2026 OPEN SOURCE IN FINANCE FORUM New York

Governing the Agentic SDLC

When the architecture changes, the controls follow.

Matthew Bain and Aaron Searle

OSFF New York, 4 November 2026



SDLC

Common Controls Catalog

From an issue in July 2025 to a FINOS project

Jul 2025

The proposal: issue 261 in FINOS DevOps Automation. Duplication, drift, fragmentation.

Oct 2025

Introduced at OSFF New York. First content lands in a new FINOS repository.

• • •

Nearly 30 working-group calls and around half a dozen workshops, every two weeks since the August 2025 kickoff.

Jul 2026

Contributed to FINOS as an Incubating project.

Oct 2026

The first batch of controls approved and starting to be consumed. Every card versioned.

OCTOBER 2026 · WHERE WE ARE NOW

A FINOS Incubating project

An open standard under the FINOS Community Specification, run in the DevOps Automation SIG. Live at sdlc-common-controls.finos.org.

Built across the industry

Maintainers from Morgan Stanley, Deutsche Bank, UBS and Kosli. Fidelity, TD Bank, JFrog, Anchore, Chainloop, GitLab and KPMG among those taking part. 87 pull requests merged.

First batch approved, starting to be consumed

Code Review (v1.0) and Version Control (v1.2) are approved by the working group, with the Insider Threat risk. Nineteen more controls have passed first reading.

Open to anyone: calls every two weeks, alternating with a three-hour workshop. **sdlc-common-controls+subscribe@lists.finos.org**

Eleven risks, twenty-two controls, fifteen months later

22 controls, SDLC-PREV-001 to 022, by the lifecycle phase each applies to

1 Code — 4

- **001 Code Review**
- 005 Vulnerability Scanning: SAST
- 008 Version Control
- 010 Secret Detection

2 Build — 5

- 003 Software Artifact Provenance
- 007 Vulnerability Scanning: Dependencies
- 009 Component Inventory
- 014 Test Evidence Retention
- 016 Test Execution and Sign-Off

3 Release — 5

- 006 Vulnerability Scanning: DAST
- 012 Deployment Gating
- 019 Version Release Approval Gating
- 020 Requirements Approval for Release
- 021 Infrastructure Dependencies

4 Runtime — 1

- 017 Service Dependency Control

Lifecycle
every phase

- 004 Requirements Repository
- 011 Vulnerability Remediation SLAs
- 013 System Inventory
- 015 Testing Requirements

Meta
governance of the controls

- 002 Content Addressable Identities
- 022 Data Retention and Disposal
- NOT YET PLACED
- 018 Dependency Curation

11 risks the outcomes the controls exist to reduce

- SDLC-OP-007 Configuration Drift
- SDLC-SEC-003 Credential and Secret Exposure
- **SDLC-SEC-004 Vulnerable Software in Production**
- SDLC-SEC-006 Unauthorised System Access
- **SDLC-SEC-008 Unauthorised Change**
- SDLC-SEC-009 Environment Breach
- SDLC-SEC-010 Dependency and Transitive Supply Chain Compromise
- SDLC-SEC-011 Build Toolchain and Service Supply Chain Compromise
- SDLC-RC-001 Insider Threat
- **SDLC-RC-005 Audit and Compliance Evidence Failure**
- SDLC-BUS-012 Business Reputation Risk from Non-Approved Software Version Releases

Filled: code review, the one control this talk follows to the gate, and the three risks it answers to.

sdlc-common-controls.finos.org

Built by a FINOS working group. Every control maps to NIST SP 800-53r5 and, where relevant, EU DORA, NIST SSDF and SLSA.

All about risk

Every job we give code review is a risk we expect it to catch.

NAMED IN THE CATALOG

SDLC-SEC-008

Unauthorised Change

Code, configuration or build artefacts changed without authority, by a malicious actor or through weak controls.

SDLC-SEC-004

Vulnerable Software in Production

Software with known vulnerabilities deployed, and left running, where attackers scan for them.

SDLC-RC-005

Audit and Compliance Evidence Failure

No records, on demand, that show the SDLC governance operated as intended.

CARRIED BY REVIEW IN PRACTICE *often written down as reviewer guidance and expectations*

1. **A malicious change slips in**
   A second person sees every change before it ships.
   REVIEW AS: **INSIDER THREAT DETECTION**

2. **Defects reach production**
   A fresh pair of eyes catches what the author cannot see: readable, maintainable, fit for purpose.
   REVIEW AS: **QUALITY ASSESSMENT**

3. **Tests that prove nothing**
   Green checks only count if the tests actually exercise this change.
   REVIEW AS: **TESTING EFFICACY**

4. **Software that does the wrong thing**
   The code does what the requirement asked, and only that.
   REVIEW AS: **MEETS THE REQUIREMENTS**

5. **A big change waved through as small**
   Someone identifies the risk of the change: tier 1 or tier 2, which sets the review it needs.
   REVIEW AS: **RISK ASSESSING THE CHANGE**

6. **Only one person understands it**
   Knowledge of the code spreads beyond its author.
   REVIEW AS: **SOCIALISATION AND AWARENESS**

We are not focusing on these risks today. If you want to help shape, define or use the controls that mitigate them, **join us!**
sdlc-common-controls+subscribe@lists.finos.org

Code review, as the catalog writes it today

From here we focus back on the machinery: the review taking place, and evidence that it did. Six requirements, quoted.

1. "The policy MUST define deterministic, machine-evaluable criteria for selecting the required review mode"
2. "non-deterministic review (such as manual peer review or **approved cognitive review engines**)"
3. "pipeline controls MUST technically block admission to the protected baseline when the required review outcome is absent, expired, revoked, or **no longer bound to the current change set**"
4. "Review outcomes MUST be bound to an immutable identity for the reviewed content, such as a commit SHA"
5. "the author of the change set **MUST NOT** satisfy the human-review requirement for their own change"
6. **And most importantly, we must evidence that it took place:**
   "**Review records MUST include** the review mode, decision, reviewer identity or approved automated-review tool identity, timestamp, policy inputs sufficient to justify the selected review mode"

SDLC-PREV-001 Code Review, version 1.0, approved by the working group October 2026. sdlc-common-controls.finos.org

Where the catalog stops

The catalog says what is expected. It deliberately stops short of how to implement and enforce it.

SDLC
Common Controls Catalog

What is expected

Shared, technology-agnostic requirements. Every institution reads the same words, and cites the same version.

THE HARD PART

How do you enforce it on an agent?

It will never read the policy, and it will happily bypass it if given the chance.

CALM
COMMON ARCHITECTURE LANGUAGE MODEL

Here is one way it could be done: **by leveraging FINOS CALM.**

Author: a bot. Reviewer: a bot. Merged.

Add position total endpoint #42

Merged **coding-agent[bot]** merged 1 commit into `main`

- **review-agent[bot]** approved these changes
- All checks have passed 6 successful checks
- **coding-agent[bot]** merged into `main`

Which of your controls did this pass?

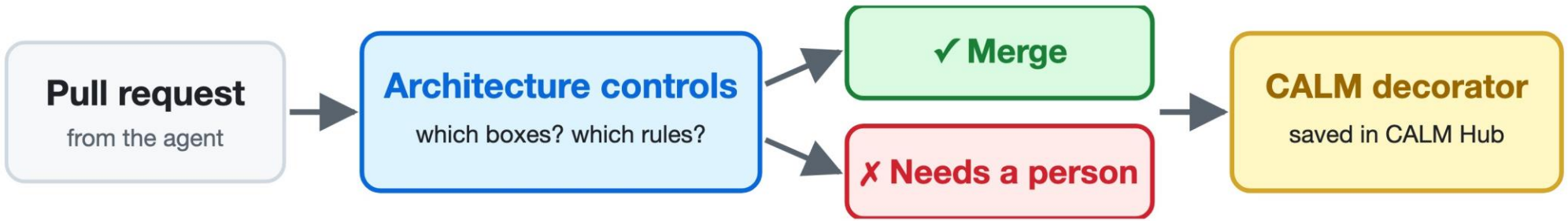
Code review? Four eyes? Change approval?

Your architecture already knows which changes are risky.

It's just trapped in a picture.

Every pull request is checked against the architecture

The rules live on the architecture as **CALM controls**.



CALM
COMMON ARCHITECTURE LANGUAGE MODEL

What a CALM control is

A named rule written into the architecture file, next to the thing it governs. It points at the requirement it implements, here a control from the SDLC catalog, and carries its own settings.

On the architecture	applies to every change `"controls": { "sdlc-prev-001-code-review": … }`
On a node	the Client Ledger's rule `client-ledger › "controls": { "client-ledger-human-approval": … }`
On a relationship	a rule for one connection `a relationship › "controls": { … }`
On a flow	a rule for one business journey `a flow › "controls": { … }`

[.] ergo the gate shows its working

An open source library for **Open Policy Agent**. The policy is written as data, inside the CALM control, and every decision comes with its evidence.

THE RULE, READABLE

```
subject: connection to the client ledger, changed
  of: every connection in the architecture
  applies_to:
    into_the_ledger: destination = client-ledger
    from_a_changed_node: source in change.nodes
checks:
  human_approved: "A human approved this change"
  approver_is_not_the_author
  approval_for_this_commit
```

Plain Rego would hide this in rules only a policy engineer reads. Here an architect or an auditor can read it.

THE DECISION, WITH ITS EVIDENCE

Plain Rego `{ "allow": false }` **But why?**

ergo

subject	check	read	cause
trade-api-to-client-ledger	human_approved	approver = null	absent
trade-api-to-trade-store	$applies	dest = trade-store	out of scope

One row per subject and check: the values it read, and why it passed, failed or was out of scope. Same policy and input, the same report, byte for byte. That report is what goes back onto the architecture.

[.] ergo report, saved on the architecture

Each decision becomes a **CALM decorator**: a standard way to attach facts to an architecture without editing it.

```
{
  "type": "control-evaluation",
  "target": [".../architectures/33/versions/1-1-0"], (1)
  "applies-to": ["trade-api", "trade-api-to-client-ledger"], (2)
  "data": {
    "control": { "unique-id": "client-ledger-human-approval",
                 "control-id": "SDLC-PREV-001" }, (3)
    "status": "not-met",
    "findings": [{ (4)
      "subject": "trade-api-to-client-ledger",
      "check": "human_approved",
      "description": "A human approved this change",
      "cause": "absent" }],
    "change": { "commit": "a27142e…", "spec": "SPEC-001" }, (5)
    "engine": { "name": "ergo", "policy-sha256": "76c558b1…" } (6)
  }
}
```

1. **Which architecture** it was decided against: version 1.1 in CALM Hub
2. **Which boxes and lines** it is about, so the Hub can colour them
3. **Which rule**, and the catalog control it comes from
4. **ergo's report**, in its own words: what was checked, and why it failed
5. **Which change**: the exact commit and its spec
6. **Which policy** decided, by its hash

How a CALM control feeds ergo

The control's **config** is the ergo policy. Nothing is translated or copied.

IN THE ARCHITECTURE FILE, ON THE CLIENT LEDGER

```
{ "unique-id": "client-ledger",
  "controls": {
    "client-ledger-human-approval": {
      "requirements": [{
        "requirement-url": ".../sdlc-prev-001-code-review",
        "config": {
          "ergo": {
            "subjects": {
              "connection to the client ledger, changed": {
                "applies_to": {
                  "into_the_ledger": destination = client-ledger,
                  "from_a_changed_node": source in the change }}},
            "requirements": {
              "client-ledger-human-approval": {
                "subject": "connection to the client ledger, changed",
                "checks": {
                  "human_approved": a person approved }}}}
```

Controls can sit on the architecture, a node, a relationship or a flow. A generic collector reads every one and merges their `ergo` blocks into one policy.



Let's see it live.

An agent opens pull requests on a real GitHub repository. The architecture decides which ones need a person.

Built from open parts

SDLC
Common Controls Catalog

What to control

The SDLC Common Controls Catalog, written once by the industry.

CALM
COMMON ARCHITECTURE LANGUAGE MODEL

Where it applies

FINOS CALM: the architecture as code, the rules on its boxes, the decisions back on it as decorators.

[.]ergo

How to decide

An open source project that checks rules written as data and says why.

The SDLC controls you just saw enforced

SDLC-PREV-001 Code Review
Reviewed by a bot that is not the author; a person approves changes near the Client Ledger and changes to the architecture

SDLC-PREV-004 Requirements Repository
Every change names an approved spec and touches only the services that spec declares

SDLC-PREV-017 Service Dependency Control
Every connection the code makes is in the architecture: the shortcut to the ledger was blocked

Everything you saw is real: the repository, the pull requests, the approvals, the checks and CALM Hub.

Pick one rule. Put it on one box. Let the architecture say no.

The demo
github.com/rocketstack-matt/osffny-sdlc-calm

The blueprint
github.com/rocketstack-matt/agentic-sdlc-blueprint

The catalog
sdlc-common-controls.finos.org

CALM
calm.finos.org

ergo
ergo.kosli.com

Your feedback
Rate this session