# Mechanical Gates over Model Consensus

## Evidence-Governed Harness Engineering for Agentic Software Development

**Governing the agentic software development lifecycle with executable evidence —a TigerAI working paper**

---

**Author**: Yeh-Hsing Lu (Morris) —Tiger AI Tech Co., Ltd., Taiwan ·**ORCID**: 0009-0006-5373-0586 **Type**: White Paper ·Research Framework | **Version**: v3.4 (retitled; framework described plainly) | **Date**: July 19, 2026 **Status**: Working paper —conceptual framework + research agenda; includes one preliminary field observation. Controlled experiments (EXP-001/002) are not yet completed, so **no claims are labeled `[validated]`** in this version.

**Suggested citation**: Lu, Y.-H. (2026). *Mechanical Gates over Model Consensus: Evidence-Governed Harness Engineering for Agentic Software Development* (v3.4, Working Paper). Zenodo (DOI to be minted).

**Disclosures**: We situate this work within **harness engineering** —the control/verification layer built around coding agents (cf. Böckeler, *Harness engineering for coding agent users*, 2026; OpenAI, *Harness engineering: leveraging Codex in an agent-first world*, 2026); our contribution is the **evidence-governance** integration of executable gates, human evidence review, rollback, and skill capitalization. We do **not** claim to originate the term *harness engineering*. Produced with a multi-engine AI workflow (Claude / Codex / Gemini); the human author set the research questions, supplied the field evidence, and **verified the cited references against sources** (including the non-arXiv MCP and Data Science Dojo references, checked 2026-06-22), and is solely responsible —AI systems are **not** authors. Claim-level labels (`[design principle]`/`[engineering spec]`/`[research hypothesis]`/`[preliminary evidence]`/`[validated]`) are used throughout; the full AI-role breakdown is given in the Generative AI Use Disclosure below.

---

## Abstract

Large language models (LLMs) can produce runnable code in a single interaction, yet enterprise software delivery requires not "runs once" but **governable, auditable, reversible, and repeatable** engineering capability. This paper proposes an **evidence-governed approach to *harness engineering*** —the control-and-verification layer built around coding agents (an emerging 2026 practice) —for the full agentic software development life cycle (SDLC), mapping it onto an autonomous, multi-agent pipeline. It rests on three claims: (1) a **closed full-lifecycle loop**; (2) **knowledge capitalization**, whereby each completed project yields reusable, testable, governable *Skill* units governed by a finite-state machine; and (3) **anti-self-deception governance**. We study a failure mode in which code-generating and verification agents that share the same implementation context or bias source may produce a **shared-context false pass** (which we earlier called *test collusion*). We **hypothesize** that multi-model cross-checking reduces *variance* but, under shared bias, may not remove systematic *bias*; this is stated as a research hypothesis pending direct measurement (EXP-001), not as established fact. The framework therefore **privileges executable evidence over model consensus** at release decisions: a **mechanical gate** (mutation testing, coverage, property-based testing, spec invariants, real execution) supplies executable, reproducible evidence that is less dependent on model opinion (strong evidence, but not by itself a correctness oracle); a **model panel** is used only at judgment-bearing checkpoints; and **humans review evidence rather than consensus**. Notably, recent empirical work [arXiv:2512.03097] shows that multi-agent *consensus* alone is not safety, whereas a **verifier anchored to trusted external guidelines** effectively defends —consistent with our position that a verifier's value comes from external evidence, not from being yet another model. As a **preliminary field observation**, we extend the framework to **vibe-coded applications** —re-expressing opaque AI-generated code as inspectable, skill-verified **workflow-native harnesses** so that non-engineer auditors can govern them in regulated settings (§4.4). We give an engineering specification grounded in a workflow-native state machine and a JSON-RPC inter-agent protocol, with n8n used in our reference implementation, and discuss

implications for data sovereignty, prompt-injection defense, and compliance in regulated, on-premises settings such as healthcare.

**Keywords**: agentic AI; software development life cycle; multi-agent systems; LLM code generation; verifiability; mechanical gates; spec invariants; human review; skill capitalization; shared-context false pass; DevSecOps; data sovereignty

---

# 1. Introduction

## 1.1 Motivation

Mainstream "AI writes code" practice reduces to `Prompt → Code → Debug`. This works for prototypes but, at enterprise scale, exposes four structural defects —unpredictable output, no architectural review, runaway technical debt, no knowledge accumulation —and lacks an auditable **evidence trail** throughout. The real bottleneck is not tool power but **whether human engineering capability and governance are in place**. The question is therefore not "make the AI write better code," but "**let AI take over the software production line within a governable framework**." A parallel shift sharpens this gap: *vibe coding* has democratized application generation —non-engineers can now produce working applications —yet governance remains bottlenecked by **code opacity**, because the generated native code is often unreadable to the very operators expected to review, audit, and maintain it (§4.4).

## 1.2 An overlooked new risk

When verification itself is handed to AI agents, a latent risk appears: **the AI verifying the AI is also an AI**. If one agent writes code and another writes tests *for that code*, both converge on "all tests green" while substantively verifying nothing. We **name and situate** this failure mode within full-SDLC governance (§7); it is **not** solved by "adding more models to cross-check."

## 1.3 Contributions

1. **Framework formalization**: SDLC cast as a multi-agent pipeline with artifacts, acceptance gates, and feedback loops (§3–4).
2. **Skill-capitalization model**: project experience formalized into reusable, testable, governable Skill units with an FSM lifecycle and an anti-degradation curation algorithm (§5–6, §9.2.1).
3. **Anti-self-deception verification**: we frame **shared-context false pass** within full SDLC governance and make **executable evidence (not more model consensus)** the admission/release gate, layered with model disagreement and human evidence review (§7).
4. **Enterprise & AI-specific security spec**: agent permission model, skill supply-chain security, pipeline threat model, prompt-injection/agent-hijacking defense (§5, §8).
5. **A falsifiable research design**: explicit research questions and an evaluation plan (§1.4, §10).
6. **Workflow-native harnesses for vibe-coded applications** `[preliminary field observation]`: re-expressing repeatable, judgment-bearing steps as inspectable, skill-verified workflow graphs to improve the enterprise governability of vibe-coded applications in regulated settings (§4.4, C9).

## 1.4 Research questions

- **RQ1 (existence)**: Does shared-context false pass cause "tests pass but spec unverified," and at what rate/severity?
- **RQ2 (mechanical gate efficacy)**: Do mechanical gates (mutation, coverage, property-based, spec invariants) reduce escaped defects more effectively than multi-model review?
- **RQ3 (capitalization payoff)**: Does a Skill Repository reduce rework, lead time, and error rate on later projects?

- **RQ4 (auditability & workflow-native governability)**: Do Evidence/Policy Repositories —and re-expressing application logic as inspectable, skill-verified workflow graphs (§4.4) —improve the auditability and governability of AI-produced software?
- **RQ5 (human boundary)**: In regulated industries, which stages must keep a human gate, and which can be safely automated?

### 1.5 Novelty statement

We **explicitly do not** claim to have invented agent loops, loop engineering, the SDLC, guardrails, mutation testing, or the *phenomenon* of code/test collusion —the last is already reported as *self-collusion* in **Code-A1 [arXiv:2603.15611]** (§2.7). Our **candidate contributions** are **integrative**, not first-of-kind: (1) **Agentic SDLC** —SDLC stages mapped to an executable, reviewable, reversible multi-agent line; (2) **Skill Capitalization** —experience formalized into a testable, governable Skill Manifest and lifecycle; (3) **Evidence-Governed Verification** —placing "shared-context false pass" inside **full Agentic SDLC governance**, using **mechanical evidence (rather than more model consensus)** as the admission/release gate, with model disagreement, human evidence review, an evidence repository, rollback, and skill lifecycle integrated.

> "Candidate" because novelty must be confirmed by EXP-001/002 and a systematic literature review (§2.7, §13.6). Tagline: *From model consensus to executable evidence —an evidence-governed lifecycle for agentic software development.* Difference from Code-A1: Code-A1 addresses "how two models (code/test) avoid colluding"; this paper addresses "how the **whole SDLC** is governed by executable evidence —auditable, reversible, and accumulative."

### 1.6 Related and concurrent work (2025–2026)

Concurrent efforts address facets of this problem. Advani [2026] characterizes *false success* —agents asserting completion while the environment disagrees, with LLM judges anchored by confident closing language —a prior characterization of the failure our gates target. Harness-MU [Fan et al. 2026] likewise argues that governance constraints should be deterministic runtime hooks rather than trusted to the LLM, but for multi-user runtime, not the software lifecycle. VeriGuard [Miculicich et al. 2025] enforces safety via formally verified, action-level policies; SAGA [Ma et al. 2025] shows that LLM-generated tests can homogenize toward the producing model's blind spots; a deterministic control plane for coding agents [Madatha 2026] adds permission tiering and audit at the agent-configuration layer; and governed capability-evolution and skill-lifecycle frameworks [Qin et al. 2026; SkillsVote, Liu et al. 2026] already stage, gate, and roll back reusable capabilities. We do **not** originate these mechanisms; our contribution is their **integration into a single full-SDLC evidence-governed harness** —mechanical gates over model consensus, project-capitalized skills with a named-state lifecycle, and (preliminary) workflow-native governance —which none of the above provides. A full comparative treatment is deferred to a later version.

---

## 2. Background & Related Work

### 2.1 DevOps / DevSecOps

We continue DevOps/DevSecOps principles of shift-left security and metric-driven improvement [Kim 2016; Forsgren 2018]. Difference: the executors here are governed AI agents, so we must additionally handle trustworthy verification of autonomous agents.

### 2.2 Agentic / multi-agent LLM systems

ReAct [Yao 2023], multi-agent debate [Du 2023], and self-consistency [Wang 2022] show multi-step collaboration potential. We argue (§7) these mostly improve variance, not systematic bias, and cannot serve as ground truth at verification gates.

## 2.3 Mechanical test-adequacy measures

Mutation testing [DeMillo 1978], property-based testing [Claessen & Hughes 2000], and metamorphic testing [Chen 1998] provide judgment-independent measures. We use these as partial oracles at gates (§7.3, §7.7).

## 2.4 Ensemble learning & estimator correlation

Ensemble efficacy depends on member diversity/independence [Dietterich 2000]; we use this to frame the (hypothesized) correlation among LLM estimators (§7.2).

## 2.5 Threat modeling & skill interoperability

STRIDE-style threat modeling [Shostack 2014] for shift-left security; Model Context Protocol (MCP) [Anthropic 2024] as a candidate skill-packaging interface.

## 2.6 LLM-as-judge bias & agentic SWE evaluation

LLM judges exhibit self-preference / position / verbosity bias [Zheng 2023] —**but the same work notes such bias can be mitigated**, so one cannot infer "all model panels are useless." SWE-bench [Jimenez 2023] measures *generation*; we are complementary (post-generation governance/verification).

## 2.7 Loop engineering, agentic SWE, test oracle, and multi-agent collusion (how we differ)

Practitioners define **loop engineering** as designing/maintaining an agent's plan–act–observe loop with guardrails [Data Science Dojo 2026; LangChain 2026]. [LangChain 2026] frames it as **stacking control/feedback loops** around a base agent (agent → verification → event-driven → hill-climbing); our lifecycle realizes the same stack at **full-SDLC scale** (§4), but at the **verification** layer it replaces a model-grader rubric with **executable mechanical evidence** (§7) —precisely because a model grader can itself produce a shared-context false pass. This sits within **harness engineering** —building the control/verification layer around a coding agent; we locate our contribution in the **evidence-governance** integration (executable gates, human evidence review, rollback, skill capitalization). Our **shared-context false pass** (earlier "test collusion") is adjacent to existing topics —a **specialization/integration, not a new discovery**:

- **Direct prior art —code/test self-collusion** [Code-A1, arXiv:2603.15611, 2026-03]: shows that white-box single-model self-play yields *self-collusion* (trivial tests for easy reward); their fix is to **separate Code-LLM and Test-LLM with opposing objectives**. This overlaps strongly with our PG↔V&V problem and "spec-isolated test generation" remedy; **we no longer claim naming/specialization originality**, repositioning to "integrate it into full Agentic SDLC governance."
- **Test oracle problem** [Barr 2015]: our spec invariants are a partial-oracle use, not new.
- **Common-mode failure / test-suite overfitting**: related concepts.
- **LLM-as-judge bias** [Zheng 2023]: systematic but mitigable.
- **Multi-agent collusion**: [arXiv:2512.03097] (healthcare) shows multi-agent **consensus** is itself unsafe (attack success up to 100%), but a **verifier anchored to trusted external guidelines drives it to 0%**. **Correct reading: the verifier works because it is anchored to external evidence, not because it is another model** —this directly supports our "executable evidence over model consensus" thesis. ([arXiv:2603.20281] is weakly related, on pricing-algorithm collusion; [arXiv:2510.04303] steganographic collusion is a side reference.)

**Candidate differentiation matrix** (cells to be confirmed by full reading; not presupposing we win):

| Work / area | Problem | Verifier is AI? | Handles shared bias? | Executable oracle? | Full SDLC? | Skill governance |
|---|---|---|---|---|---|---|
| Test Oracle survey [Barr 2015] | oracle absence | no | partial | many | no | no |
| LLM-as-Judge bias [Zheng 2023] | judge bias (mitigable) | yes | yes | no | no | no |
| Agentic SWE Roadmap [2509.06216] | agentic SWE roadmap | yes | no | partial | partial | no |
| **Code-A1 [2603.15611]** | **code/test self-collusion** | yes | yes | adversarial tests | no (code/test only) | no |
| Multi-agent collusion [2512.03097] | collusion (verifier+external defends) | yes | yes | external guideline | no | no |
| **This paper (candidate)** | Agentic SDLC **governance integration** | yes | yes (hypothesis) | mechanical evidence | yes | yes |

**Search reproducibility**: literature surveyed 2026-06-21; arXiv entries verified via the arXiv API and the non-arXiv references (MCP, Data Science Dojo) verified 2026-06-22. A full **systematic** review (exhaustive query log + inclusion/exclusion criteria) is still recommended before formal journal submission (§13.6).

---

# 3. Framework Definition

## 3.1 Core definition

An evidence-governed, auditable engineering cycle that lets AI agents proceed from context intake, requirements, analysis, design, coding, verification, security, deployment, to monitoring. Each completed project is distilled into reusable, verifiable, governable Skill units; after validation they enter the Skill Repository so later agents can modularly assemble new production lines.

## 3.2 Positioning

| Facing | Positioning |
|---|---|
| Internal (engineering) | Agentic SDLC —the AI agent's software lifecycle |
| External (strategy) | AI-Native Software Factory OS |

### 3.3 Three orthogonal layers

`Process Layer (stages/gates/feedback) + Artifact Layer (versioned auditable deliverables) + Skill Layer (reusable skills feeding the next round).`

### 3.4 Careful phrasing

We deliberately avoid "AI can fully autonomously produce systems." The accurate claim: AI agents can **incrementally raise** automation under an **auditable, verifiable, reversible** human-in-the-loop framework.

### 3.5 Scope boundary

**Applicable**: enterprise internal systems, AI workflow/automation, RAG/agentic apps, DevSecOps automation, on-prem/regulated delivery. **Not applicable (or heavy human involvement)**: exploratory tasks with no definable acceptance criteria (mechanical gates lose their upstream ground truth, §12.2); environments with no test/deploy permission boundaries; high-risk production deployment without a human gate; black-box automation with no evidence repository.

---

## 4. The Harness Loop

> **The harness as stacked loops**: it realizes "loop engineering" in the [LangChain 2026] sense —control/feedback loops stacked around a base agent —at full-SDLC scale: **(L1) agent loop** = per-stage agents (PG, etc.); **(L2) verification loop** = the V&V mechanical gate (§7), but anchored to *executable evidence* rather than a model-grader rubric; **(L3) event-driven loop** = the workflow-native state machine and feedback→requirement signals (§9.1); **(L4) improvement loop** = the §9.5 meta-loop (Kaizen) plus skill capitalization (§6). Its distinctive move is at L2: it does **not** trust another model's judgment as the gate.

### 4.1 Full flow

The main flow uses **Git as the evidence backbone**, forcing each stage to commit a versioned deliverable.


Figure 1: The harness loop —an evidence-governed SDLC loop (Context Intake → Requirement → ⋯→ Skill Repository, with feedback loops).

### 4.2 Deliverables & gates (excerpt)

| Stage | Deliverables | Gate |
|---|---|---|
| Context Intake | `context.md` | enough to avoid "technically feasible but commercially wrong" |
| Requirement | `requirement.md`, `acceptance_criteria.md`, **`spec_invariants.md`** | acceptance criteria defined; machine-checkable invariants produced (§7.7) |
| SA | `analysis.md` | risks/boundaries clear |
| SD | `architecture.md`/ADR, `threat_model.md`, `data_classification.md` | rationale; threat model + data classification |

| Stage | Deliverables | Gate |
|---|---|---|
| PG | source code | style/module compliance |
| V&V | `test_report.json` | **mechanical gate** (§7.2/§7.7) passes |
| Security | `security_report.json` | secret/RBAC/dependency/exfiltration scans clean |
| Reviewer/Human | `review_report.md`, `approval_record.md` | major debt? human decision needed? |
| Deployment | `deploy_report.md`, `rollback_plan.md` | health check; rollback plan |
| Monitoring | `monitoring_spec.md`, `alert_rules.md` | key events traceable |
| Feedback | `retrospective.md` | extractable skills flagged |
| Skill Extraction | `skill_manifest.yaml` | callable by other agents; verifiable |

## 4.3 Iron rules

```
No evidence, no release.
No validation, no skill promotion.
No rollback plan, no production deployment.
No spec invariant, no acceptance.
```

## 4.4 Vibe Coding Creates Applications; Harness Engineering Makes Them Governable `[preliminary field observation]`

*Vibe coding* lowers the barrier to application generation, but it does not by itself provide enterprise governability. **Harness engineering is the control, verification, and operational layer wrapped around AI-generated applications** —vibe coding *creates* the applications; harness engineering makes them *governable*. The resulting "vibe coding governance gap"is by now recognized: the generated native code is frequently a **black box** —opaque to the business operators and IT reviewers expected to review, audit, and maintain it —creating shadow-IT and unmanaged attack-surface risk [CSA 2026; Dataiku 2026]. Independently, treating a **workflow / orchestration layer as a governance control plane** —with identity, RBAC, and audit embedded in the workflow rather than bolted on afterward —is an established enterprise pattern [Okta 2026]. We claim **neither** as our discovery.

Building on our own prior system work —a RAG pipeline in which visual workflow graphs proved more inspectable to enterprise reviewers than the equivalent native code [Lu & Yang 2026] —we **generalize** that observation into the harness framing of this paper: a *workflow-native harness* re-expresses application behavior as inspectable workflow graphs, so that governance (identity/RBAC, credential boundaries, execution logs, source control/environments, deployment) can be attached at the **workflow layer**, where the same mechanical gates and evidence trail (§4, §7) apply.

Our specific `[preliminary field observation]` comes from regulated settings —banking and public-sector organizations under internal-audit and internal-control mandates —where business units increasingly build their own applications (citizen-developer / vibe-coding democratization), and the native code is opaque to the internal-audit and control functions expected to review it. In the observed case three pieces operate together, and the case exercises this paper's own machinery end-to-end:

- **(1) Creation.** Business units use vibe coding to *do* the work quickly —for example, financial-product research and prototyping. This lowers the barrier but yields opaque native code.
- **(2) Governed conversion.** The **repeatable, judgment-bearing** steps that emerge —a QC pass/fail decision, a product-review approval chain, a customer-request routing rule —are re-expressed as inspectable workflow graphs. This conversion is not a mere translation but an *evidence-governed pass*: a reusable **Skill** (§6) applies the mechanical and security gates (§7, §8) as **skill-based verification units**, checking code efficiency, security, permission boundaries, data-exfiltration risk, and operational maintainability before

the workflow is accepted. Acceptance rests on **mechanical gates** —evidence, logs, tests, scans, policy checks —not on a model asserting the output "looks fine."
- **(3) Governance.** The workflow layer then supplies the enterprise governance controls for the recurring judgment: identity/SSO, RBAC, audit logging, CI/CD, high availability, and credential boundaries —and, valued most by non-engineer auditors, a representation they can **visually follow**, tracing a transaction path directly on the graph.

The division of labor is deliberate: vibe coding keeps creation fast, the Skill-driven gates make the *conversion itself* accountable, and the workflow layer makes the *judgment* controllable and auditable —rather than rewriting every artifact as a workflow.

Crucially, the requirements here were **set by the customer's control functions, not by any tool**. We therefore state the target abstractly. Define a **governance-grade workflow platform** as a general-purpose, self-hostable workflow/orchestration system that (i) exposes application logic as an **inspectable graph a non-engineer can visually follow**, and (ii) natively provides verifiable identity and least-privilege access (RBAC), a tamper-evident execution audit trail, external secret management, versioned and environment-separated change control, in-process human-approval steps, and on-premises deployment for data sovereignty. These are **governance requirements derived from the control functions, not a product specification** —several platforms could satisfy them. Internal audit and control specified requirements that map onto this class; in the case we observed, a single such platform satisfied the observed control requirements at the workflow layer, where the same mechanical gates and evidence trail (§4, §7) apply.[1] The contribution is that the control functions' requirements were met at the workflow layer —not the selection of a product. A platform of this class (general-purpose, well-documented, self-hostable) also appears, in our observation, more amenable to AI-assisted generation and conversion than a bespoke in-house system, which typically lacks such controls to begin with. We therefore hypothesize —as a `[preliminary field observation]`, not a `[validated]` result —that **workflow-native harnesses improve the enterprise governability of vibe-coded applications** by converting opaque generated code into inspectable workflow graphs and applying skill-based checks for performance, security, permission boundaries, and operational readiness. These affordances still require deliberate configuration; this remains a preliminary, anonymized field observation, not a validated comparison, and a comparative case study is future work (§10; C9).

---

# 5. Five Core Modules

1. **Governed Process** —Git-backed staged pipeline.
2. **Repository System** —Project / Skill / Evidence (append-only + hash chain, secret-redacted) / Policy repositories.
3. **Agent Roles** —Orchestrator; per-stage agents; an **independent Reviewer agent** (producer ≠ reviewer); a **Skill Curator agent**.
4. **Skill Manifest** —unified manifest: typed I/O, prerequisites, guardrails, validation/eval thresholds, security checks, rollback, evidence_required, depends_on (versioned), cost_profile, emits, idempotent, provenance, promotion/deprecation (Appendix A).
5. **Skill Lifecycle (FSM)** —`Draft → Candidate → Validated → Certified` (+ `Deprecated/Blocked`); Certified requires ≥3 independent projects, 0 blocked, 0 critical scan findings, mutation ≥0.80, human approval; retrieval forbids Deprecated/Blocked.

## 5.6 Agent permission model `[design principle]`

Least-privilege per role, enforced as scoped tokens by Policy-as-Code:

[1] The governance-grade workflow platform adopted in the observed case —and in the prior RAG work of [Lu & Yang 2026] —was **n8n**, an open-source, self-hostable workflow-automation platform whose enterprise controls —role-based access control, SSO, execution-log streaming/audit, source control & environments, and external secrets —are documented in [n8n 2026]. We name it here only for reproducibility; the contribution is platform-independent, and we do not claim n8n is a required, endorsed, or uniquely suitable choice.

| Agent | Allowed | Forbidden | Gate |
|---|---|---|---|
| Requirement | read reqs, write acceptance/invariants | modify production code | Human review |
| SD | architecture/ADR, API/DB design | direct deploy | Reviewer Gate |
| PG | modify source, branch | merge main | PR Gate |
| V&V | run tests, produce report | **modify acceptance criteria / invariants** | Mechanical Gate |
| Security | scan, sanitize external input | ignore critical finding | Security Gate |
| Deployment | staging, rollback | unapproved production deploy | Human Gate |
| Skill Curator | candidate, merge | promote Draft→Certified directly | Skill Validation Gate |
| Orchestrator | assemble, route, dispatch | use Deprecated/Blocked skills; skip hard gates; **modify workflow routing/Policy** | Policy Gate |

---

# 6. Skill Capitalization

The strategic claim: **not one-off code, but turning each project into reusable, verifiable, governable, auto-assemblable capability.** Engineering implication: **decreasing marginal cost** once enough Certified skills accumulate.

**Honest caveat —marginal cost may be U-shaped, not monotone.** As the repository grows, **curation overhead** (dedup, dependency upkeep, contract-test regression, deprecation) rises too. Decreasing marginal cost holds **only if curation cost is sub-linear in repository size**, maintained by the promotion FSM (§5.5), the anti-degradation algorithm (§9.2.1), and skill-health monitoring (§9.5). Otherwise it degrades into **skill sprawl**. The curation/promotion gate is the single point on which this claim rests.

---

# 7. Anti-Self-Deception Verification Governance

## 7.1 Failure mode: shared-context false pass (earlier "test collusion")

> **Terminology calibration**: we earlier called this "test collusion"; we adopt the more neutral, descriptive **shared-context false pass** to avoid implying intentional coordination, and to align with **Code-A1's** *self-collusion* (which already reports the phenomenon; we claim no naming originality).

**Definition** `[research hypothesis]`: in an Agentic SDLC, the producer (PG agent) and verifier (V&V agent) may form a correlated false-pass failure because they share implementation context, spec translation, training bias, or data source. It is an **integration** of existing topics (test oracle, common-mode failure, test-suite overfitting, LLM judge bias, code/test self-collusion) at full-SDLC governance —**not a new discovery**. Its rate and conditions are to be measured (EXP-001); we present mechanism + a single illustrative case, not a known general magnitude.

## 7.2 Why multi-model cross-checking may be insufficient —variance, bias, correlation〔hypothesis〕

> **Scope**: this section *argues a hypothesis* — "if estimators share bias, adding models lowers variance but not bias." Its premises (LLM errors are highly correlated; conditional covariance → 1) **await direct measurement (EXP-001)** and must not be read as established fact.

(a) Bias-term form: with $\hat{f}_i = f^* + b + \varepsilon_i$, the $n$-model average → $f^* + b$; adding models removes only variance, not shared bias $b$.
(b) Conditional-covariance form: when V&V conditions on the implementation, $\mathrm{Corr}(\mathrm{err}(f_{PG}), \mathrm{err}(g_{VV}) \mid \text{Implementation}) \to 1$, i.e. joint failure.
(c) Why LLMs *might* be correlated estimators (inductive-bias view, hypothesis): shared pretraining corpora and RLHF reward homogenization *may* align inductive biases —but "**different model**" ≠ "**statistically independent**", and the actual correlation must be measured (EXP-001 group B vs A).

> **Hypothesis (not a proven proposition)**: if the premises hold, model panels reduce variance, not bias; hence usable to "flag disagreement," not to "decide pass/fail." EXP-001's A/B/C/D arms directly test this (if B ≪ A, the hypothesis is revised).

## 7.3 Mechanical gate —executable evidence less dependent on model opinion

> `[design principle]`: mechanical gates give **executable, reproducible** evidence as a hard release condition. But coverage/mutation/property tests are **strong evidence, not a full correctness oracle**; they ensure "tests bite the code," not "requirements are correct" (garbage-in still needs humans + §7.7 invariants).

Mitigations not "add another AI" but model-independent facts: mutation testing; coverage thresholds; property-based testing; **tests generated from `acceptance_criteria.md`, not from the implementation**; **spec invariants** (§7.7); real execution & real scanners. Example gate (in Policy Repo):

```
vv_gate:
  type: mechanical
  rules: [coverage.line>=0.85, coverage.branch>=0.80, mutation.score>=0.75,
          tests.derived_from==acceptance_criteria.md, spec_invariants.all_hold==true, execution.exit_coo
  on_fail: route_to_PG_with_context
```

## 7.4 Three-layer verification

- **Layer 1** —**Mechanical gate (hard)**: every gate; pass/fail by §7.3/§7.7 metrics.
- **Layer 2** —**Model panel (soft)**: only where no mechanical ground truth exists (architecture review, threat modeling); blind, independent; explicit disagreement/escalation policy.
- **Layer 3** —**Human review of evidence**: humans review **mechanical evidence** (mutation report, scan diff, fail→pass test diff), **not** "**three AIs agreed**" (avoiding automation bias).

## 7.5 Core rules

```
Mechanical gates decide machine-checkable conditions; models only flag disagreement; humans retain autho
Tests derive from the spec, never from the implementation.
Spec invariants are mechanical red lines; they hold or the gate fails.
No global budget, no autonomous loop.
Humans review evidence, not consensus.
```

## 7.6 Self-correction safety valve

A global session budget (token/time/cost circuit breaker); skipping verified nodes only for *generation* stages (SA/SD), never verification; returning to V&V reruns the full suite; structural changes mark `architecture.md`

dirty and trigger design-consistency checks.

### 7.7 Spec invariants —upstream defense against garbage-in `[design principle]`

The Requirement agent must also produce `spec_invariants.md` —system-level **red-line invariants** and **metamorphic relations** (e.g., balance ≥ 0 under any operation; stricter filter never increases count). They describe "what the system must never violate," catching "requirement correct but implementation crosses the line" and "requirement omits a red line." The example below is a **post-fix regression safeguard** —the pre-fix inflated snapshot was not preserved, so it does **not** reproduce the original interception; it guards against recurrence. It uses **two separate properties**, because the historical defect was a *pagination duplicate-row* bug, which only a de-duplication metamorphic relation can catch (a single hard-coded upper bound cannot):

```python
from hypothesis import given, strategies as st

# (1) De-duplication metamorphic relation: re-appending existing rows must NOT
#     change the unique-key aggregate. The buggy no-dedup sum FAILS this; the
#     dedup-by-id aggregate PASSES. This is the relation that actually catches
#     pagination duplicate rows. (Verified on synthetic data: naive sum 180→330
#     under duplication = FAIL; dedup 180→180 = PASS.)
def aggregate_unique(records):
    unique = {r["id"]: r for r in records}
    return sum(r["count"] for r in unique.values())

records_strategy = st.lists(
    st.fixed_dictionaries({"id": st.integers(1, 50), "count": st.integers(1, 5000)}),
    min_size=1, max_size=100)

@given(records=records_strategy)
def test_duplicate_invariance(records):
    duplicated = records + records[: max(1, len(records) // 2)]
    assert aggregate_unique(duplicated) == aggregate_unique(records)

# (2) External upper-bound invariant: a channel's monthly count must not exceed
#     the national monthly total, where the bound comes from an INDEPENDENT
#     dataset — NOT a hard-coded constant (the real vv_crosscheck.py computes it).
def test_national_upper_bound(channel_count, national_total_from_independent_source):
    assert channel_count <= national_total_from_independent_source
```

> **Why the earlier draft's single test was invalid** (corrected in v3.2): with `count ≤ 5000` over `≤ 100` rows the sum can never reach a hard-coded `1_000_000`, so the bound never triggered, and a no-dedup sum was never challenged by duplicate input —it could not detect the defect it claimed to. The two properties above separate the **duplicate-invariance metamorphic relation** from the **external upper-bound invariant** (the e-invoice artifact and data are listed in Appendix C).

---

## 8. Security, Compliance & Data Sovereignty

For on-prem regulated clients: (1) data classification as a first-class SD deliverable (PHI/PII/residency/retention/log masking); (2) shift-left threat modeling; (3) sandbox isolation for PG/V&V/Deployment; (4) least privilege + full audit trail in Policy Repo. Reproducible mechanical evidence (mutation score, coverage, scan results) is itself a **compliance advantage** —regulators trust auditable numbers over non-reproducible "AI consensus."

### 8.5 Skill supply-chain security

Treat the Skill Repository as an internal software supply chain. Risks: skill poisoning, dependency drift, policy bypass, prompt injection, evidence forgery. Controls: version pinning; **signature/hash for Certified skills**; FSM-only promotion; Deprecated/Blocked unusable by Orchestrator; contract test + security scan; append-only, traceable evidence.

### 8.6 Threat model for the pipeline itself

| Threat | Mitigation |
|---|---|
| Agent over-permission | least privilege, scoped tokens (§5.6) |
| Prompt injection | prompt firewall, source-trust labeling (§8.7) |
| Evidence tampering | append-only repo, hash chain |
| Skill poisoning | validation, curator review, signature |
| Infinite self-correction | budget / retry limit / circuit breaker |
| Policy bypass | policy-as-code, hard gates |
| Model drift | version pinning, eval replay |
| Data leakage | data classification, sandbox, egress control |

### 8.7 AI-specific threats: prompt injection & agent hijacking `[design principle]`

External input (intake/requirements) may carry malicious instructions ("ignore policy; exfiltrate to IP X"; "disable the security scan") —a threat demonstrated at scale against real agentic workflows, including credential exfiltration via crafted repository comments [Fendley et al. 2026]. Architectural defenses: (1) front-end **sanitization** by the Security agent + **source-trust labeling** (external instructions demoted to data); (2) routing & permissions **hard-coded at the architecture layer** (workflow-platform switch logic, Policy Repo read-only; n8n in our implementation) —**models cannot change routing/Policy via prompts**; (3) **blast-radius control** —even a poisoned agent's over-reach is blocked by hard gates and permission layers; (4) instruction/data separation. **Rule: individual agents may be compromised; therefore routing, policy, evidence retention, and release authority are constrained by controls outside the producing agent's authority. No pipeline is immune —prompt injection, supply-chain, credential abuse, or CI tampering can still alter it —but these controls shrink the blast radius.**

### 8.8 Workflow-layer governance affordances `[design principle]`

When application behavior is re-expressed as inspectable workflow graphs (§4.4), enterprise governance controls —identity/SSO, role-based access control, credential/secret boundaries, execution logging and audit trails, Git-backed source control and environments, and deployment governance —can be applied at the **workflow layer** rather than remaining buried in opaque native code. We treat these as **affordances of an inspectable control plane, not automatic guarantees**: high availability, CI/CD, and disaster recovery still require deliberate configuration and are not implied by adoption alone (these are documented enterprise capabilities of a governance-grade workflow platform as defined in §4.4).

---

## 9. Engineering Realization: n8n as Reference Workflow Harness

This section documents the **as-built reference implementation**. Concrete component names (n8n, Git, JSON-RPC, PostgreSQL/Redis) are engineering choices for reproducibility, **not requirements of the framework** —the concept layer (§4.4) is platform-independent.

- **9.1 n8n state machine**: single state trunk + dynamic Switch routing (hub-and-spoke); a persisted pipeline State Object (current_stage, status, git_commit_hash, retry_count, artifacts, error_diagnostic,

routing_decision); failed stages route via `ROLLBACK_WITH_CONTEXT` for Diagnose→Patch without rebuilding. Routing logic lives in Code Nodes, not editable by production agents via prompt.

- **9.2 JSON-RPC** between Orchestrator and Skill Curator (`submit_draft_skill` → `ACCEPTED/REJECTED`); additive vs breaking merges (minor vs major bump + contract tests + human gate for breaking). The choice of a JSON-RPC harness protocol mirrors industry practice (cf. OpenAI's Codex App Server, a bidirectional JSON-RPC harness [OpenAI 2026b]).
- **9.2.1 Curator anti-degradation algorithm**: semantic-overlap detection via vector + **artifact-specific similarity** (AST for code; embeddings for prompts; node/edge graph for n8n workflows; rule-set diff for policies). If combined similarity $S > \tau$ (initial policy 0.85, **organization-tuned, not a validated threshold**), force merge; on merge run **contract regression tests** against prior consumers; deprecate superseded skills.
- **9.3 State/evidence separation**: high-frequency mutable state in PostgreSQL/Redis (not in Git); only boundary snapshots + hashes to the append-only Git Evidence Repo; large artifacts in a content-addressed object store (Git stores only hashes).
- **9.5 Meta-loop (Kaizen)**: monitor the *pipeline itself* (per-agent latency/cost/retries/refusal/gate-failure/self-correction; per-skill assembly count/prod-failure/demotions) → pipeline-health dashboard → Retrospective.
- **9.6 Cost engineering of the mechanical gate**: mutation testing is a cost hotspot (10–100× the suite). Use diff-scoped mutation, sampled operators, fail-fast ordering, tiered gates (PR-level incremental vs release-level fuller), caching + parallel containers. The gate's pass/fail authority is non-negotiable, but its execution strategy is a tunable Policy parameter.

## 9.7 Claim–Evidence Matrix

**No claim is `[validated]` as of v3.4.**

| ID | Claim | Evidence | Level | Next |
|---|---|---|---|---|
| C1 | shared-context false pass exists | e-invoice **retrospective field observation** (same-source false pass; pre-fix snapshot not preserved) | `[preliminary]` | EXP-001 AI-to-AI data |
| C2 | mechanical gate > model review at blocking defects | hypothesis; no controlled comparison yet | `[hypothesis]` | controlled ablation |
| C3 | capitalization lowers cross-project cost | none | `[hypothesis]` | 2nd case + tracking |
| C4 | Evidence/Policy repos improve auditability | architectural | `[design principle]` | auditor study |
| C5 | LLMs highly correlated; swapping models doesn't reduce bias | side evidence (judge bias) + statistical argument | `[hypothesis]` | EXP-001 error-correlation |

| ID | Claim | Evidence | Level | Next |
|---|---|---|---|---|
| C6 | spec invariants catch garbage-in defects | post-fix invariant passes at 17.3% of an independent upper bound (present safeguard demonstrated; historical interception **unverified** — pre-fix snapshot not preserved) | `[preliminary]` | EXP-001 D arm |
| C7 | No-evidence/No-rollback iron rules | engineering practice | `[design principle]` | — |
| C8 | State Object / RBAC / append-only / JSON-RPC | implemented in reference stack | `[engineering spec]` | — |
| C9 | workflow-native harnesses improve governability of non-code-readable AI-generated applications | preliminary field observation, generalized from a prior RAG system case [Lu & Yang 2026]: vibe-coded native code opaque to operators; re-expressing behavior as governed workflows, with a reusable Skill applying mechanical/security gates (efficiency, security, permission-boundary, data-exfiltration checks), enabled RBAC / secrets / execution logging / Git source-control-environments / deployment governance at the workflow layer | `[preliminary]` | comparative case study (native-code vs workflow-native governability) |

## 10. Evaluation Plan

| RQ | Claim | Method | Metrics |
|---|---|---|---|
| RQ1 | shared-context false pass exists | test-from-implementation vs test-from-spec on seeded-defect tasks | mutation score, escaped defects, false-pass rate |
| RQ2 | mechanical gate efficacy | with/without mutation+property+invariants (ablation) | defect detection, coverage, regression failure |
| RQ3 | capitalization lowers cost | with/without Skill Repository across projects | lead time, rework, reuse rate |
| RQ4 | governance auditability & workflow-native governability | with/without Evidence+Policy repos; native-code vs workflow-native re-expression | evidence completeness, approval traceability, rollback readiness, auditor-traceability of a transaction path |

**Minimal experiment EXP-001** is frozen as a protocol: 20–50 containerized seeded-defect tasks across five arms (A same-model/same-context, B cross-model/same-context, C spec-isolated, D mechanical gate, E human golden), reporting false-pass/defect-detection/escaped/mutation/coverage/cost. Planned statistics: **McNemar / paired-proportion tests** for false-pass rates (paired by task); **Wilcoxon signed-rank / paired-t** for cost and time; each with **effect sizes and confidence intervals**, and sample size set by a **power analysis** before the run. Results (supporting or refuting) will be written back to §7/§9.7/§10/§12.2. **Status: Protocol frozen; runner incomplete.**

**Study 2 —workflow-native governability (RQ4 / C9).** Companion comparative case study: 6–10 vibe-coded applications/tasks reviewed **native-code vs workflow-native**, measuring **auditor-traceability time, permission-boundary defects found, security findings, rollback readiness, and approval completeness**; anonymized regulated cases follow the case-study evidence protocol (multiple-case design, ≥2 independent sources per claim). **Status: design drafted; not yet run.** Until Study 2 completes, **C9 remains `[preliminary field observation]`, not `[validated]`.**

---

# 11. Maturity Model

L0 AI Coding → L1 Agent Task Automation → L2 Artifact Discipline → L3 Governed Agentic SDLC → L4 Skill Capitalization → L5 AI-Native Software Factory. Each level has entry criteria / required artifacts / mandatory gates / evidence retention / human authority / measurable KPIs.

> KPI numeric thresholds are **organization-defined**, not framework-validated universals; the figures in the Chinese edition are **examples** to be calibrated per organization and with EXP-001/cross-project data. Do not read them as validated maturity thresholds. Pragmatic advice: reach **L3 first** (governance + verifiability), then L4/L5.

---

# 12. Discussion

## 12.1 Difference from generic AI coding

`Prompt → Code → Debug` vs `Requirement(+Spec Invariants) → Analysis → Architecture → Coding → Verification(mechanical) → Security → Deployment → Monitoring → Skill Extraction`.

### 12.2 Limitations & threats to validity

1. **No large-scale empirical evaluation** yet (§10).
2. **Mechanical-gate coverage boundary (garbage-in)**: coverage/mutation ensure "tests bite code," not "requirement correct." §7.7 invariants reduce but do not eliminate this; invariant completeness still needs humans.
3. **spec→test translation residual**: the test-writer is still an LLM and may mistranslate the spec. The trust root is "human-accepted acceptance criteria + invariants + translation fidelity"; the last is an open gap.
4. **Cost & latency**: multi-agent + self-correction + model panel + mutation amplify cost; §7.6 budget and §9.6 cost engineering are necessary but not sufficient.
5. **Composability assumption**: typed I/O + dependency declarations may not guarantee semantic compatibility; contract tests needed (§9.2.1).
6. **Repository-scale degradation**: marginal cost may be U-shaped; sub-linear curation is an assumption.
7. **Human bottleneck**: over-reliance on human gates offsets autonomy.
8. **Reproducibility**: the model-panel layer is hard to reproduce (version drift, sampling), hence excluded from pass/fail decisions.

### 12.3 Ethics & responsibility

Under autonomous deployment and high privilege, accountability, audit trail, and final human approval are mandatory. The framework reserves human approval for high-risk actions (production deploy, schema migration, security exceptions, budget overrun), enforced as architectural constraints (§5.6, §8.7).

---

## 13. Future Work

1. Empirical studies (§10): quantify shared-context false pass rate and mechanical-gate interception across real projects.
2. Requirement-level formal verification: SAT/SMT consistency/satisfiability on `acceptance_criteria.md` and `spec_invariants.md`; machine-checkable spec→test traceability.
3. Skill semantic compatibility beyond type level.
4. Pipeline self-optimization from §9.5 meta-loop data.
5. Cross-organization skill federation under data-sovereignty constraints.

### 13.6 Pre-publication checklist

This is a calibrated Working Draft; before formal journal submission: run EXP-001 (pilot), fully verify every citation, complete §2.7 search-reproducibility, align case-study wording, keep the originality statement non-absolute, update §9.7/§1.5 as results arrive, and consider splitting into White Paper / Technical Specification / Evaluation Protocol.

---

## 14. Conclusion

This work reframes AI code generation from "one-off output" into a **governable, verifiable, accumulative software-production lifecycle**. Its three candidate pillars —closed full-lifecycle loop, skill capitalization, and anti-self-deception three-layer verification —address a question under-integrated by prior "AI coding" work: **when the verifier is also an AI, how to reduce the risk of a "all-green but crashing" pipeline.** Core stance (a design orientation, pending empirical confirmation): **use reproducible mechanical evidence to decide machine-checkable release conditions; model panels only flag disagreement; humans review evidence rather than consensus and retain authority over ambiguous and high-risk decisions; individual agents**

**may be compromised, so control over routing, policy, and release sits outside the producing agent (reducing —not eliminating —blast radius).** Claim strengths are tagged in §9.7 and await EXP-001/002.

---

## Generative AI Use Disclosure

Generative AI systems were used to assist with drafting, restructuring, literature discovery, statistical formalization, experiment-protocol design, and **independent adversarial critique**. Roles: Claude (operationalization, integration, verification, version control), Codex (independent adversarial review, prior-art and consistency checking —which identified the renaming, the Code-A1 prior art, and a citation misreading corrected in v3.0), and Gemini (formalization, code examples, copy-editing). The named human author (Yeh-Hsing Lu) selected the research questions, supplied the field evidence, reviewed and revised all generated material, **verified the cited references against source documents** (including the non-arXiv MCP and Data Science Dojo references, checked 2026-06-22; see §2.7 and §13.6), and accepts full responsibility for the manuscript. AI systems are **not** authors. This aligns with IEEE/ACM policies on AI-generated content.

---

## Appendix C: Artifact & Data Availability

- **Live demo**: E-Invoice Open-Data Dashboard —https://tigerai-ai-dashboard-demo.yesinlu.workers.dev [D1]
- **Data source**: Ministry of Finance, Taiwan —E-Invoice Open Data (government open-data license) [D2]
- **Verification gate implementation**: `vv_crosscheck.py` (system-level upper-bound invariant; §7.7); post-fix run PASS, ratio 17.3% of the independent upper bound —a **present safeguard demonstration**, not evidence that the gate intercepted the original 4.5× defect (pre-fix snapshot not preserved; see case study).
- **Confidentiality**: the proprietary delivery *skill set* distilled from the worked example is a trade secret; the paper provides only an abstract description and external outcome metrics, not the skill list, data contract, or internals. This does not affect the public demo or open data above.

---

## References


**Foundational** 1. DeMillo, R. A., Lipton, R. J., & Sayward, F. G. (1978). *Hints on Test Data Selection.* IEEE Computer 11(4). 2. Claessen, K., & Hughes, J. (2000). *QuickCheck.* ICFP. 3. Chen, T. Y., Cheung, S. C., & Yiu, S. M. (1998). *Metamorphic Testing.* HKUST-CS98-01. 4. Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., & Yoo, S. (2015). *The Oracle Problem in Software Testing: A Survey.* IEEE TSE 41(5). 5. Kim, G., et al. (2016). *The DevOps Handbook.* IT Revolution. 6. Forsgren, N., Humble, J., & Kim, G. (2018). *Accelerate.* IT Revolution. 7. Shostack, A. (2014). *Threat Modeling.* Wiley. 8. Dietterich, T. G. (2000). *Ensemble Methods in Machine Learning.* LNCS 1857. 9. Yao, S., et al. (2023). *ReAct.* ICLR. arXiv:2210.03629. 10. Wang, X., et al. (2022). *Self-Consistency.* arXiv:2203.11171. 11. Du, Y., et al. (2023). *Multiagent Debate.* arXiv:2305.14325.

**Supply-chain provenance** 12. SLSA (2023). *Supply-chain Levels for Software Artifacts v1.0.* OpenSSF. 13. Torres-Arias, S., et al. (2019). *in-toto.* USENIX Security.

**LLM judge bias / Agentic SWE / collusion (arXiv-verified)** 14. Anthropic (2024). *Model Context Protocol (MCP).* Open standard for connecting AI applications to data, tools, and workflows. https://modelcontextprotocol.io (verified 2026-06-22). 15. Jimenez, C. E., et al. (2023). *SWE-bench.* ICLR 2024. arXiv:2310.06770. 16. Hassan, A. E., Li, H., Lin, D., Adams, B., Chen, T.-H., Kashiwa, Y., & Qiu, D. (2025). *Agentic Software Engineering: Foundational Pillars and a Research Roadmap.* arXiv:2509.06216. Discusses agentic SE (SE 3.0), ACE/AEE environments (related work). 17. Bashir, A., Han, T. A., & Shamszaman, Z. U. (2025). *Many-to-One Adversarial Consensus:*

*Exposing Multi-Agent Collusion Risks in AI-Based Healthcare.* arXiv:2512.03097. **Verified reading**: unprotected systems reach attack/harmful-recommendation rates **up to 100%**; a verifier checking decisions against external (clinical) guidelines **restored accuracy by blocking adversarial consensus** (verifier effective —not a claim that consensus always fails or verifiers always succeed). 18. Keppo, J., Li, Y., Tsoukalas, G., & Yuan, N. (2026). *On the Fragility of AI Agent Collusion.* arXiv:2603.20281. (collusion in pricing scenarios; weakly related) 19. Tailor, O. (2025). *Audit the Whisper: Detecting Steganographic Collusion in Multi-Agent LLMs.* arXiv:2510.04303. 20. Bisconti, P., Galisai, M., Pierucci, F., Bracale, M., & Prandi, M. (2025). *Beyond Single-Agent Safety: A Taxonomy of Risks in LLM-to-LLM Interactions.* arXiv:2512.02682. 21. Zheng, L., et al. (2023). *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena.* NeurIPS 2023. arXiv:2306.05685. 22. Data Science Dojo (2026). *Agentic Loops: From ReAct to Loop Engineering (2026 Guide).* https://datasciencedojo.com/blog/agentic-loops-explained-from-react-to-loop-engineering-2026-guide/ (verified 2026-06-22). 23. Wang, A., Yan, Y., Zhou, N., et al. (2026). *Code-A1: Adversarial Evolving of Code LLM and Test LLM via RL.* arXiv:2603.15611. **Direct prior art —code/test self-collusion.**

**Author's prior work (research line)** 24. Lu, Y.-H. (2026). *AI-Native eBook Production: Multi-IDE Orchestration as Methodology Engineering Infrastructure —A Design Science Investigation.* Zenodo. DOI 10.5281/zenodo.20264772. 25. LangChain (2026). *The Art of Loop Engineering.* https://www.langchain.com/blog/the-art-of-loop-engineering (verified 2026-06-27). Frames loop engineering as stacking agent / verification / event-driven / hill-climbing loops. 26. Böckeler, B. (2026). *Harness engineering for coding agent users.* MartinFowler.com. https://martinfowler.com/articles/harness-engineering.html (verified 2026-07-17). Frames coding-agent harnesses as feedforward guides plus feedback sensors, distinguishing computational and inferential controls. 27. Lu, Y.-H., & Yang, W.-H. (2026). *Rule-Driven Adaptive Chunking with Hierarchical Keyword Taxonomy for Domain-Specific Retrieval-Augmented Generation* (v0.2.6, preprint). Zenodo. DOI 10.5281/zenodo.20579355. **Author's prior system case** —visual workflow graphs more inspectable/auditable to enterprise reviewers than native code; the workflow-native governance observation this paper generalizes (§4.4). 28. Cloud Security Alliance (2026). *The Vibe Coding Governance Gap* (AI Safety Initiative research note, June 2, 2026). https://labs.cloudsecurityalliance.org/research/csa-research-note-vibe-coding-ai-governance-gap-20260602-csa/ (verified 2026-07-18). 29. Okta (2026). *AI Agent Orchestration: Identity Control Plane.* https://www.okta.com/identity-101/ai-agent-orchestration/ (verified 2026-07-18). Workflow orchestration as an identity/RBAC/audit control plane. 30. Dataiku (2026). *Enterprise vibe coding has an AI governance problem.* https://www.dataiku.com/stories/blog/enterprise-vibe-coding-problem (verified 2026-07-18). 31. n8n (2026). *Enterprise documentation: role-based access control, SSO, log streaming, source control & environments, external secrets.* https://docs.n8n.io (verified 2026-07-18). Cited for the reference implementation's enterprise controls (§4.4 note, §9). *(Formal-submission TODO: split into per-feature docs URLs —RBAC, SAML/OIDC SSO, log streaming, source control/environments, external secrets.)* 32. OpenAI (2026). *Harness engineering: leveraging Codex in an agent-first world.* https://openai.com/index/harness-engineering/ (verified 2026-07-18). Primary industry field report on coding-agent harnesses (an agent-first codebase built largely by Codex agents); grounds *harness engineering* as an established 2026 practice. 33. OpenAI (2026). *Unlocking the Codex harness: how we built the App Server.* https://openai.com/index/unlocking-the-codex-harness/ (verified 2026-07-18). Describes a bidirectional **JSON-RPC** App Server powering the Codex harness —supports our JSON-RPC inter-agent design (§9.2). 34. Fendley, N., Liu, Z., Guan, A., Zhong, J., & Cao, Y. (2026). *Comment and Control: Hijacking Agentic Workflows via Context-Grounded Evolution.* arXiv:2605.11229. Real prior art on agentic-workflow hijacking / prompt injection (≈5,000 GitHub workflows; credential exfiltration) —motivates §8.7.

**Related & concurrent work (2025–2026) —see §1.6** 35. Advani, L. (2026). *From Confident Closing to Silent Failure: Characterizing False Success in LLM Agents.* arXiv:2606.09863. Prior characterization of false success / judge anchoring on confident closing language. 36. Fan, W., Nie, X., & Dai, Z. (2026). *Harness-MU: A Safe, Governed, and Effective Harness for Multi-User LLM Agents.* arXiv:2606.21856. Governance as deterministic runtime hooks (multi-user runtime, not the SDLC). 37. Miculicich, L., Parmar, M., Palangi, H., Dvijotham, K. Dj, Montanari, M., Pfister, T., & Le, L. T. (2025). *VeriGuard: Enhancing LLM Agent Safety via Verified Code Generation.* arXiv:2510.05156. Formally verified, action-level policy guard (offline verification + online monitoring). 38. Ma, Z., Zhang, T., Cao, M., Liu, J., Zhang, W., Luo, M., Zhang, S., & Chen, K. (2025). *Rethinking Verification for LLM Code Generation: From Generation to Testing (SAGA).* arXiv:2507.06920. LLM-generated tests homogenize toward the producing model's blind spots. 39. Madatha, P. (2026). *A Deterministic Control Plane for LLM Coding Agents.*

arXiv:2606.26924. Permission tiering + phase state machine + hash-chained audit at the agent-configuration layer. 40. Qin, X., Luan, S., See, J., Boukhers, Z., Yang, C., & Li, Z. (2026). *Governed Capability Evolution: Lifecycle-Time Compatibility Checking and Rollback for AI-Component-Based Systems.* arXiv:2604.08059. Seven-stage governed lifecycle with compatibility checks, regression detection, rollback. 41. Liu, H., Yang, H., Jiang, T., Tang, B., Xiong, F., Luo, Y., & Li, Z. (2026). *SkillsVote: Lifecycle Governance of Agent Skills from Collection, Recommendation to Evolution.* arXiv:2605.18401. Evidence-gated updates + anti-degradation (prevents indiscriminate updates polluting context).

**Artifacts (Appendix C)** - [D1] TigerAI (2026). *E-Invoice Open-Data Dashboard —Live Demo.* https://tigerai-ai-dashboard-demo.yesinlu.workers.dev (accessed 2026-06-21) - [D2] Ministry of Finance, Taiwan —*E-Invoice Open Data.* Government open-data license.

---

*—End of Working Paper (v3.4, English edition; evidence-governed harness engineering for agentic software development) —*