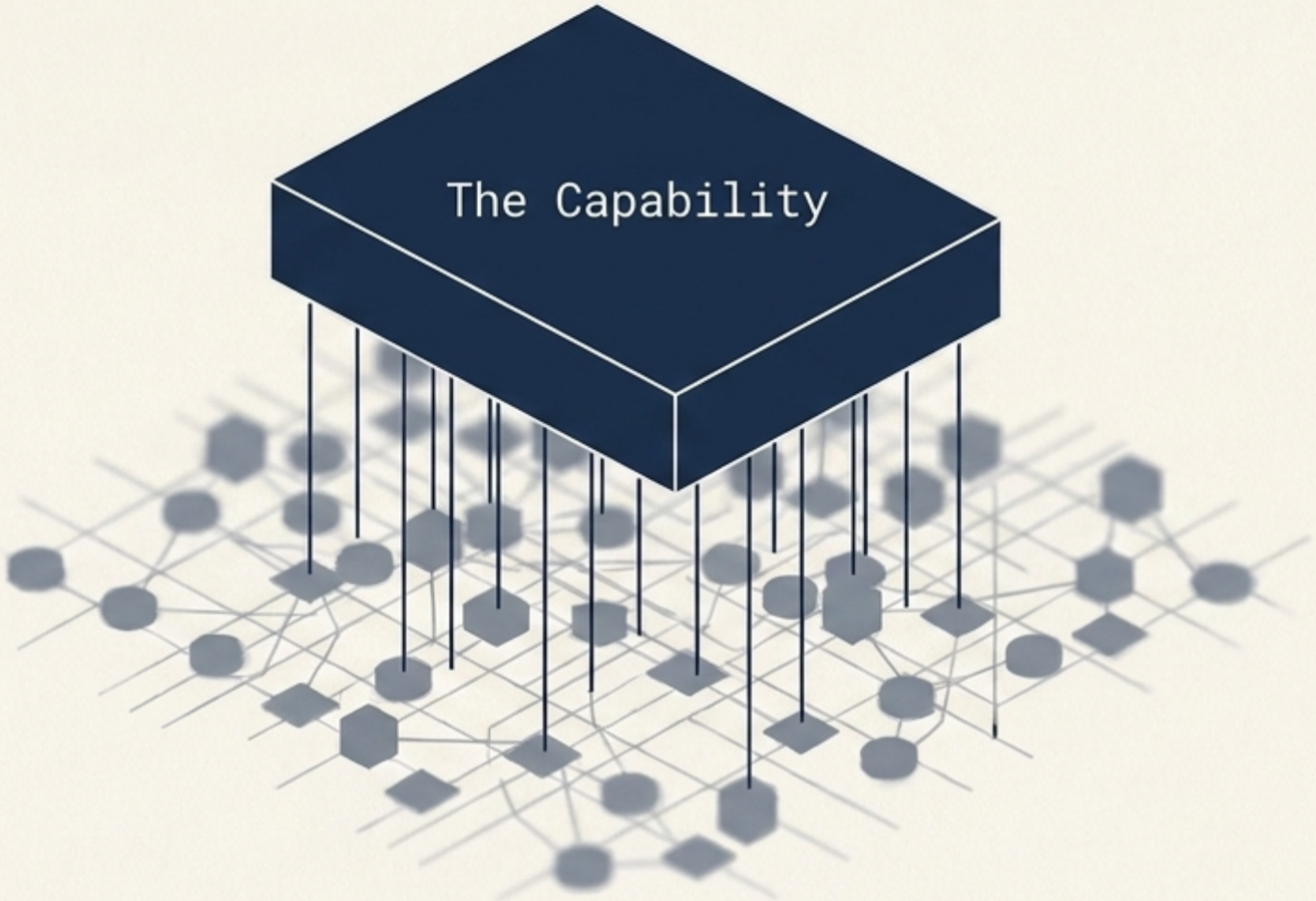
In the age of agentic AI, how software executes is highly fluid. A process might be executed by deterministic code today, an AI agent tomorrow, or dynamically generated code next week.




**If you document the implementation, your documentation breaks the moment the execution method changes.**

# The capability is the most stable object in modern software.

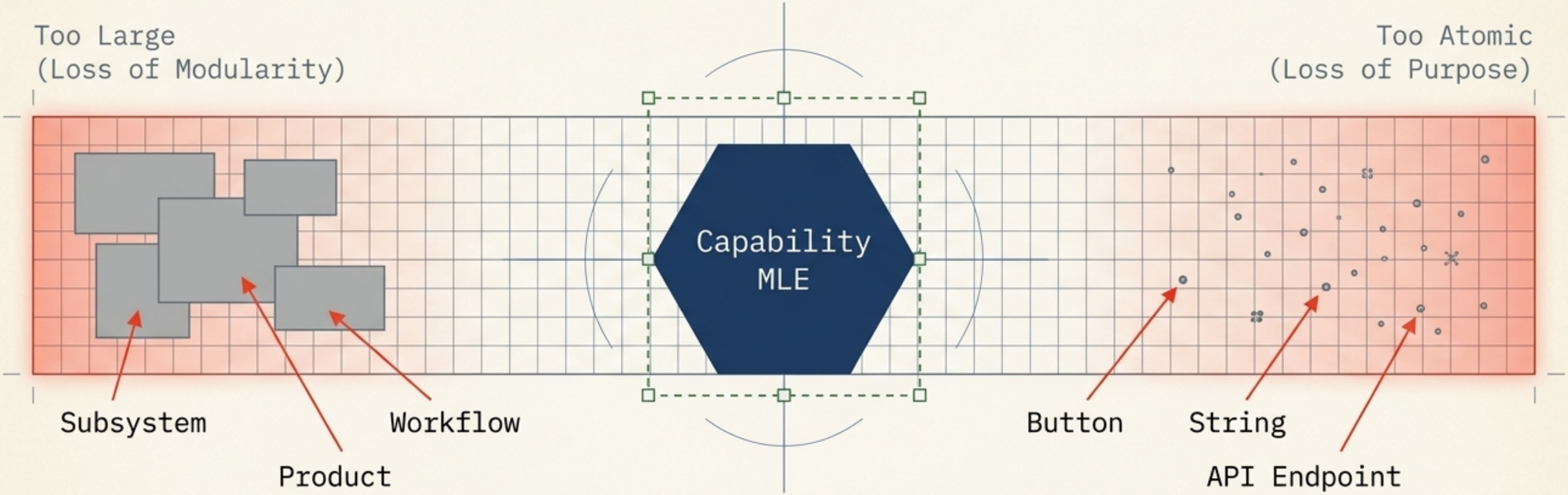
A Capability Requirements Document (CRD) documents a meaningful ability entirely independent of how it happens to be implemented.




Provides a common semantic language for:

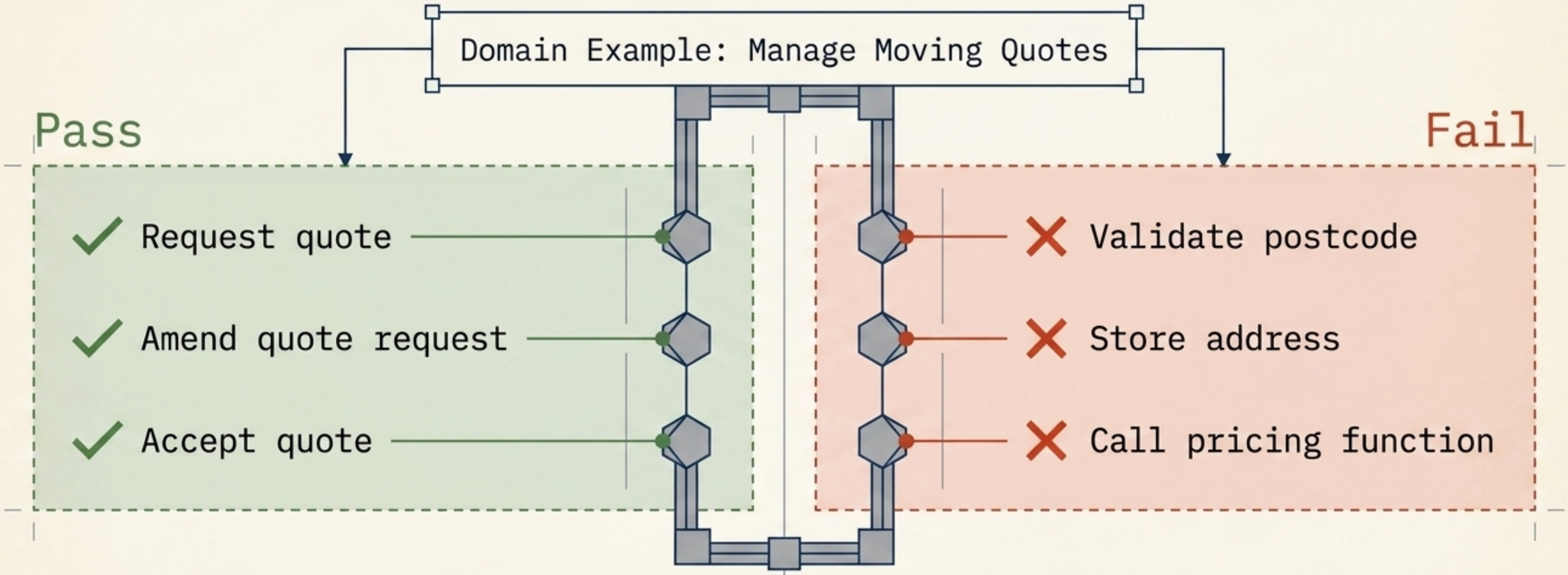
- ○ - Human Stakeholders (Business, UX, Architects)
- □ - Runtime Agents & Coding Agents
- ◇ - Future Composability Systems

# Finding the optimal boundary: The Capability MLE.




MLE (Minimum Logical Element): The smallest complete, contextually meaningful ability that produces a meaningful outcome.

# The Contextual Logical Meaning Test.




Note: These represent independently complete and contextually meaningful abilities.

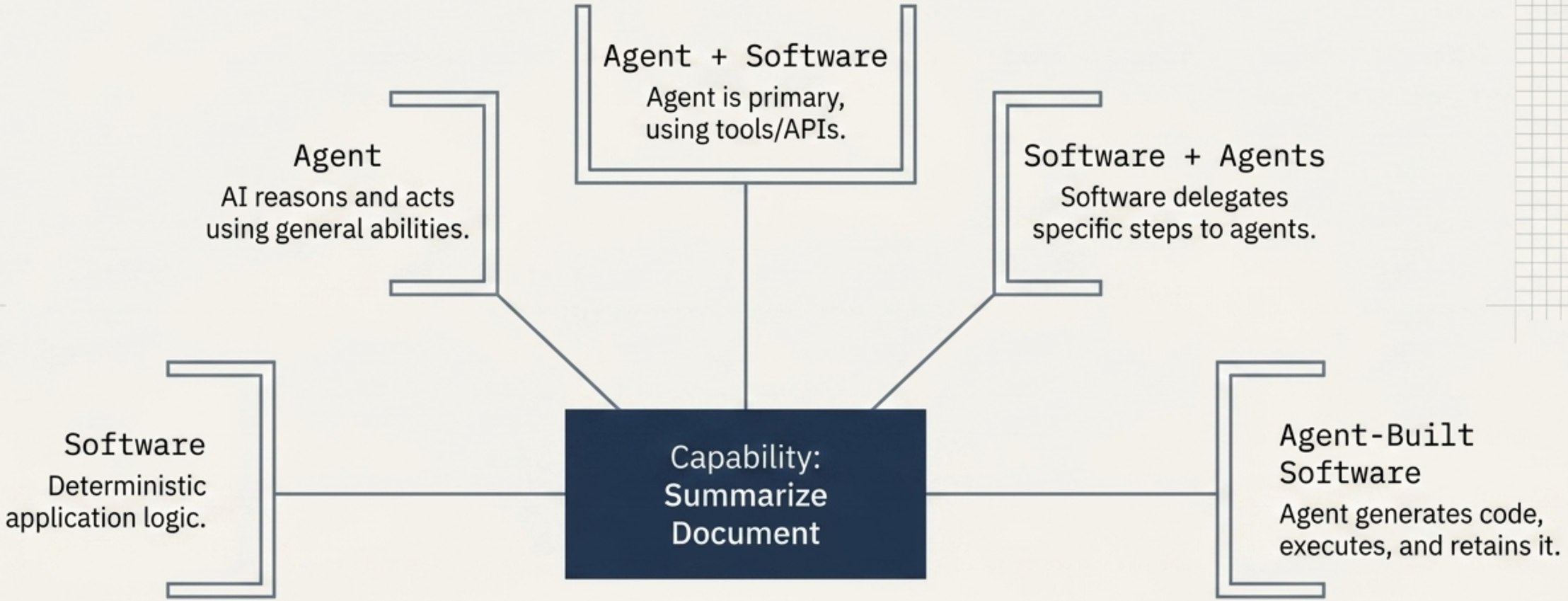
Note: These are highly reusable, but their purpose context disappears outside of a larger sequence.

# Existing documentation artifacts leave a semantic void.

| | Human Readable | Agent Executable | Implementation-Agnostic | Preserves Purpose Context |
|---|---|---|---|---|
| PRD | ✓ | ✗ | ✓ | ✓ |
| User Story | ✓ | ✗ | ✗ | ✓ |
| API Docs | ✓ | ✓ | ✗ | ✗ |
| Agent Skills | ✗ | ✓ | ✗ | ✓ |
| CRD | ✓ | ✓ | ✓ | ✓ |

Synthesis: Existing formats describe specific perspectives. Only the CRD occupies the exact middle ground required to compose software dynamically.

# Capabilities are fundamentally independent of their realization.




Capability defines *what* meaningful ability exists.
Realization defines *how* that ability is currently produced.

# Decoupling the concept from the operational instance.

Universal & Abstract

## Reusable Capability Definition

- Name:
- Purpose:
- Boundaries:
- Rules:
- Recommended Defaults:
- Interaction Contracts:

Example: “Schedule Meeting” (Implementation-blind).

Current & Specific

## Operational Capability Realization

- API Availability:
- MCP Status:
- Auth:
- Telemetry:
- System Owner:
- Realization Type:

Example: “Implemented via Google Calendar API using OAuth.”

# Humans infer context. Agents require explicit semantics.

| | |
|---|---|
| [RULE / INVARIANT] | Binding requirement.<br>*Example: Never capture a payment twice.* |
| [RECOMMENDED DEFAULT] | Preferred behavior, safely override-able.<br>*Example: Recommend three alternative times by default.*<br>**Crucial for autonomous agents to act without false invariants.** |
| [RATIONALE / INTENT] | The ‘why’ behind a choice.<br>*Example: Currency conversion added for international clients.* |
| [INFERENCE VS. FACT] | Distinguishing reverse-engineered agent assumptions from explicit source truth. |
| [UNKNOWN / UNRESOLVED] | Deliberately unspecified gaps. |

# Anatomy of a minimal Capability Requirements Document.

## The Mandatory Footprint

### Identity & Definition

Name, Capability Purpose, Meaningful Outcome.

### Boundaries

Explicitly what is inside vs. outside.

### Requirements

Rules, Invariants, Recommended Defaults.

### Contracts

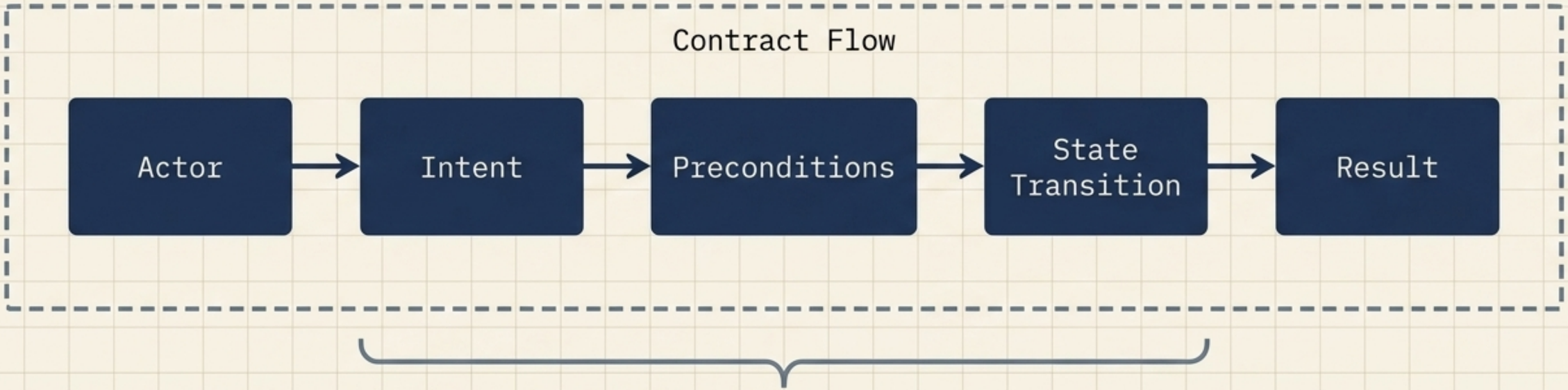
The Interaction Contracts.

As small as possible while containing everything necessary to retain contextual logical meaning.

A CRD is not a miniature PRD.

# The execution bridge:
# The Interaction Contract MLE

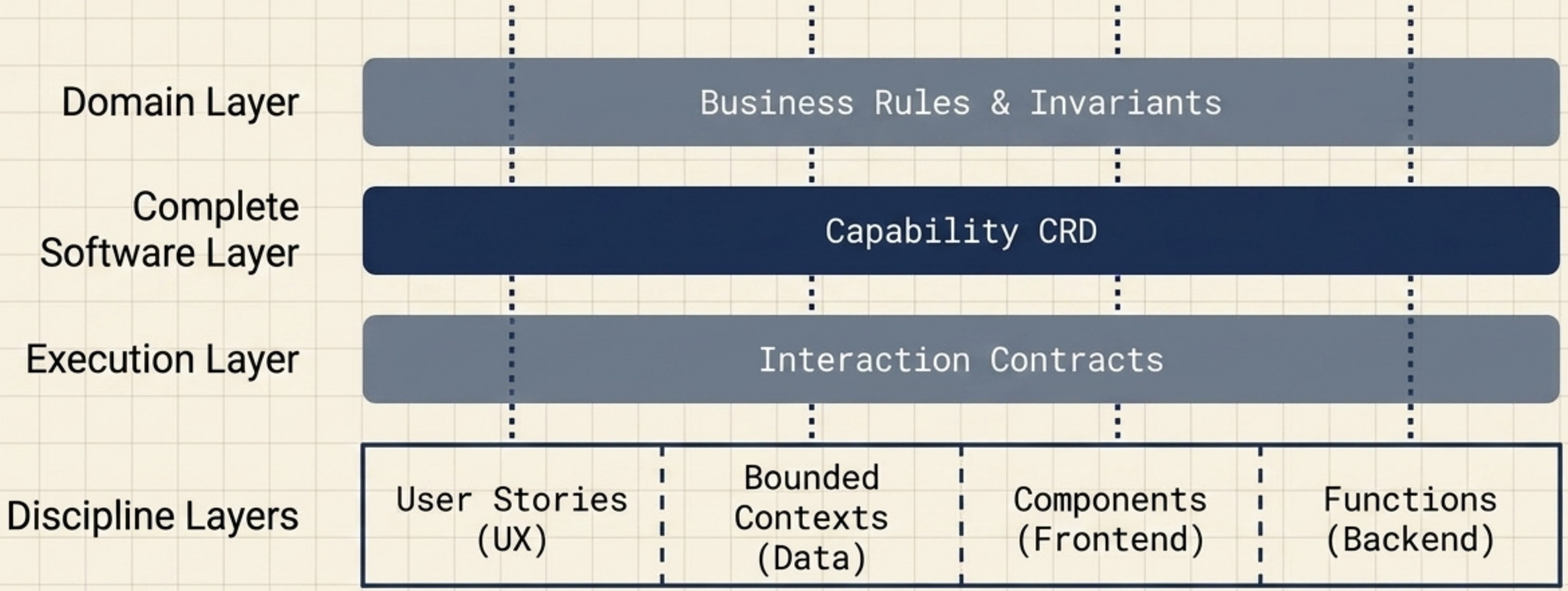
The smallest **contextually meaningful** executable behavior underneath a capability.




This neutral contract can later manifest physically as:

`GraphQL schema | MCP tool | Python function | UI action | Agent workflow`

# Traceability and contextual provenance.




Retain **purpose history**. Components remain technically reusable while their documentation remembers *why* they exist.

# **Systemic self-awareness:** Capability Inventories

Answers the ultimate systemic question for any autonomous agent: ***What can this system meaningfully do?***

## General Capability Inventory

Open, reusable catalogue of technology-independent definitions.

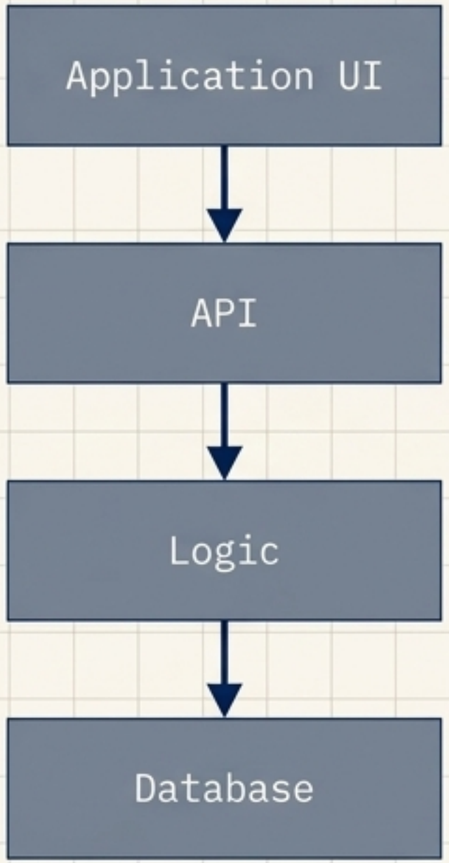
- Schedule Meeting
- Request Quote
- Summarize Document

## Service Capability Inventory

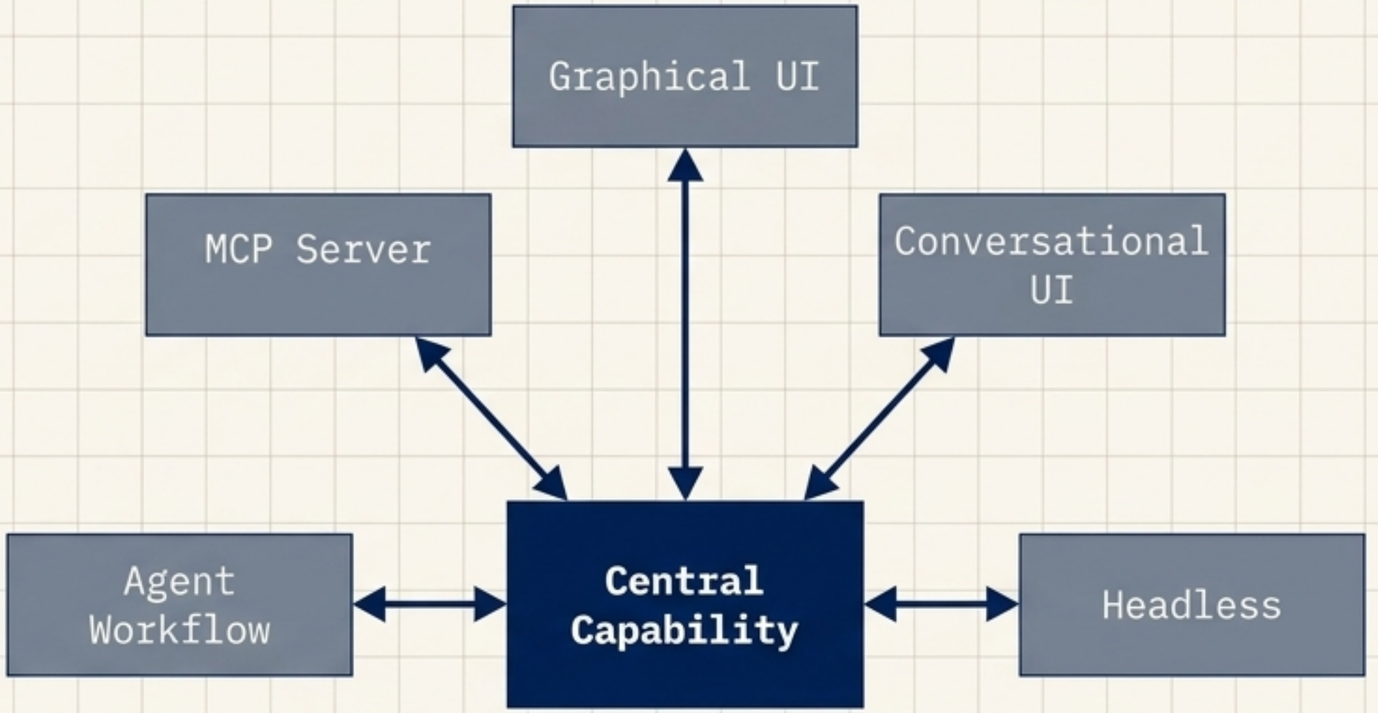
Catalogue of capabilities a particular running service actually provides.

- Operational status
- Realization type
- Supported API
- MCP exposure
- Constraints

# Capability-first architecture (UI is optional)

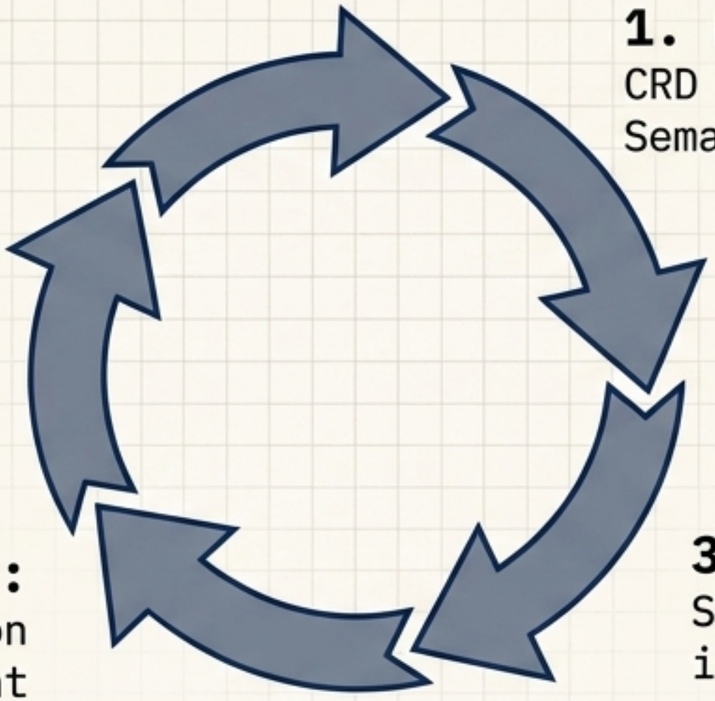



Operations are fragmented and bound to screen flows.




API and MCP design must expose meaningful outcomes, not just technically coherent but contextually fragmented operations.

# **The Future:** Documentation-Driven Development.



**1. Define:**
CRD exists as
Semantic Truth

**2. Generate:**
AI Agent uses CRD
to generate Code/UI

**3. Execute:**
Software operates
in ecosystem

**4. Verify:**
Agent verifies realization
against CRD intent

**5. Update:**
Constraints update
Realization registry

By defining *what* software can do, we create an active semantic language where humans, AI agents, and applications seamlessly discover, compose, and operate capabilities.