Agent Harnesses Need a Pattern Discipline — Scaling Alone Cannot Absorb Their System Responsibilities

Haoyi Xiong

Abstract

An agent's behavior arises not from a model alone, but from a model embedded in an executable harness that assembles context, exposes tools, enforces permissions, maintains state, evaluates work, and decides when to stop. As models improve, they may absorb some of this scaffolding. Other harness responsibilities have a different status: authorization, acceptance, audit evidence, recovery, and termination are obligations to people and systems outside the model, and their violation remains consequential regardless of model scale. Yet these responsibilities are usually recorded only as component names, implementation details, or prompt recipes, making them difficult to preserve as harnesses evolve. We argue that agent harnesses need *pattern discipline*: revisable records that connect recurring mechanisms to protected contracts, permitted variation, authority, evidence, and lifecycle conditions. The A^2 (Activities × Assets) taxonomy follows these records through design, verification, and evolution. From this framework we derive three major hypotheses: inspectable commitments, rather than particular prompts or code, are the durable asset (H1); agent-led evolution requires mechanized separation of concerns (H2); and explicit variation boundaries improve self-modification relative to unconstrained revision (H3). Nine analytical cases and three structural drifts develop the conceptual argument. An exploratory audit of 52 literature-sourced records in 41 repositories supplies real-code examples and shows why retrieval cues, structural matches, pointcuts, and contracts must remain distinct. We specify prospective, falsifiable protocols for all three hypotheses, including a two-arm study of bounded self-modification; these protocols are designs, not completed causal tests. The contribution is therefore a research framework and instrument for studying how harness responsibilities can survive change, rather than a claim that the proposed discipline has already demonstrated causal benefit.

2012 ACM Subject Classification Computing methodologies → Artificial intelligence

Keywords and phrases agent harnesses, software design patterns, system responsibilities, software evolution, agent evaluation

1 Introduction

An agent's behavior is produced by a model embedded in a *harness*: the executable runtime that assembles context, exposes tools, enforces permissions, maintains state, invokes evaluators, and decides when to stop. ReAct made the reasoning–action loop explicit [55]; SWE-agent showed that the agent–computer interface changes outcomes [54]. A benchmark result is therefore a property of model × harness under a task, evaluator, and budget—not of the model in isolation [26, 1]. Building and maintaining this runtime is now an engineering activity in its own right, with official guides, reference implementations, and surveys [2, 35, 28, 19, 31].

As harnesses grow, their builders reach for object-oriented design (OOD) to keep them changeable. OO-Harness specializes related harnesses through inheritance and routes tasks among subclasses polymorphically [25]. HarnessX constructs and evolves reusable harness components [13], and natural-language harness documents turn run-level policy, handoffs, and validation gates into explicit artifacts [36]. Each design exposes a seam: a place where behavior may change while an interface is meant to hold.

These seams are increasingly edited by agents. Harness^2 improves a harness with contrastive probes and carries adaptation experience across iterations [53]; RRSI regularizes repeated harness improvement [51]; Harmony Harness searches among competing functionality and security objectives [57]; and observability-driven evolution uses traces to guide candidate changes [26]. Harness engineering is thereby becoming closed-loop: the system that is evaluated also proposes its successor. We use *recursive self-improvement* (RSI) for this system-level loop, which is distinct from training model weights.

Empirical studies of harnesses have taken three forms.

- **Source studies** read implementations: an audit of eleven coding agents reports recurring mechanisms [6].
- **Literature studies** mine publications: Cai et al. extract quality attributes and 16 design patterns from 94 papers on LLM-based multi-agent systems for software engineering [11].
- **Outcome studies** vary the runtime and measure task results, cost, or ablated modules [1, 26, 36].

Together they show that mechanisms recur and that harness choices matter. What most of them record, however, is a mechanism name or an end-to-end score.

The missing object is the responsibility that must survive change. Harnesses are repeatedly modified as models, tools, interfaces, and tasks change, and a rewrite can alter what the model observes, which actions require approval, what counts as acceptance, or which traces remain auditable. Existing descriptions often fail at precisely this transition. A component name such as "memory," "router," or "evaluator" identifies a mechanism without stating the behavior it protects or who may revise it. Nor is recurrence itself validation: a lexical cue, a source instance, a matched pointcut, a tested contract, and a measured outcome answer different questions. The difficulty intensifies when the maintainer is an agent capable of rewriting its own checks, permissions, or provenance. Even apart from self-modification, utility remains model-conditional: a mechanism that helps one model snapshot may be unnecessary for the next, while the obligation it enforced remains.

Software engineering met a similar problem when systems outgrew their first designs, and it answered with design patterns. We read that literature as two streams. The *assets* stream says what a design keeps stable and where it may vary: contracts, frozen points, hot spots, and variation mechanisms [16, 8, 37]. The *activities* stream says how those assets are used over a lifecycle: in design and implementation, in testing and verification, and in maintenance and evolution [15, 24]. Our intuition is that harness engineering should inherit both streams and extend them for model–runtime interaction, stochastic behavior, model-conditional utility, and components that can change their own evaluator, permission boundary, or evidence trail. We cross the two streams into the A^2 (Activities $\times$ Assets) taxonomy (Figure 1). The assets axis states what a harness mechanism protects, its *protected contract*, and where it may change, its *permitted variation*. The activities axis states when that obligation is specified (A1, design and implementation), tested (A2, testing and verification), and maintained or reopened (A3, maintenance and evolution). Ordinary customization stays inside the declared variation seam; changing the protected contract itself returns from A3 to A1 and requires explicit authority. Every candidate pattern in this paper is placed on these coordinates. Table 1 crosses the two streams and shows, cell by cell, how each software concern extends to a harness responsibility.

These considerations lead to the paper's central position: agent harnesses need **explicit, revisable responsibility records**. Each record states a protected contract, permitted variation, authority, acceptance evidence, failure consequences, and lifecycle conditions. We call the practice of recording and testing these obligations **pattern discipline**. Its normative

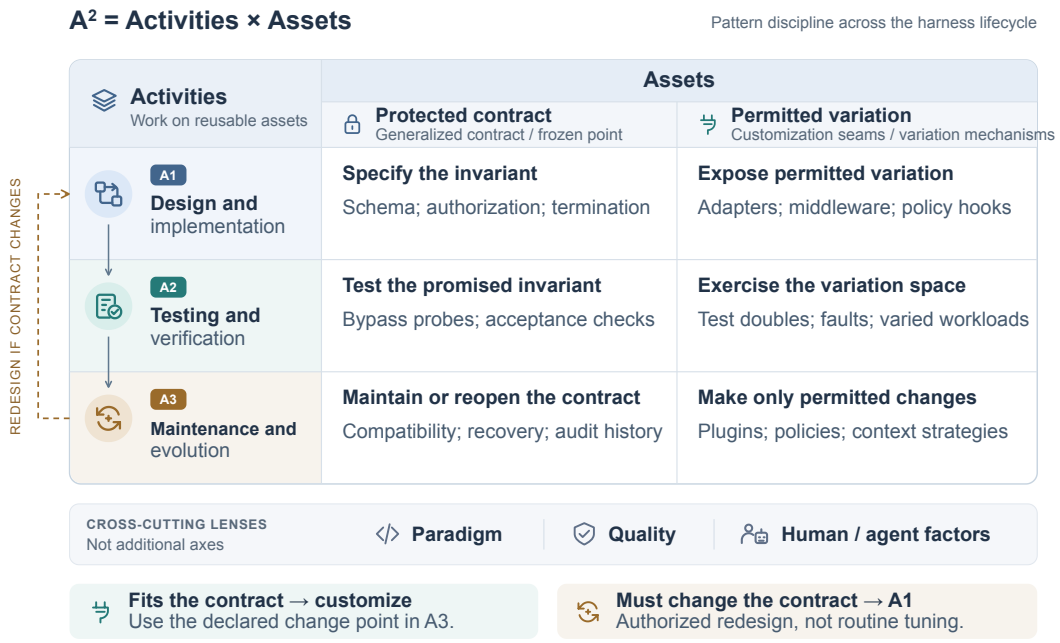


Figure 1 The proposed A^2 (Activities × Assets) taxonomy. The Activities axis comprises design and implementation (A1), testing and verification (A2), and maintenance and evolution (A3). The merged Assets header groups generalized contracts and customization seams: frozen points state scoped invariants, while hot spots and variation mechanisms expose permitted changes. Cell contents illustrate our proposed harness interpretation and are not an exhaustive catalog. Solid arrows indicate the lifecycle relationship, which is not a mandatory sequential workflow. The dashed A3-to-A1 return applies when an asset's protected contract must change; ordinary customization stays within the documented variation seam. Redesign requires explicit authority, and retirement does not remove external obligations. Paradigm, quality, and human/agent factors are cross-cutting lenses and not additional axes.

minimum is that consequential responsibilities remain identifiable, authorized, and auditable through change. Whether the particular record schema proposed here improves that practice remains an empirical question.

Two clarifications fix the scope of the position. First, the *system responsibilities* of the title are obligations whose violation has consequences outside the model: permissions, external side effects, acceptance authority, audit evidence, recovery, and termination. A stronger model can absorb cognitive scaffolding such as decomposition or planning aids, and we expect it to. It cannot, by improving, assign authority or preserve an external audit trail. The position concerns the second kind. Second, a record is not enforcement. A record makes a responsibility identifiable; keeping it authorized and auditable also takes runtime mechanisms. We therefore treat record utility and runtime enforcement as separate empirical questions, and their combination in closed-loop self-modification as a third.

The paper makes three connected contributions.

- **A responsibility-preservation framework.** Pattern discipline attaches authority, acceptance evidence, violation consequences, and lifecycle conditions to a mechanism, and the A^2 taxonomy follows those obligations through design, verification, and evolution.
- **An evidence instrument** that keeps retrieval cues, source locations, structural matches, pointcuts, contracts, lifecycle trajectories, and causal outcomes distinct.

Table 1 Software-to-harness transitions under the proposed A^2 lens. Read down an activity or across from its protected contract to its permitted variation. Each cell pairs a software-engineering focus with our proposed harness-engineering extension as an analytical mapping. Coding-agent mechanisms attributed to Barbaste et al. [6] carry source locators in the online inventory, and named evolution methods are research mechanisms. The online inventory supplies the detailed harness records and provenance; Appendix A.2 reports coverage.

Activities	Assets	
	Protected contract	Permitted variation
A1 *Design and implementation*	*Software engineering:* keeps interfaces and control rules stable across software components [16, 8, 41]. *Harness engineering:* keeps rules for what models and tools can see, do, return, and control stable while they work together [6, 55, 39, 2].	*Software engineering:* changes behavior through code hooks and replaceable components [16, 41]. *Harness engineering:* extends these change points to model providers, processing steps, tools loaded when needed, work routing, parallel tasks, and control handoffs [6, 3, 35, 39].
A2 *Testing and verification*	*Software engineering:* exposes dependencies so tests can control inputs and check outputs [15, 16, 41]. *Harness engineering:* also protects stopping rules, minimum permissions, isolated execution, and time or work limits from agent bypass [6].	*Software engineering:* swaps settings or components to test different cases [15]. *Harness engineering:* makes task rules, reviewers, memory, and improvement goals replaceable while keeping required safeguards fixed [6, 2, 42, 51, 57].
A3 *Maintenance and evolution*	*Software engineering:* stores component state and records changes so a system can be inspected or restored [16, 41, 14]. *Harness engineering:* also preserves event history, alternative session paths, the origin of changes, and evidence needed after context or model updates [6].	*Software engineering:* updates or replaces configurable parts as requirements change [41, 16, 22]. *Harness engineering:* also updates memory, task routing, policies, and the harness itself, using observed failures while preserving permissions, checks, records, and recovery duties [6, 25, 53, 13, 12, 26].

- **Prospective hypotheses and protocols.** H1–H3 convert the position into claims that can be supported, rejected, or left unresolved.

These contributions build on, rather than replace, existing harness catalogs and executable harness documents [6, 11, 36].

An exploratory repository audit supports the argument. From the literature we collect 52 candidate records in five source families and search for their vocabulary in 41 frozen repositories from 36 repository families. The vocabulary is readily retrieved: 1,265 of the 2,132 candidate–repository cells carry a lexical cue. For 16 structurally exact candidates we then walk the same cells up an evidence ladder. Under the coding of one model with 200 prompt profiles, six gain majority support for the pointcut and four keep support for a complete contract under every profile. The audit is purposive and model-assisted. We use it for three things: an instrument with a frozen expression, specification, and negative control for every record; worked examples of detection and tracking; and a sampling frame for independent adjudication. It is not an estimate of how common patterns or contracts are.

The paper develops this argument at three levels.

- At the **normative** level, consequential harness responsibilities should remain identifiable, authorized, and auditable through change.
- At the **conceptual** level, the A^2 taxonomy and responsibility record connect protected contracts and permitted variation across design, verification, and evolution.
- At the **empirical** level, three hypotheses turn the framework into a research program: **H1**, inspectable commitments rather than particular prompts or code are the durable asset; **H2**, agent-led evolution requires mechanized separation of concerns; and **H3**, explicit variation boundaries improve self-modification relative to unconstrained revision.

Nine analytical cases and three structural drifts examine the distinctions behind these hypotheses, while prospective protocols state how their empirical claims could be supported, rejected, or left unresolved (Sections 4–6).

The supporting material keeps this argument connected to practice. A 52-record seed inventory supplies a detection and matching rule for every record, representative UML class views reconstruct proposed contracts, and an exploratory audit of 41 repositories exercises the evidence ladder on real code (Sections 3.1 and 3.3; Appendices A, A.4, and B). The conceptual contributions can be examined through the cases; the causal claims await the prospective studies; and the audit remains exploratory. Each claim therefore carries its own evidence level (Table 19).

Section 2 reviews harness engineering, the repository panel, software design patterns, and empirical methods. Section 3 presents the candidate inventory, exploratory audit, and A^2 taxonomy. Section 4 develops the three major hypotheses, and Section 5 analyzes the structural drifts that motivate them. Section 6 gives the prospective protocols; Sections 7–9 address alternative views, implications, and conclusions. The appendices contain the complete inventory, analytical class views, audit protocol, and a consolidated account of limitations and future work.

2 Background and Related Work

Three bodies of work frame the argument. We first define the harness as the object of study and the repository panel used throughout the paper, then recall how software design patterns separate the assets a design keeps stable from the activities that use them, and finally state the empirical standards by which the audit and the prospective protocols should be judged. Together these set up the open problems that the taxonomy is meant to resolve.

2.1 Harness Engineering and Representative Repositories

Let M be a fixed model, $t \sim \mathcal{T}$ a task, $s_0(t)$ its initial state, and B a budget. A harness H is the executable runtime and configuration that mediates interactions among M, tools, and an environment [19, 31]. Together they induce a trajectory distribution $\tau \sim P_H(\tau \mid t, M, s_0, B)$. An evaluator E maps a trajectory and resulting state to outcomes. Holding M, t, s_0, B, and E fixed while changing H defines a harness contrast rather than a model contrast. Estimating a causal harness effect additionally requires randomized or matched repeated runs, version-pinned components, and run-level uncertainty.

Model reports document the model side of this boundary [33, 17], including distinct context regimes [18]. Model-level system cards characterize model risks [34]; a harness's enforcement is validated at the system level. Functionally, H assembles instructions and context, exposes tool schemas, executes actions under permissions, maintains state, and controls retries or termination. Tool use, feedback, and external environment state show why these are operational rather than purely model-level choices [40, 42, 52]. The boundary

admits learned components: a learned router or model-based reviewer is inside H when it is part of the system under study.

Motivated by agent-evaluation and harness-cost analyses [56, 1], we propose a minimal reproducibility record: model and decoding configuration; instructions and context assembly; tool and environment versions; state and memory policy; permission and stopping rules; evaluator; and resource budget. It must separate task-time actions from development-time changes. Section 5.1 operationalizes this record and its lifecycle use.

Engineering practice has converged on a small vocabulary of runtime structures. Anthropic's guide and reference implementations describe prompt chaining, routing, parallelization, orchestrator–workers, and evaluator–optimizer workflows [2, 3]. OpenAI's guide adds manager coordination and decentralized handoffs [35]. The Model Context Protocol literature reuses Mediator, Observer, publish–subscribe, and Broker for agent communication [39]. Coding agents add execution loops, tool registries, permission layers, durable session state, and stop-time verification [6]. Section 3.1 turns this vocabulary into a source-indexed inventory.

Object-oriented design gives this practice a structure for change. OO-Harness places shared behavior in a base harness, specializes it in subclasses, and dispatches tasks through a polymorphic router [25]. Composition is the alternative: HarnessX builds harnesses from reusable components [13]. HarnessForge adapts runtime structure and policy together [12]. The design question is classical—inheritance versus composition around a stable interface [16]—but the author of the subclass or component may now be an agent. In our panel, cues for inheritance and polymorphic routing (record E01) appear in 33 of 36 repository families. The study protocol located no official OO-Harness repository, so these cues reflect inheritance in public harness classes and not the OO-Harness system itself.

To see how this practice appears in code, we study 41 open-source repositories grouped into 36 repository families (Table 2). A *family* groups repositories from one lineage—a fork, a successor, or a split of one project—so that shared ancestry counts once. Four families hold more than one repository: OpenHands (three), Cline and Roo Code, OpenAI Swarm and the Agents SDK, and AutoGen and Agent Framework. The panel spans Python (25 repositories), TypeScript (13), Rust (2), and JavaScript (1). It covers coding agents (19), general assistants (17), and browser, research, embodied, and mixed-domain systems (5), with single-agent (10), multi-agent (15), mixed (10), and research-specific (6) architectures.

Repository selection followed a frozen protocol. A repository is eligible when it contains executable source for an LLM-based agent runtime or harness that mediates at least one responsibility: model invocation, tool execution, state, permission, delegation, evaluation, recovery, context, or termination. Documentation-only, prompt-only, benchmark-only, generated-scaffold, and simple API-client repositories are excluded; archived repositories remain eligible as a boundary condition. Candidates came from declared search queries and a curated channel seeded by this paper's source index. They were deduplicated by repository identity and clustered into families before selection. Discovery prioritized architecture and evolution signals, and star count was recorded only as descriptive metadata. The core frame holds 24 families (29 repositories), stratified for diversity across core runtimes, orchestration frameworks, and configuration-driven harnesses; across Python and TypeScript; and across single- and multi-agent architectures. The protocol records that pattern outcome counts were not used to add, remove, or replace non-pilot families. The frame keeps families with a visible architecture event—a runtime-boundary change in OpenHands, the Cline and Roo Code fork divergence, the Swarm successor, the AutoGen migration, and configuration shifts in ECC and Langflow—because such events carry lifecycle evidence. Two purposive supplements were

Table 2 Repository panel: 41 repositories in 36 repository families. Parentheses group the repositories of one family, which share a lineage and count once in family-level statistics. The core frame was frozen first; the two supplements were added purposively afterwards. Research-supplement repositories are the official code named by the cited paper or its project page.

Pool and stratum	Fam.	Repos.	Repositories
Core frame (24 families, 29 repositories)			
Python core runtimes	4	6	(OpenHands, OpenHands software-agent-sdk, OpenHands automation), SWE-agent, Aider, browser-use
TypeScript core runtimes	7	8	(Cline, Roo Code), OpenCode, Gemini CLI, pi, deepseek-harness, OpenClaw, Orca
Python orchestration	10	12	(OpenAI Swarm, OpenAI Agents SDK), LangGraph, (AutoGen, Agent Framework), CrewAI, MetaGPT, Google ADK, smolagents, Pydantic AI, Agno, DeerFlow
TypeScript orchestration	1	1	Mastra
Configuration-driven and visual	2	2	ECC, Langflow
Research supplement (6 families, 6 repositories)			
Evolution mechanisms	5	5	Harness^2 [53], RRSI [51], HarnessX [13], HarnessForge [12], Agentic Harness Engineering [26]
Embodied assistance	1	1	CarryOn [47]
Implementation supplement (6 families, 6 repositories)			
Implementation and lineage variants	6	6	Codex, Goose, Continue, Qwen Code, mini-SWE-agent, OpenAI Agents JS
Total	36	41	

added after the frame was frozen: six official research repositories for evolution mechanisms named in the literature, and six repositories chosen for implementation variation and lineage comparison, including a fork ecosystem, a simplified successor, and a cross-language variant. The 36-family panel is therefore purposive. Every count in this paper describes the panel and is not a prevalence estimate for agent software.

2.2 Design Patterns in Software Engineering: Assets and Activities

A design pattern names a recurring problem in a context, a solution structure, and its consequences [16, 8]. We sort the software-engineering literature into two streams, because the taxonomy of Section 3.4 crosses them. The *assets stream* asks what a design keeps stable and what it leaves open. The *activities stream* asks how engineers use that distinction when they design and implement, test and verify, and maintain and evolve a system. Table 3 sorts the works we build on.

In the assets stream, the first asset is the *protected contract*. Parnas's criterion decomposes a system so that each module hides a design decision that is likely to change behind an interface that is not [37]. The Gang-of-Four and POSA catalogs make the stable part explicit as participant roles and collaboration rules [16, 8, 41]. Domain-driven design protects a model's language and boundaries [14], and resource-management patterns state acquisition and release obligations [22]. In each case the reusable asset is a contract: an interface together with the invariants that its clients may rely on.

Table 3 Software-engineering foundations sorted into the assets and activities streams. The assets stream states what a design protects and where it may vary; the activities stream states how that distinction is used across the lifecycle and how it is studied empirically. The last column names what each stream leaves open for harnesses. The assignment of works to streams is our reading.

Stream and coordinate	Representative works	Contribution used here	Left open for harnesses
Assets stream: what is protected and what may vary			
Protected contract	Modular decomposition [37]; GoF and POSA catalogs [16, 8, 41]; domain-driven design [14]; resource management [22]	An interface and its invariants hide the decisions that are likely to change.	Contracts over what a model may observe, do, and approve; authority to revise them.
Permitted variation	GoF hooks and POSA interceptors [16, 41]; aspects and concerns [21, 43]; pattern languages [10, 9]	Hot spots, join points, and connections among patterns name where change is welcome.	Variation chosen at run time by a stochastic model or edited by an agent.
Activities stream: how the assets are used and studied			
A1 Design and implementation	Pattern catalogs [16, 8]; modular criteria [37]	A shared vocabulary and a default decomposition before code exists.	Designs that assign context, tools, and control to agent roles.
A2 Testing and verification	Test seams [15]	A variation point doubles as a control point for tests.	Checks that the tested system can bypass or rewrite.
A3 Maintenance and evolution	Laws of software evolution [24]; resource lifecycles [22]	Continuing change is expected; patterns localize it.	Change proposed by the system itself; value that ends when the model changes.
Empirical study of assets and activities	Pattern detection [45]; repository mining [20]; study design and sampling [38, 5, 30, 23]	Recover assets from artifacts; observe activities in histories; bound claims by sample and evidence level.	Evidence levels that separate a cue from a contract and a contract from an outcome.

The second asset is the *permitted variation*. The same catalogs name where change is welcome. Strategy, Template Method, Adapter, and Observer expose hooks, and interceptors and component configurators extend a running system [16, 41]. The aspect-oriented tradition that began with Kiczales et al. [21] contributes the vocabulary we use for detection: a *join point* is a place in execution where behavior can attach, and a *pointcut* selects the join points to which a concern applies. Multi-dimensional separation of concerns generalizes the idea that one system can be decomposed along several concerns at once [43], and pattern languages connect individual patterns into sequences and alternatives [10, 9].

The activities stream follows the lifecycle. In *design and implementation* (A1), patterns give designers a shared vocabulary and a default decomposition: fix the contract, expose the variation point, and name the consequences before code exists [16, 8]. In *testing and*

verification (A2), Feathers defines a seam as a place where behavior can be altered without editing in that place [15]. A seam lets a test substitute a dependency, control inputs, and observe outputs. The variation point of design thus becomes the control point of verification. In *maintenance and evolution* (A3), Lehman's laws observe that a system in use must be continually adapted and that its complexity grows unless work is done to contain it [24]. Patterns localize that adaptation. A change that fits a declared variation point is routine; a change that does not fit signals that the contract must be redesigned.

Pattern thinking has already reached agents (Table 4). Read through the two streams, this work is strongest on assets: it catalogs what recurs and how patterns connect. Its treatment of activities is partial. Individual studies address pattern evolution, lifecycle framing for one concern, or validation gates, and to our knowledge none carries one record through design, verification, and evolution. Recent harness surveys organize the domain by components, control–agency–runtime boundaries, or interfaces and mechanisms [28, 19, 31].

We take what these lines establish as given. Pattern catalogs for agents exist, and natural-language harness documents already state run-level policy, validation gates, and artifact contracts [36]. Explicit patterns and explicit contracts are therefore not our claim. Our contribution is the synthesis that connects them: a responsibility record that adds authority, acceptance evidence, violation consequences, and lifecycle conditions to a mechanism description; the A^2 organization that follows a record through design, verification, and evolution; an evidence ladder that keeps retrieval, exemplars, pointcuts, contracts, trajectories, and outcomes apart; and prospective protocols that separate the value of enforcement from the value of records.

2.3 Empirical Methods in Software Engineering and Repository Mining

The exploratory audit in Section 3.3 and the protocols in Section 6 take their method from empirical software engineering. The two streams apply here as well: some methods recover *assets* from artifacts, and others observe *activities* in histories.

In the assets stream, design-pattern detection recovers pattern instances from source structure, for example by scoring the similarity between a system's class graph and a pattern's role graph [45]. Its lesson for harnesses is that a detector returns *candidates*: a structural or lexical match must still be adjudicated against the pattern's intent. We therefore separate a retrieval cue from a pointcut match, and a pointcut match from a supported contract.

In the activities stream, repository mining treats version control, releases, and project metadata as data about how software is built and changed. Kalliamvakou et al. document its perils on GitHub: many repositories are not engineered projects, and forks duplicate history [20]. Our eligibility rule and family clustering respond to both. Lehman's laws motivate following a record across releases instead of reading it at one snapshot [24].

For study design, case-study guidelines ask for a protocol, multiple sources of evidence, and a chain from data to claim [38]. Sampling guidance distinguishes probability samples from purposive ones and cautions against generalizing from the latter [5], and the diversity of a sample can be reported explicitly [30]. Evidence-based software engineering asks that practice rest on critically appraised evidence [23]. We adopt a frozen protocol, a diversity-oriented purposive panel, and an explicit evidence ladder, and we report the audit as exploratory.

Table 5 orders the evidence used in this paper: a retrieval cue, a canonical source exemplar, a pointcut and contract specification against which an implementation can be adjudicated, a lifecycle trajectory, and a causal experiment. A level never implies the one above it. The exploratory audit of Section 3.3 reaches the lower levels for one purposive panel, with model-assisted coding at the pointcut and contract level. Independent adjudication and the

Table 4 Positioning this engineering synthesis against direct prior work. The distinctions state complementary scopes. "Explicit" means the feature is central to the cited work rather than merely compatible with it.

Work	Primary object and connection	Lifecycle / validation	Distinction relevant here
Taylor et al. [44]; Weyns [49]	Agent-based and self-adaptive multi-agent architectures; interrelated architectural patterns.	Pattern mining, use, and evolution; case illustrations.	Prior agent-architecture precedents on which harness-level records build.
Cai et al. [11]	Quality attributes, 16 design patterns, and rationale mined from 94 LLM-based multi-agent software-engineering papers.	Systematic descriptive study; no connected harness lifecycle proposed.	Establishes a stronger empirical catalog baseline than isolated records.
Yu [58]	Twelve connected NFR patterns for security, reliability, observability, and cost, including agent-specific aspects.	Goal-to-aspect method and an open-source framework case study.	Relevant agentic quality-attribute work with complementary scope.
Milošević and Rabhi [29]	Design patterns for governed agentic communities, roles, protocols, and human–agent accountability.	Formal framework and clinical-trial-matching case study.	Centers organizational governance; our unit is the executable harness contract and its permitted variation.
Billiris and Gill [7]	A federated observability architecture pattern for changing telemetry.	AI-SDLC framing and a controlled proof of concept.	Provides lifecycle depth for one concern; our proposal supplies a common record schema across multiple harness responsibilities.
Pan et al. [36]	Executable natural-language documents for run-level harness policy, modules, handoffs, state updates, validation gates, and artifact contracts.	Shared-runtime comparisons, module ablations, and results across coding, terminal-use, and computer-use benchmarks.	Direct harness-representation neighbor with empirical evidence; it motivates our emphasis on authority, evidence, and lifecycle controls.
This paper	A pattern-discipline position: responsibility records, the A^2 taxonomy, nine analytical cases, and three structural drifts.	Promotion obligations spanning pointcut, contract, variation, evidence, and lifecycle; A1 design, A2 verification, and A3 evolution/retirement.	Adds authority, acceptance evidence, and lifecycle conditions to mechanism descriptions, with an exploratory audit and three prospective protocols.

Table 5 Evidence ladder, and how far the exploratory audit climbs it. Each level answers its own question and never implies the level below it in the table. The audit results describe one purposive panel.

Level	Unit	Status in this paper	What the level shows
Repository retrieval	Repository–pattern cell	1,265 of 2,132 cells carry a lexical cue; 1,246 in the current snapshot	The vocabulary of a candidate is present in the code; pattern presence is a later level
Canonical exemplar	Pattern	Source-backed exemplar for 51/52 patterns	At least one concrete source location anchors each supported pattern in the atlas
Pointcut/contract specification	Pattern definition	52/52 specifications with context, trigger, structure, effect, and contract fields	The comparison instrument is complete enough to adjudicate implementations
Model-coded pointcut and contract	Exact structural candidate cell	16 matcher-selected cells coded by one model under 200 prompt profiles: six majority-supported pointcuts, four with a complete contract under every profile	The coding step can be carried out and is stable to prompt wording in most cells; its correctness awaits independent adjudication
Independently adjudicated pointcut and contract	Stratified sample of cells	Specified as the next study (Section 8.4)	Cue-to-pointcut precision and recall, and contract support, with human agreement
Lifecycle trajectory	Candidate–repository trajectory	621/1,272 trajectories have at least two observed checkpoints	Large source-path changes and movement are observable in sampled histories; cue gaps are tracked as detector gaps
Causal experiment	Matched task or participant	H1–H3 designs with decision rules; outcome data to be collected	The record-value, external-enforcement, and controlled-variation hypotheses are specified for execution

causal level are specified as next steps (Sections 8.4 and 6). Appendix B.1 gives the audit's evidence model, unit of analysis, and statistics.

2.4 Open Problems

The literature establishes that harness mechanisms recur, that their contracts matter, and that repository histories can be studied empirically. It leaves a connected set of problems unresolved. Catalogs and guides still move more readily from mechanism names to implementations than from implementations to the responsibilities, variation rights, and authority that should survive change. Detection faces a parallel gap: without a shared evidence ladder, a retrieval cue can be mistaken for an instance, and an instance for a

supported contract. Lifecycle research adds a third difficulty because design, verification, and evolution are rarely traced through the same record.

These gaps become more consequential when the maintainer is itself an agent and when a model upgrade changes the value of a mechanism without changing the external obligation. Finally, neither the value of external enforcement nor the value of complete records follows from recurrence alone; both require controlled comparison. The A^2 taxonomy addresses the representational and lifecycle gaps, the evidence ladder disciplines detection, and the three major hypotheses of Section 4 turn the remaining questions into tests of assets, authority, and self-modification.

3 Harness Design Patterns and the A^2 Taxonomy

This section addresses the first three open problems. Section 3.1 shows that the literature already supplies harness design patterns. Section 3.3 reports an exploratory audit of where their vocabulary appears in repositories and how its location changes. Section 3.4 defines the A^2 taxonomy and the responsibility record, which are the core of the section. Section 3.5 uses them to map, detect, track, and analyze patterns through case studies.

3.1 Design Patterns Already Exist: Families and Representative Records

Harness design patterns do not need to be invented. The literature already names them, and we take each definition from the source that reports it. We call the entries *records*, because they differ in standing: some are recurring patterns in the classical sense, the coding-agent labels are anchored to a source-code study of eleven harnesses, and the evolution entries are mechanisms instantiated by one research system each. Our inventory holds 52 records in five source families: 29 coding-agent mechanisms (P), seven official-guide workflows and coordination patterns (W), four communication patterns (C), two reasoning and feedback patterns (R), and ten evolution mechanisms (E). Table 6 lists representative records with their source descriptions, and Appendix A gives all 52 by category.

The P family covers the runtime core of coding agents: execution loops, tool handling, permission and sandbox layers, durable state, and stop-time checks. The W family holds the five workflows of Anthropic's guide and the two coordination forms of OpenAI's guide. The C family holds the four classical communication patterns that the Model Context Protocol literature applies to agents. The R family holds the reason–act loop and verbal reflection. The E family holds research mechanisms for harness evolution, from inheritance-based specialization to regularized and multi-objective search. The families record where a description comes from; they do not add an axis to the taxonomy.

This is a scoped, argument-driven inventory: the union of named patterns, official-guide workflows, communication patterns, reasoning mechanisms, and evolution mechanisms used in the paper's comparisons. These five provenance families organize the inventory; they do not add an axis to A^2. Activity and asset coordinates are assigned across families as part of the paper's analysis. We include a record when a cited source names a recurring solution or workflow, or when a research system instantiates a mechanism needed for the evolution analysis. We exclude generic component names without a recurring decision, uncited examples, and additional relabelings introduced only to increase coverage. Records remain separate when their authority, state, or variation contracts differ, even if their surface structures overlap. The record count describes this inventory; frequency estimates come from the audit.

P01–P29 are anchored to the eleven-system source-code study of Barbaste et al. [6], whose catalog reports 29 recurring patterns. We checked each record against that paper (arXiv:2609.00006v1): The online inventory gives a section and page locator for every P record (`https://xhyccc.github.io/pohe/inventory.html`). Seventeen records use the source's label or a shortened form of it, and eight adapt or merge neighboring source mechanisms. P03 generalizes one system's loop, P10 is our synthesis of the paper's edit-error and output-truncation findings, P11 recasts its "disappearing boundary between tool and workflow" as a boundary to declare, and P29 joins harness mimicry with harness hosting. These readings are ours. W01–W05 follow Anthropic's guide and official reference implementations [2, 3]; W06–W07 follow OpenAI's manager and handoff guidance [35]. C01–C04 are classical communication patterns discussed in the MCP survey [39]. R and E records cite the corresponding papers. E records are instantiated research mechanisms, not claims of independently established recurring patterns; E08–E09 are adjacent embodied settings. Google's architectural components [50], surveys, and benchmarks are not relabeled as additional patterns merely to increase the count.

The records vary in granularity and evidence level, and some describe the same idea at different granularities. Completeness here means coverage of this declared inventory. The inventory is the source-indexed basis for the audit below and the seed for the registry requested in Section 8.

3.2 Worked Record: P01, Minimal Linear Loop

A complete record is more than a name. P01, *Minimal linear loop*, shows what the inventory asks a reader to inspect. The record is one of the 29 coding-agent mechanisms anchored to the source-code study of Barbaste et al. [6]. Its problem is to alternate model turns and action execution without losing the turn boundary; its join point is `agent_turn_execution`. The pointcut applies when a harness executes an agent with at least one model or action turn. The trigger is a model response that requests tools or another turn. The structure is an ordered sequence of turn and result records plus an explicit repeat-or-terminate decision. The effect is that every turn yields a terminal result, a tool action, or another legal turn.

Table 7 states the contract, the permitted variation, the test obligation, and the negative control. Figures 2 and 3 show the same record as a UML class view and as an object-interaction view. The first view names participant roles and the calls each receives; the second places those calls in a numbered collaboration and message sequence. Both are analytical reconstructions of the proposed contract, not observed execution traces. Figure 4 adds the two design aids used throughout the field guide: a prompt sketch and a small code sketch. They are teaching aids, not repository excerpts or conformance claims.

At the frozen audit commit, the canonical exemplar is the run loop in `openai/openai-agents-python` (`https://github.com/openai/openai-agents-python/blob/86bd29878b30062757fdb4f32fb95eb8eccc7e36/src/agents/run_internal/run_loop.py#L987`); the accompanying test at `tests/test_agent_runner.py:1` supplies support. The field-guide record also links four further public files retrieved by the frozen lexical cue. Those retrieval cues identify vocabulary to inspect; they do not establish that the full contract is implemented.

Table 6 Representative records from the five source families of the 52-record inventory. The mechanism column paraphrases the cited source and is the definition used in this paper. "Fam." counts the 36 repository families with a retrieval cue for the record (Section 3.3); it is not pointcut or contract support. Appendix A lists all 52 records by category.

ID	Record	Source-reported mechanism	Source	Fam.
Coding-agent mechanisms (P; 29 records; Barbaste et al.; source-located labels)				
P01	Minimal linear loop	Alternate model turns and action execution.	[6]	35
P04	Middleware-pipeline loop	Layer behavior around model/tool calls.	[6]	34
P14	Event-sourced conversation engine	Reconstruct state from recorded events.	[6]	31
P17	Verify-on-stop guards	Check artifacts before accepting completion.	[6]	8
P18	Hybrid loop detection	Detect stalled or repetitive behavior.	[6]	8
P21	Syntax-aware command permissioning	Inspect command structure before execution.	[6]	2
P23	Layered permission and sandbox	Combine policy decisions with execution isolation.	[6]	36
P26	Session-tree version control	Branch or revisit execution history.	[6]	30
P27	Lineage and threshold compaction	Bound history while retaining useful state.	[6]	35
Official-guide workflows and coordination (W; 7 records; Anthropic and OpenAI guides)				
W02	Routing	Select a specialized path for an input.	[2, 3]	36
W04	Orchestrator–workers	Generate and delegate subtasks dynamically.	[2, 3]	30
W05	Evaluator–optimizer	Revise an artifact using iterative feedback.	[2, 3]	17
W07	Decentralized handoffs	Transfer control to another specialized agent.	[35]	29
Communication patterns (C; 4 records; Sarkar and Sarkar)				
C01	Mediator	Coordinate interactions through an intermediary.	[39]	5
C02	Observer	Notify registered dependents of changes.	[39]	29
C03	Publish–subscribe	Decouple producers from consumers.	[39]	24
C04	Broker	Mediate discovery and invocation across boundaries.	[39]	22
Reasoning and feedback (R; 2 records; Yao et al. and Shinn et al.)				
R01	Reason–act–observe (ReAct)	Interleave reasoning and external actions.	[55]	35
R02	Verbal reflection memory	Reuse feedback across attempts.	[42]	1
Instantiated evolution mechanisms (E; 10 records; one research system each)				
E01	Inheritance and polymorphic routing	Specialize related harnesses.	[25]	33
E02	Propose–Probe–Compose	Improve a harness using contrastive probes.	[53]	6
E03	Regularized harness improvement	Limit overfitting during repeated edits.	[51]	12
E04	Multi-objective evolution and composition	Search among competing objectives.	[57]	8
E05	Composable harness foundry	Construct and evolve reusable harness components.	[13]	2
E10	Observability-driven harness evolution	Use traces to guide candidate changes.	[26]	1

Table 7 The P01 record. Pointcut, protected contract, variation, test obligation, and negative control, transcribed from the frozen specification.

Field	Statement
Context	A harness executes an agent with at least one model or action turn.
Trigger	A model response requests tools or another turn.
Structure	Ordered turn/result records and an explicit repeat-or-terminate decision.
Effect	Every turn yields a terminal result, a tool action, or another legal turn.
Requires	Model output, action dispatch, and a termination predicate are available.
Ensures	Actions are correlated with their turn and a terminal outcome is returned.
Invariant	Each completed turn has one recorded decision and no unexplained action.
Violation	Premature stop, malformed action, or repeated retry must terminate or surface an error.
Variation	Prompt, tool set, stopping predicate, and turn-result encoding.
Test obligation	Test normal completion, immediate malformed output, and retry exhaustion.
Negative control	A one-shot model call with no action cycle does not satisfy the pointcut.

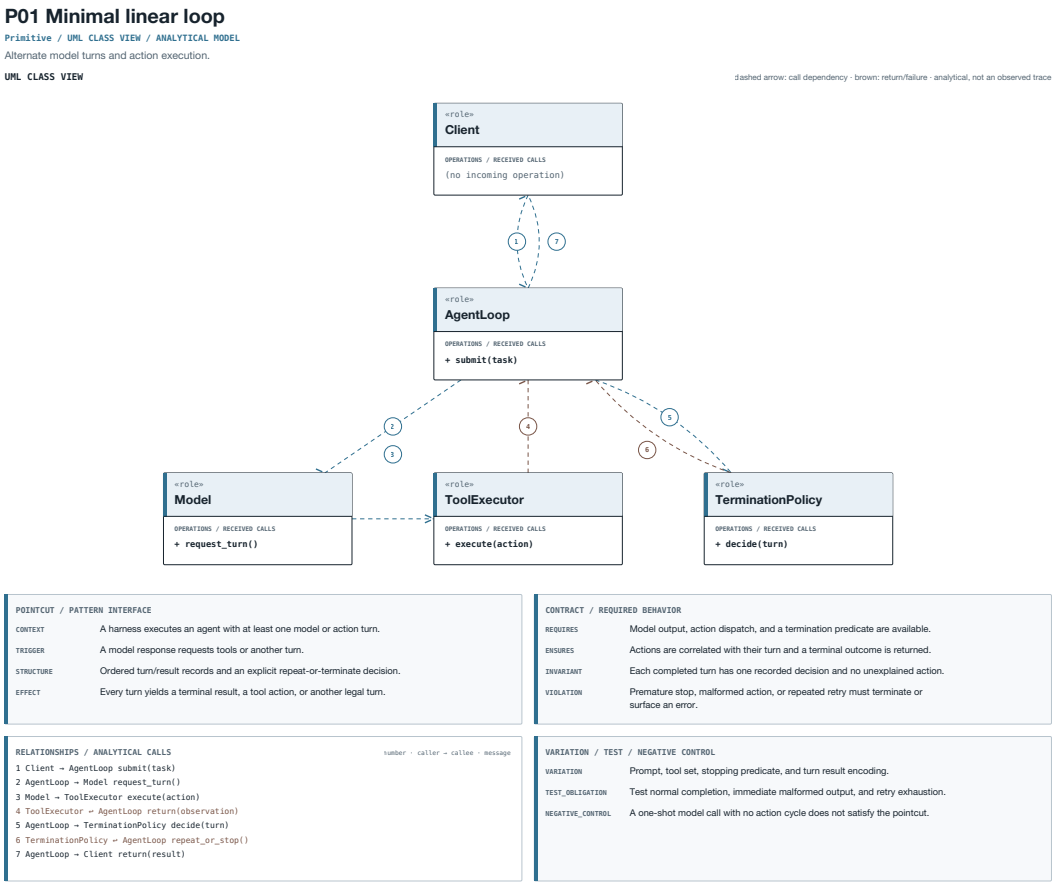


Figure 2 UML class view of P01. Participant roles, the operations each role receives, and numbered call dependencies. Dashed arrows are calls; brown arrows are returns or failures. The view is an analytical model of the proposed contract, not a repository class declaration.

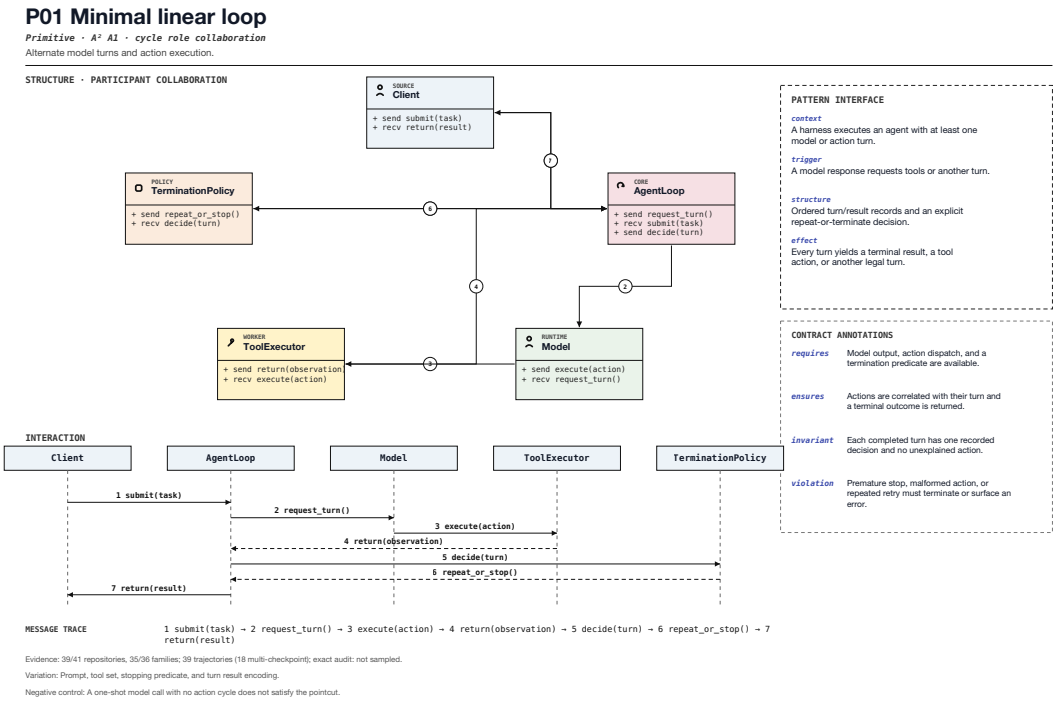


Figure 3 Object-interaction view of P01. The same roles as a participant collaboration and a numbered message sequence, with the pointcut, contract, variation, and negative control repeated alongside the trace. The sequence is an analytical reconstruction, not an observed execution trace.

Prompt sketch. The prompt states the loop, the record boundary, and the terminal obligation in operational terms.

```
Work in a bounded loop. On each turn: call the model, execute any requested
tool actions, record each observation, then ask the termination policy whether
to continue. Return exactly one terminal result, or stop with a typed
exhaustion error.
```

Code design sketch. The runner, rather than the prompt, owns turn identity and the terminal decision.

```
def run(task, model, tools, policy):
    history = [user_message(task)]
    for turn in range(policy.max_turns):
        reply = model.respond(history)
        history.append(reply.as_message())
        for action in reply.tool_calls:
            observation = tools.execute(action, turn=turn)
            history.append(observation.as_message())
        decision = policy.decide(history, turn)
        if decision.stop:
            return decision.result
    raise LoopExhausted(turns=policy.max_turns)
```

Figure 4 Illustrative prompt and code design for P01. The sketches name an interface and a control boundary; they are not an implementation of the canonical exemplar or evidence that any repository conforms.

3.3 Exploratory Support: Candidate Cues in Repositories and How Their Locations Change

The inventory comes from the literature. To see how it meets real code, we audited the 52 records across the 41 repositories (36 families) of Section 2.1, at the current snapshot and at sampled historical releases. This subsection reports what we observed (Figure 5, Table 8). How the audit was run and checked is in Appendix B: the retrieval and matching procedure, the sensitivity analyses, the model-assisted coding, and the statistics. The observations describe this panel of repositories.

3.3.1 Cue Coverage: The Candidate Vocabulary Is Readily Retrieved

The vocabulary of the inventory is widely present in harness code. A lexical cue appears in 1,265 of the 2,132 candidate–repository cells, 1,246 of them in the current snapshot, and a curated exemplar anchors seven more. The median family holds cues for 35 of the 52 records (IQR 28.2–37.0), and six records drew current cues in all 36 families: P02, P10, P12, P23, P25, and W02. These six are concurrency, delegation, sandbox, scope, and routing mechanisms, the shared vocabulary of harness code. A cue marks where a pattern may be; whether the pattern is there is the next question (Section 3.3.2).

Coverage follows a clear gradient across record families (Figures 6 and 7). Mean family cue coverage is 76.9% for coding-agent records and 67.1% for workflow records, 55.6% for communication, 50.0% for reasoning, and 19.2% for evolution records. The mechanisms that run a session are named in nearly every harness, while the mechanisms that change a harness over time are named in few.

The same contrast appears across strata (Table 9). Evolution records have current or historical cues in 18.0% of cells, against 74.5% for primitive records. Python repositories yield 53.8%, TypeScript 67.9%, and Rust 70.2%; source pools differ less (53.5%–64.1%).

Organized by the activity coordinates of Appendix A (Table 10), records with a design coordinate are the most widely cued (74.6%) and records with a verification coordinate the least (44.1%). The verification records split sharply. Layered permission and sandbox (P23) and scope-based authorization (P25) draw cues in all 36 families. Verify-on-stop guards (P17), hybrid loop detection (P18), and the policy floor (P22) draw cues in 8 each, and syntax-aware command permissioning (P21) in 2. Harnesses widely name the boundaries of what an agent may do and far less often name checks on whether the agent's work is done.

3.3.2 From Cues to Structural Candidates: Retrieval, Pointcut, and Contract Separate

Most observed cells rest on a current lexical cue (1,246 of 1,272). With a stricter rule of at least two current code paths plus support, 857 cells remain. Exact symbol-level matching narrows the set to 16 structural candidates, and coding these against their pointcut and contract fields separates them further. The pointcut held in six cells: hybrid loop detection (P18) in Pydantic AI and Roo Code, syntax-aware command permissioning (P21) in Qwen Code, OpenCode, and OpenClaw, and R02 in CarryOn. It did not hold in eight and stayed unresolved in two, P17 in DeerFlow and P21 in Gemini CLI. Five of the six held cells recorded all four contract fields: requires, ensures, invariant, and violation behavior. Context and trigger were clear in every cell; the open question was effect, that is, whether the mechanism actually changes control flow.

Retrieval, structural match, pointcut, and contract are thus different questions that receive different answers on the same code. A keyword shows where to look. A pointcut shows

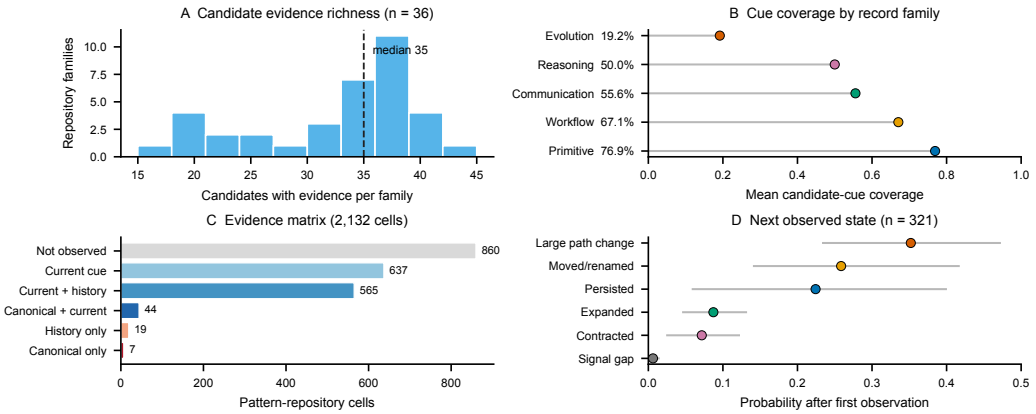


Figure 5 Observed candidate cues, evidence provenance, and lifecycle signals in the selected panel. (A) Candidate-cue richness across 36 repository families. (B) Mean family cue coverage by record family; "Primitive" denotes the coding-agent (P) records. (C) Composition of the complete pattern–repository matrix; unobserved cells remain explicit. (D) Row-conditional next states after a candidate trajectory's first observation, with 95% repository-cluster bootstrap intervals. Family, rather than cell, is the independent unit. All results are descriptive for the selected frozen panel.

that a mechanism intercepts the right join point. A contract shows what the mechanism promises when it fires. A responsibility record makes this distinction explicit. The six held cells, P21 in particular, are concrete exemplars of fully specified records. Appendix B.3 gives the per-cell evidence and its robustness.

3.3.3 Co-occurrence and Cue Trajectories: Cue Locations Move

Records co-occur in structured ways. Twenty-two of 1,326 candidate pairs were flagged at $q < 0.05$. The strongest pair, event-sourced conversation state with observer notifications, appears together in 29 of 36 families ($\phi = 0.817$). Shared architecture and repository lineage are plausible sources of such pairings.

Cue locations change often across releases. Across 1,272 trajectories and 2,611 frozen observations, 321 transitions followed a first observed cue (Table 11): 35.2% were large cue-path changes (95% family-cluster interval [21.3%, 47.3%]), 25.9% moves or renames, and 22.4% persistence. They come from 20 repositories in 16 families.

At least 398 trajectories (31.3%) contain a move, large path change, expansion, or contraction. Once a cue persists, it tends to keep persisting ($67/75 = 89.3\%$). Contracted cue paths often expand again ($16/33 = 48.5\%$), so cue locations do not change monotonically. No explicit retirement record was observed. The practical reading is that the code a record points to is reorganized often, which is why a record carries a lifecycle field.

3.3.4 How the Paper Uses the Audit

Taken together, the observations support cue-level recurrence as a descriptive premise. Every canonical candidate except E08 has a source locator, and the median family holds cues for 35 of the 52 records. On the 16 exact candidates, retrieval, pointcut, and contract separate. Records co-occur in structured ways, and the code they point to moves often across releases. How such movement affects task quality, safety, cost, or reliability is an outcome question.

Table 8 Key descriptive results from the frozen repository audit. Counts and intervals describe the selected panel. Current cues are retrieval signals, canonical locators are one atlas exemplar per pattern by design, and support locators point to tests, documentation, or a specification. S/R/U denotes supported, rejected, and unresolved candidate pointcuts under the indicated rule. All quantities are descriptive.

Measurement	Result
Repositories / independent families	41 / 36
Current-cue cells	1,246 / 2,132
Retrieval-cue cells (current or historical)	1,265 / 2,132
Canonical-locator-only cells	7 / 2,132
Cells with any evidence layer	1,272 / 2,132
Patterns with canonical exemplar	51 of 52
Atlas support: test / docs / spec.	44 / 7 / 1
Pointcuts: majority / invariant (S/R/U)	6/8/2 / 4/8/4
Median family coverage	35 of 52 (IQR 28.2–37.0)
Exploratory co-occurrence flags	22 of 1,326
Multi-checkpoint realizations	621 of 1,272 (48.8%)
Large path change after first observation [95% CI]	35.2% [23.3–47.3]

Table 9 Retrieval coverage by corpus stratum. Counts are pattern–repository cells with any current, historical, or canonical evidence. They are descriptive retrieval yields, not pointcut prevalence; repository families remain the independent unit.

Stratum	Value	Observed cells	Share
record family	Communication	86/164	52.4%
	Evolution	74/410	18.0%
	Primitive	886/1189	74.5%
	Reasoning	40/82	48.8%
	Workflow	186/287	64.8%
source pool	frozen	905/1508	60.0%
	implementation supplement	200/312	64.1%
	research supplement	167/312	53.5%
language	JavaScript	41/52	78.8%
	Python	699/1300	53.8%
	Rust	73/104	70.2%
	TypeScript	459/676	67.9%

The paper uses the audit in three ways. It supplies an instrument: 52 specifications with expressions, aliases, exemplars, and negative controls (Appendix A). It supplies worked examples for the detection and tracking case studies of Section 3.5. And it supplies a sampling frame for the next evidence step, an independently adjudicated, stratified sample that estimates cue-to-pointcut precision and recall (Section 8.4). The argument of this paper does not rest on the size of any gap between cues and contracts, and we do not claim to have measured one.

Because the instrument bounds what the audit can support, we also audited the cue path and made it reproducible (Appendix B.5). A deterministic runner emits cues sorted by stable identifier with no generation timestamp and records observability and truncation ledgers; repeated runs on the same inputs are byte-identical. A gated loop on labeled development fixtures verified three implementation defects—missed `checkpoint_` identifiers, a bare `route` alternative that matches configuration strings, and cues produced by comment-only mentions—and retained a combined candidate only because no positive regressed, every

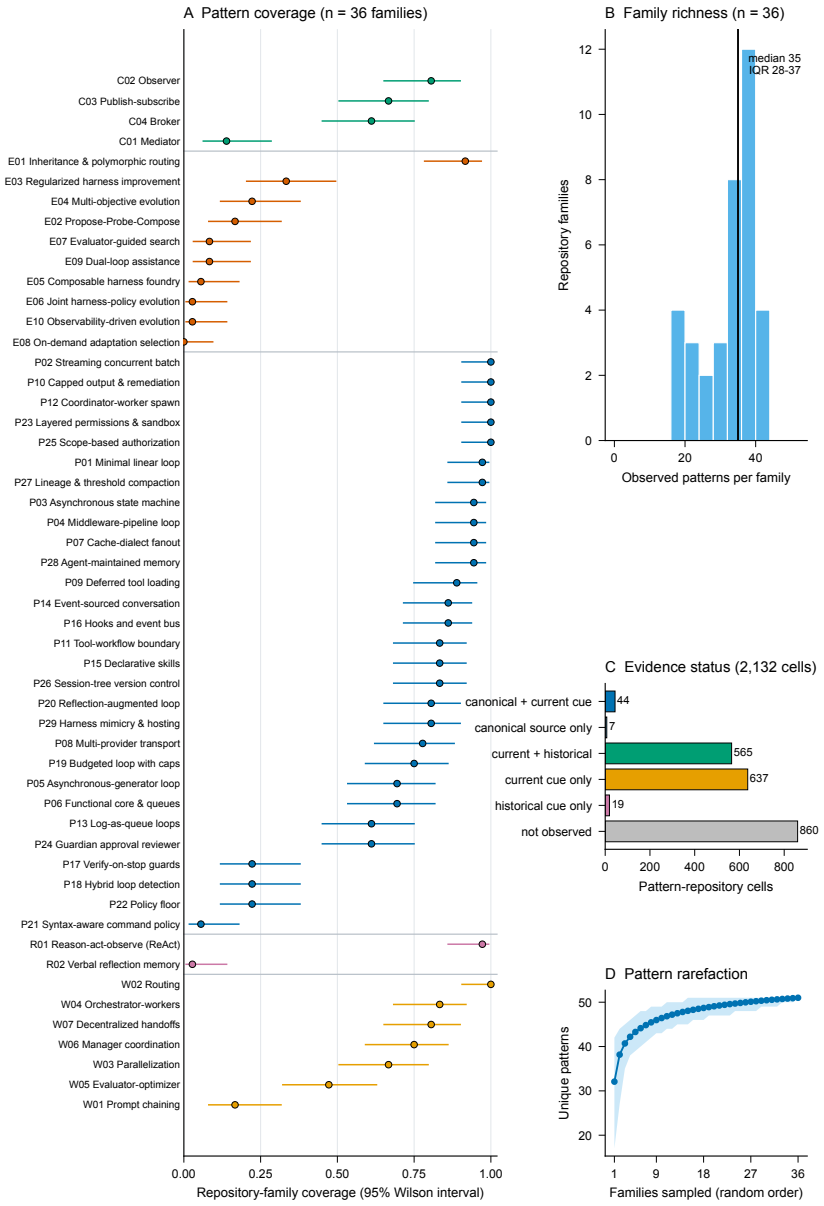


Figure 6 Coverage and diversity in the selected panel. (A) Family-level coverage for all 52 candidates with 95% Wilson intervals; colour denotes record family. Horizontal rules separate pattern families. (B) Distribution of observed-pattern richness across 36 families; the vertical line marks the median. (C) Evidence-layer composition across the complete 2,132-cell matrix. (D) Mean and 95% envelope from 5,000 random family orders. The unit is the repository family. Unobserved cells remain explicit and are not classified as absent mechanisms.

near miss stayed unmatched, and every frozen output hash was unchanged. That candidate is not applied to the frozen scan. Fixture checks are non-generalizing, and independent human adjudication of both the fixtures and the 16 coded candidates remains open, so these limits bound every observation in this section.

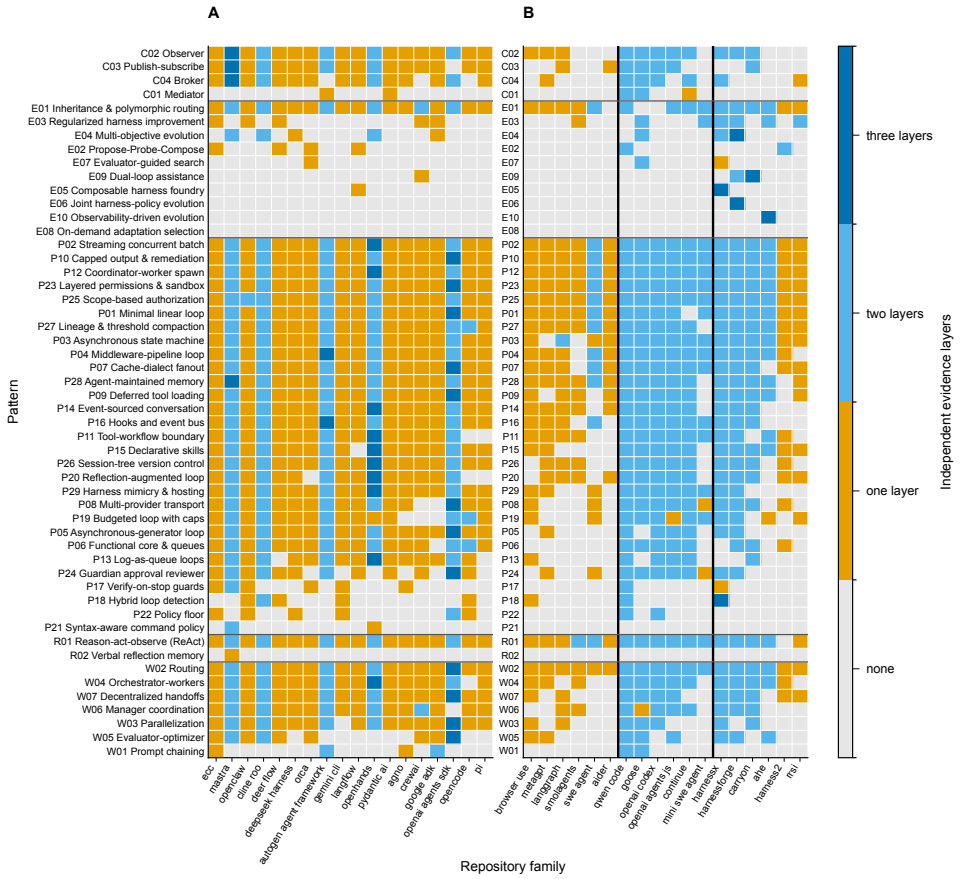


Figure 7 Pattern coverage by repository family and evidence layer. The 52 candidate rows are shared across two panels that continue through the 36 repository-family columns. Cell shade records the number of independent evidence layers available in a family: current cue, historical cue, and canonical source locator. Thin white rules separate cells; horizontal black rules separate pattern families, and vertical black rules separate the frozen, research, and implementation-supplement repository pools. Complete pattern names and values are provided in Appendix A.2.

3.4 The A^2 Taxonomy of Harness Design Patterns

The inventory says which patterns exist, and the audit says where their cues appear. Neither says what a pattern protects, or when that protection is defined, tested, and revised. The A^2 (Activities $\times$ Assets) taxonomy supplies both coordinates (Figure 1). It crosses the two streams of Section 2.2. The activities axis indexes *when* an obligation is defined, tested, and reconsidered. The assets axis states *what* must remain stable and what may change. Its foundations are classical patterns, modular decomposition, and test seams [16, 8, 37, 15]. A^2 is a proposed coordinate system for organizing records, and Section 6.5 states how its value as a lens can be tested.

Table 10 Family cue coverage by activity coordinate. A record is counted under every activity in its inventory coordinate, so the three activity rows overlap. Mean coverage is the mean share of the 36 repository families with a retrieval cue; the median is in families. The values are re-tabulated from the coverage records in Appendix A.2.

Activity coordinate	Records	Mean coverage	Median families
A1 Design and implementation	31	74.6%	30
A2 Testing and verification	16	44.1%	10
A3 Maintenance and evolution	22	48.2%	19.5
All records	52	61.8%	27.5

Table 11 Observed transitions after a candidate trajectory's first checkpoint. Probabilities are conditional on `first_observed`. Intervals use 2,000 repository-cluster bootstrap resamples.

Next state	Count	Probability [95% CI]
Large path change	113	0.352 [0.233, 0.473]
Moved / renamed	83	0.259 [0.140, 0.418]
Persisted	72	0.224 [0.058, 0.400]
Expanded	28	0.087 [0.045, 0.132]
Contracted	23	0.072 [0.024, 0.123]
Signal gap	2	0.006 [0.000, 0.015]

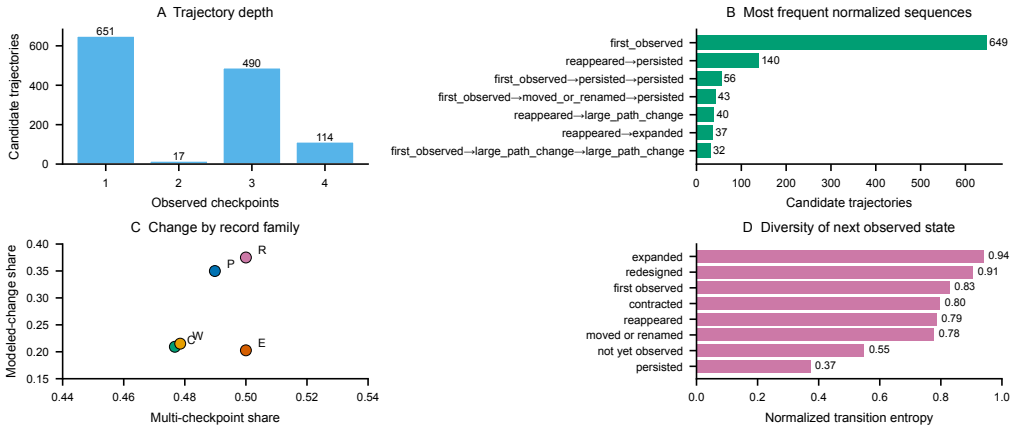


Figure 8 Candidate lifecycle trajectory structure. (A) Number of observed checkpoints per candidate–repository trajectory. (B) Most frequent normalized state sequences after removing leading `not_yet_observed` states. (C) Multi-checkpoint share and modeled-change share by record family. (D) Normalized Shannon entropy of the next observed state; higher values indicate more diverse observed transitions. A missing cue is a detector gap, not evidence of retirement. Sequence counts are descriptive for this checkpoint panel and are not probabilities for unseen repositories.

3.4.1 The Activities Axis

The axis has three activities, taken from the activities stream: design and implementation (A1), testing and verification (A2), and maintenance and evolution (A3). A1 declares authority, state ownership, boundaries, and anticipated change points. A2 exercises those declarations under malformed results, partial failure, adversarial instructions, and task variation. A3 uses operational or repeated-run evidence to decide whether the existing seam

still absorbs required change. If it does not, the A3-to-A1 return is *re-generalization*—redesign of the contract, a step beyond routine tuning. The axis is an evidence cycle that any record can re-enter.

3.4.2 The Assets Axis

The axis has two assets, taken from the assets stream. The *protected contract* is the behavior that clients, operators, and auditors may rely on: an interface together with its authority, state, and acceptance semantics. The *permitted variation* is the set of changes that preserve that contract, reached through a named variation mechanism such as an adapter, hook, policy profile, or router. Two probes locate the boundary between them. A *substitution probe* changes an adapter, routing policy, evaluator rubric, or context projection while holding the declared invocation, authority, state, and acceptance semantics fixed. If the replacement remains admissible, it lies inside the variation seam. A *violation probe* asks which change would invalidate the design: altering an error meaning, lowering a permission floor, discarding required provenance, or letting a component approve its own output usually changes the protected contract. The contract is therefore a semantic commitment, often broader than an API, and permitted variation is the configuration that preserves it.

The boundary is also setting-dependent. A sandbox provider may vary while the execution-isolation guarantee remains fixed; an evaluator rubric may vary while independent acceptance remains fixed; a memory projection may vary while event identity and ordering remain fixed. In another setting, one of those mechanisms may itself be protected. A record must therefore state the actor allowed to vary the asset, the evidence used to check the invariant, and the response to a violation. Ambiguity is resolved by competing contract hypotheses and tests. Paradigm, quality, and human/agent factors qualify these judgments without adding new record fields [14].

3.4.3 From Coordinates to Pattern Records

Pattern discipline governs how a recurring mechanism becomes revisable engineering knowledge. It extends cataloging with a decision: whether a recurring mechanism should be reused, revised, or retired. Five obligations define a pattern record. *Boundary:* state the context, trigger, structure, effect, and a negative case outside the pointcut. *Contract:* name authority, precondition, invariant, postcondition, and violation behavior. *Variation:* separate the frozen point from the hot spot and identify their connecting mechanism [16, 8, 15]. *Evidence:* distinguish retrieval cues, source instances, pointcut matches, contract tests, and measured outcomes. *Lifecycle:* version the record and state retest, migration, and retirement conditions. A candidate is promoted once every obligation is verified. The records are catalog-agnostic: they let a familiar label, repeated token, or single implementation carry exactly the evidence it has. Table 12 contrasts a record with a mechanism description.

Three properties of this schema bound what we claim for it. The five obligations are one candidate field set. The normative minimum of Section 1 holds whichever field set a community adopts, and the value of this one is the subject of Hypothesis 1. A record also has a cost; whether its benefit repays that cost is part of what the record-utility study measures. Finally, a record documents an obligation and does not enforce it. Authorization and auditability need runtime mechanisms, which Hypothesis 2 tests separately.

The distinction applies throughout the harness lifecycle. A mechanism list answers "what components or behaviors have been seen?" Pattern discipline adds "in what context does it apply, which contract does it protect, what may vary, who has authority, what evidence

Table 12 What a disciplined pattern record adds to a mechanism description. The example context is replacing a tool provider or adapter.

Question	Mechanism description	Disciplined pattern record
When does it apply?	Names an adapter, authorization check, or confinement mechanism.	States the recurring problem, context, assumptions, and provenance.
What remains, and what may vary?	Describes an implementation or interface.	Separates the protected contract from the permitted provider, schema, or adapter variation.
How should the change be tested?	May supply local examples or component tests.	Names authority boundaries, consequences, and tests for bypass, interference, or invalid combinations.
What happens after later updates?	Leaves lifecycle decisions to the implementer.	States re-evaluation evidence and keep–adapt–retire conditions across A1–A3.

checks it, and when should it be retested or retired?" The first supports retrieval; the second supports design comparison, verification, migration, and maintenance.

A concrete example makes the comparison explicit. Suppose a team replaces a tool provider or adapter. A mechanism list can identify adapters, authorization, and isolation, but it need not state the contexts in which each applies or the obligations that survive replacement. Table 12 states what a disciplined pattern record would add. The comparison is an illustration evaluated by the falsification tests in Section 6, not an established empirical advantage.

3.5 Using the Taxonomy: Mapping, Detection, Tracking, and Analysis

The taxonomy supports four uses. It *maps* a record to activity and asset coordinates. It tells a detector what to look for when it *detects* the record in a repository. It gives a history something to follow when it *tracks* the record across versions. And it supports the *analysis* of relations among records. We illustrate each use with case studies drawn from the inventory and the audit.

3.5.1 Mapping Records to Coordinates: Nine Analytical Cases

Appendix A assigns every record an activity coordinate, a proposed protected contract, and a proposed permitted variation. Section 3.3.1 re-tabulates the exploratory cue coverage by those coordinates.

Table 13 selects nine analytical cases spanning 22 inventory records across design, verification, and evolution (Figure 9). They were selected to expose alternatives, dependencies, complementary obligations, and conflicts. Each case separates three evidence statuses: the cited source reports a mechanism; the protected contract and consequence are our interpretation; and the discriminating test is a proposal. Section 6.5 specifies how independent analysts could test the reliability and coverage of those annotations.

1. Delegation is not handoff. Anthropic's orchestrator–workers workflow delegates dynamically generated subtasks [2], whereas OpenAI's handoff guidance transfers control to a specialized agent [35]. Both customize who performs work, but their frozen contracts differ: a worker returns a result to a coordinator; a handoff must specify what state and

Table 13 Nine analytical cases located with the same A^2 grammar. The case names and cited mechanisms are source-reported; the contract, variation, and consequence columns are our proposed interpretation, and Section 3.5.1 gives the corresponding discriminating tests. The cases are analytical illustrations selected for contrast. Complete records, sources, and provenance qualifications are in the online inventory (`https://xhyccc.github.io/pohe/inventory.html`) and Appendix A.3.

Case / activity	Assets: proposed interpretation		Proposed consequence
	Protected contract	Permitted variation	
1. Delegation vs. handoff *A1 · Design*	Task/result envelope; explicit control ownership.	Assign workers by dispatch; transfer roles by handoff.	Both distribute work, but transfer different authority and context.
2. Tool extension *A1 · Design*	Invocation/result schema; error semantics.	Swap adapters; intercept middleware; load tool registries lazily.	Compatibility need not preserve order, permissions, or side effects.
3. Message topology *A1/A3 · Design and evolution* [39]	Message schema; delivery semantics; participant authority.	Mediate or broker messages; register observers or topics.	Decoupling creates ordering, trust, and subscription obligations.
4. Permission layers *A2 · Verification*	Minimum authorization; execution-isolation boundary.	Compose scoped profiles; swap sandbox adapters.	A correct policy decision does not prove execution confinement.
5. Feedback vs. acceptance *A2 · Verification*	Feedback interface; protected acceptance/exit rule.	Vary rubrics in evaluator loops; vary checks at exit hooks.	Improvement is not authorization; both may share faulty checks.
6. Durable state *A1/A3 · Design and evolution*	Event identity; ordering; branch provenance.	Swap context projectors and recovery policies through event/branch APIs.	History enables inspection, not reversal of external effects.
7. Context revision *A2/A3 · Verification and evolution* [42]	Retained-evidence contract; memory-update contract.	Change projections by compaction; add feedback memories by reflection.	Both alter later observations and may lose evidence or retain bad advice.
8. OO-Harness *A1/A3 · Design and evolution* [25]	Base interfaces; specified inherited behavior.	Override by inheritance; specialize tasks by router dispatch.	A narrow edit seam may hide parent coupling; dispatch does not enforce invariants.
9. Evolution constraints *A2/A3 · Verification and evolution* [51, 57]	Evaluation access; objectives; acceptance protocol.	Bound edits and compose candidates within a controlled search space.	Local gains may overfit selection feedback or hide objective trade-offs.

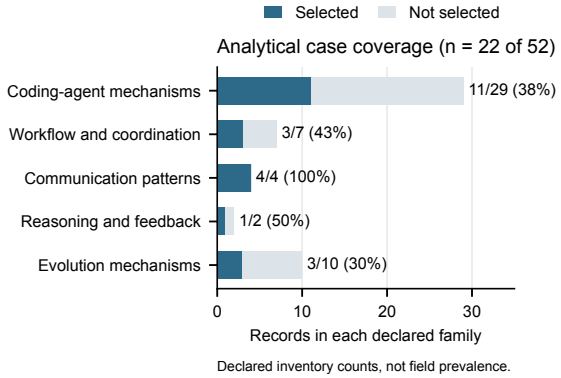


Figure 9 Analytical coverage of the declared inventory. Dark segments show records selected for the nine analytical cases; light segments show the remaining records in each source family. The cases include 22 of 52 records and all five families, but selection is uneven and argument-driven. Counts describe this paper's declared inventory, not field prevalence or pattern effectiveness.

authority move with control. A2 therefore needs different probes for missing worker results and circular or overprivileged handoffs. The organizational drawing alone does not capture this distinction.

2. An extension interface does not fix execution semantics. Consider the middleware, provider-adapter, and deferred-loading entries P04/P08/P09 [6]. An adapter translates an invocation; an interceptor can also reorder, reject, or repeat it; a lazy registry changes when a capability becomes visible. All may preserve a nominal tool schema while changing observable behavior. Their A1 commonality is an extension seam, not interchangeability. A2 should test error translation, retry side effects, and authorization after loading. Retiring a wrapper after a model upgrade must preserve its external obligations, not just its score.

3. Communication patterns supply contracts, not only labels. The MCP survey covers Mediator, Observer, publish–subscribe, and Broker in agent communication [39]. Our reading separates the participant/message contract from routing and subscription policies. Replacing a mediator with a topic bus changes who receives evidence, who can invoke whom, and where ordering is enforced; it is not merely renaming coordination. A3 also matters: stale observers, schema changes, and revoked participants can invalidate an A1 contract. This case acknowledges classical pattern reuse while exposing lifecycle questions that a communication topology alone leaves unanswered.

4. Authorization and confinement are separate seams. The policy-floor, sandbox, and scoped-profile mechanisms provide an illustrative case [6]. A task-specific profile may tighten a protected authorization floor, but a correct approval decision says little about an execution adapter that permits additional effects. Conversely, an isolated environment does not decide whether an action is authorized. The A^2 analysis requires both contracts and tests the seam between them: scope inheritance, revocation, and enforcement after tool substitution. Policy-floor redesign is privileged re-generalization, not ordinary agent customization.

5. Feedback is not acceptance authority. Evaluator–optimizers give revision feedback [2]; a verify-on-stop design intercepts termination before accepting the artifact. These functions can coexist, but neither a persuasive critique nor a passing checker proves that the acceptance path is protected. The intended frozen point is the acceptance protocol, while project checks are hot spots reached through an exit hook. A2 must test bypasses, solver edits to checks, flaky failures, and budget exhaustion. Changing a rubric and changing who may approve

that rubric are different operations.

6. Durable history is not reversible execution. The log-as-queue, event-sourcing, and session-tree entries P13/P14/P26 connect A1 execution to A3 recovery [6]. Stable identities and ancestry can support multiple context projections and branch policies, but a replayed tool call may repeat an external action. A maintenance change that prunes history can also remove evidence required by A2. The seam therefore needs an explicit retained-evidence and side-effect contract; an append-only log or a rollback command is not sufficient. Recovery tests should inspect both restored local state and duplicated external effects.

7. Compaction and reflection alter evidence differently. P27 compaction changes what history is projected into context [6]; Reflexion uses verbal feedback across attempts [42]. One can discard a necessary fact, while the other can preserve mistaken advice. Their shared activity is context revision, not an identical memory mechanism. The A1 interface should state provenance and update authority; A2 should test retained facts and contaminated feedback; A3 should specify when a summary or reflection is revised or removed. A token-saving policy is not successful if it invalidates a later check.

8. Inheritance localizes edits but does not freeze behavior. OO-Harness supplies a concrete inheritance-based mechanism and polymorphic task routing [25]. A base interface can make permitted edits visible, but overrides may still violate assumptions elsewhere in the hierarchy. Our proposed frozen point is therefore the specified inherited contract, not the mere existence of a parent class [16]. The case motivates regression tests across subclasses and routes, and a comparison with composition under matched changes. Section 5.1 examines how agent-driven editing closes the A3-to-A1 loop within this space.

9. The mutation seam and the acceptance rule are different assets. RRSI regularizes repeated harness improvement; Harmony considers multi-objective evolution and composition [51, 57]. These are not a head-to-head mechanism comparison. Together they motivate separating what a candidate may edit from what evidence admits it. Edit budgets and composition policies can vary within an experiment, while evaluation access and declared objectives must remain controlled. Repeated selection feedback cannot double as an untouched final test. An A3 gain that changes its own acceptance criterion does not validate the A1 generalization.

Across the cases, one question recurs: what change can the asset absorb, and what evidence would force contract renegotiation? In every case the consequence column names the failure that would be hidden by a catalog entry that records only the mechanism. Each case thus contributes a contract and a proposed rejection test, and the protocols in Section 6 turn the cases into comparative experiments.

The cases establish expressiveness: the taxonomy can state distinctions that a mechanism name leaves implicit. They are argued from the cited sources and from the semantics of the mechanisms. Whether independent analysts assign the same coordinates, and whether A^2 covers more or costs less than an existing scheme, are empirical questions with a stated test (Section 6.5).

3.5.2 Detecting a Pattern in a Repository: Pointcuts and Contracts

A coordinate becomes detectable once it is written as a specification. The *pointcut* states context, trigger, structure, and effect: where in execution the pattern applies and what it changes. The *contract* states what the mechanism requires and ensures, the invariant it protects, and its behavior on violation. A *negative control* names a look-alike that lies outside the pointcut. Table 14 gives the specifications for three verification records.

The specifications make the model-coded dispositions of the exploratory audit readable

Table 14 Pointcut and contract specifications for three verification records. The pointcut (context, trigger, structure, effect) states where a record applies and what it changes; the contract (requires, ensures, invariant, violation) states what it protects; the negative control names a look-alike outside the pointcut. The text is taken from the study's frozen pattern specification, which holds one such entry for each of the 52 records.

Field	P17 Verify-on-stop guards	P18 Hybrid loop detection	P21 Syntax-aware command permissioning
Pointcut			
Context	The agent proposes completion after producing an artifact.	Successive agent observations or actions can repeat without progress.	An agent proposes a shell command with security-relevant structure.
Trigger	A stop or finish action is requested.	A loop signal crosses a configured threshold or composite condition.	The command is about to be parsed, classified, or executed.
Structure	Exit interception, verifier execution, and accept-or-continue result.	Repetition signals, thresholds, and an intervention interface.	Shell parser, normalized command tree, policy evaluator, and execution handoff.
Effect	The stop is accepted only after the verifier reports sufficient evidence.	The harness diagnoses a loop and stops, redirects, or escalates.	Authorization uses parsed operations and a parse failure cannot fall through to execution.
Contract			
Requires	Stop can be intercepted and at least one verification is available.	Action or observation history is available in comparable form.	The command is representable by a parser or rejected before execution.
Ensures	Failed verification returns actionable feedback or blocks completion.	Intervention is attributed to explicit evidence and recorded in the trace.	Every subcommand and redirection is covered by a policy decision.
Invariant	The same acceptance gate cannot be bypassed by another stop path.	A detector cannot block all exploration solely because outputs are similar.	Parsed intent and executed command refer to the same normalized operation.
Violation	Unavailable or flaky verifier yields explicit unknown or failure, not silent success.	Detector error, disabled mode, or false positive has a recoverable path.	Unsupported syntax fails closed or requests explicit review.
Negative control	Tests run by the user after the run are not verify-on-stop.	A recursion depth limit alone is not semantic loop detection.	A substring denylist is not syntax-aware permissioning.

(Section 3.3.2; per-cell votes in Table 24). The dispositions are one model's coding under 200 prompt profiles, so we use them as worked examples of how a specification is applied, not as verified findings about the repositories. For syntax-aware command permissioning (P21), the matcher selected seven repositories. Three gained majority support (Qwen Code, OpenCode, and OpenClaw), where parsed commands feed a permission decision. Three were rejected under every profile (DeerFlow, Codex, and Orca), because the declared effect or boundary was absent. Gemini CLI stayed unresolved: parsed commands feed its confirmation policy, but a coarse fallback remains disputed. For hybrid loop detection (P18), Pydantic AI and Roo Code were supported under every profile and Mastra was rejected. For verify-on-stop (P17), the one selected cell, DeerFlow, stayed unresolved. A post-completion acceptance verdict exists, but checker exceptions do not change the completed status and no enforced stop gate is shown. The negative control addresses exactly this case: a check that runs after completion is feedback, and only an intercepted stop is acceptance. Across all 16 cells, the profiles agreed on every context and trigger judgment and varied on four effect judgments. In these cells and under these prompts, variation therefore concentrates at the effect, which is where the contract begins. Whether effect is the hard step in general is a question for independently adjudicated samples.

A matched pointcut is still short of a complete record. The Verify-on-Stop record below states what a detected instance must supply; it instantiates Case 5. Figure 10 draws the same record as an analytical UML class view: who proposes completion, who verifies, and which gate alone may accept. Appendix A.4 draws three further class views, and the online inventory gives every complete record. Each view reconstructs a proposed contract, not an execution observed in a repository.

An A2 record: Verify-on-Stop.

Context and problem. A tool-using agent can claim completion without checking the resulting artifact. A guard should prevent premature acceptance while making failures actionable. *Frozen point and hot spot.* The termination interception and evaluator access boundary are protected contracts. Project-specific tests, check selection, and diagnostic formatting are customizable. Test changes have a separate approval process from ordinary solver changes.

Variation mechanism. A controlled exit hook runs the checker and returns a structured verdict. A failure can reopen the rollout with diagnostics, subject to retry and resource bounds. Budget exhaustion or an unavailable evaluator must be distinguishable from acceptance.

Consequences. Verification adds latency and can reject correct work when tests are flaky or incomplete. Model-based checks can be correlated with the solver. The guard does not protect a writable test suite merely because it is outside the solver's prompt.

Evidence and retirement. Compare otherwise matched runs with and without the hook, recording incorrect acceptances, incorrect rejections, cost, and escalation. A faster or stronger model is not sufficient reason to remove enforcement; removal must preserve the acceptance contract through another evaluated mechanism.

3.5.3 Tracking a Pattern across Versions

The A3 activity asks whether a record persists, moves, or is replaced. Tracking follows one record–repository pair across frozen releases and labels each checkpoint with a detector-derived state. Figure 11 shows six illustrative pattern–repository trajectories on a calendar-time axis. The selected 24 checkpoints span stable continuity, cue-intensity and path changes, cross-repository lineage persistence, name or location movement, and detector gap recovery.

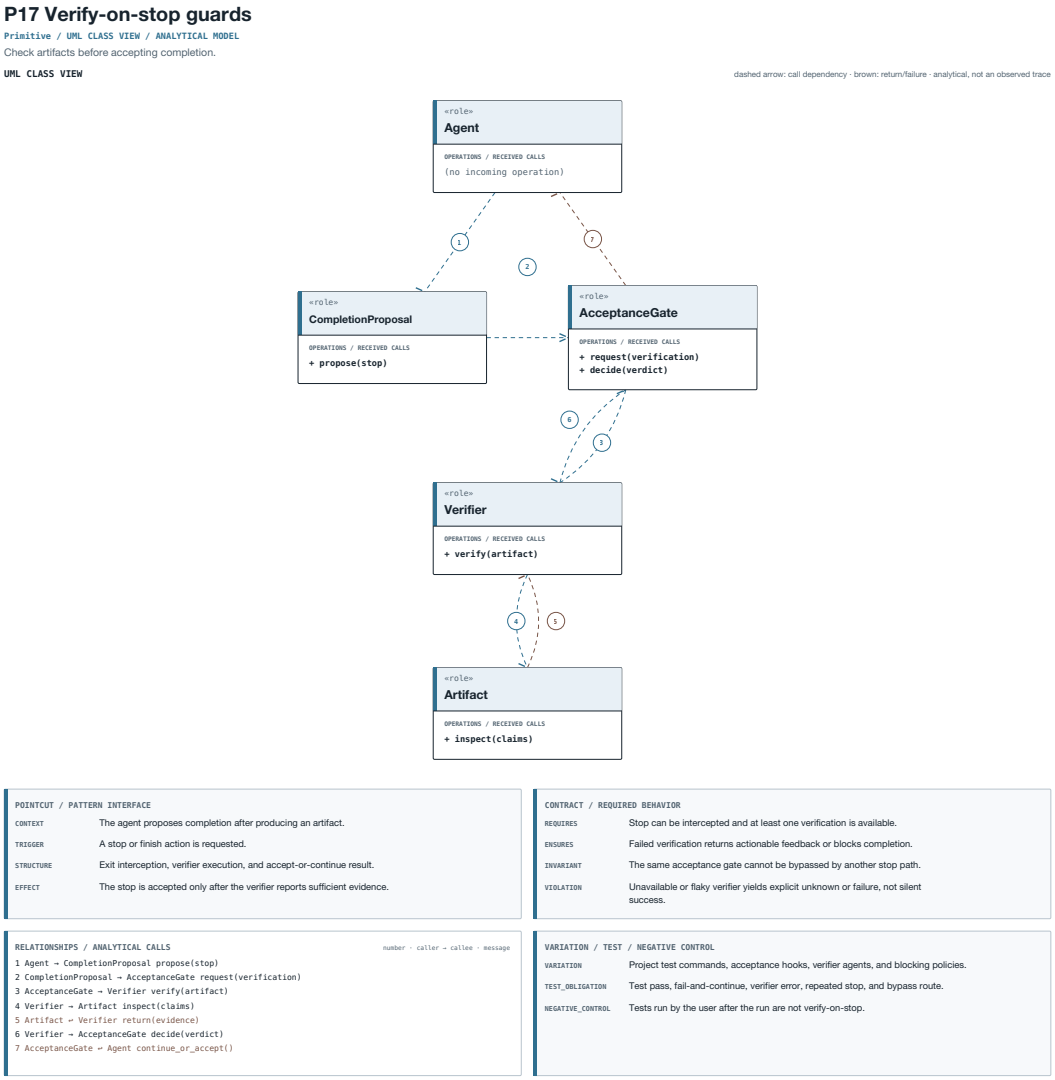


Figure 10 The Verify-on-Stop record (P17) as an analytical UML class view. Each class box is a proposed participant role; its operation compartment lists the calls that role receives. Numbered dashed arrows are call dependencies, and brown arrows are returns or failures. The relationship panel restates every call; the lower panels restate the record's pointcut, contract, variation, test obligation, and negative control from its frozen specification. The view is an analytical reconstruction of the proposed contract, not an observed execution trace or a repository class declaration. The invariant is the point of the record: the acceptance gate cannot be bypassed by another stop path.

Lines connect sampled observations only; they do not interpolate unobserved change or establish retirement.

The six trajectories cover four situations. *Persistence across a lineage:* cues for the minimal linear loop (P01) and the reason–act loop (R01) appear in SWE-agent from v0.1.0 (April 2024) and persist into its simplified successor mini-SWE-agent through v2.4.6 (July 2026); cues for the coordinator–worker spawn flow (P12) persist across the migration from AutoGen to Agent Framework. *Reorganization within a repository:* cues for layered permission and sandbox (P23) in OpenHands show large cue-path changes at both later checkpoints

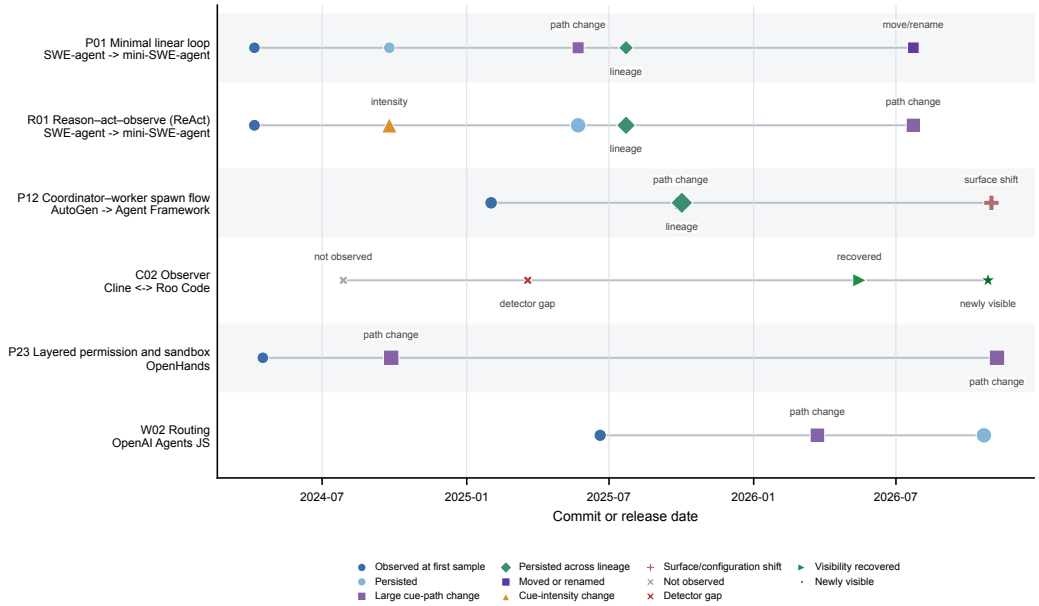


Figure 11 Representative calendar-time pattern trajectories. Each row is a pattern–repository trajectory. The horizontal axis gives the frozen commit or release date, not a normalized lineage time. Marker shape and colour encode the detector-derived lifecycle state, and marker area indicates the number of detector cues at that checkpoint. The selection covers six trajectories and 24 checkpoints with stable, changing, lineage-crossing, and gap/recovery states; it is illustrative rather than a probability sample. A detector gap is not interpreted as retirement, and the lines do not establish source-level architectural redesign.

between 0.3.0 and v1.37.0, and routing (W02) in OpenAI Agents JS changes paths once and then persists. *Detector gap and recovery:* Observer (C02) has no cue at the two earliest Cline and Roo Code checkpoints and becomes visible at the later ones, which is recorded as a gap and not as absence.

A cue path that moves shows that code was reorganized. It does not show whether the contract survived the move. Each specification therefore carries an evolution question that a source review must answer. For P23 it asks whether one policy surface evolved into layered cross-platform enforcement; for P12, whether delegation gained nested history, approvals, and a background lifecycle; for P17, whether completion verification became mandatory instead of advisory. Answering these questions requires commit-level adjudication, which the present audit leaves to the next study (Section 8.4). The Session-Tree Recovery record states what such tracking must preserve; it instantiates Case 6.

An A3 record: Session-Tree Recovery.

Context and problem. Long tasks can require revisiting a decision without discarding the entire execution history.

Frozen point and hot spot. Stable identifiers, branch relationships, and provenance form the record contract. Branch selection, checkpoints, and summarization are variable policies. Their interface must state what happens to pending actions and external side effects.

Variation mechanism. A branch/checkpoint API separates task-specific recovery from the representation of durable state.

Consequences. Branching retains more data and may expose sensitive history. A summary

can omit a fact needed by a later verifier. A rollback of the local record cannot undo an irreversible external action.

Evidence and retirement. Inject interruptions at defined points and check recovery, duplicate side effects, and provenance retention. State migration is privileged re-generalization if it breaks the old contract, not a routine pruning policy.

3.5.4 Analyzing Relations among Patterns

Relations among mechanisms are optional metadata that extend the contribution. This subsection provides a notation for machine-readable catalogs, graph representations, or annotation studies; validating the proposed types for completeness and consistent annotation is future work.

Let m_i and m_j be mechanism records. We write

$$m_i \xrightarrow[\phi]{\rho} m_j,$$

where ρ names the connection and ϕ states the context or precondition under which it is claimed. No edge is context-free. The provisional connection types are:

- *requires:* m_i's stated contract is valid only if m_j supplies a named obligation;
- *complements:* the records address distinct obligations of the same problem, and neither is presented as sufficient;
- *specializes:* m_i narrows m_j's context, invariants, or permitted variation without contradicting them;
- *alternates with:* the records address the same problem but make different trade-offs or authority assignments; and
- *conflicts with:* their stated assumptions or invariants cannot hold together in the named context.

Activity order supplies a separate lifecycle connection: a pattern can be introduced in A1, probed in A2, adapted or retired in A3, and returned to A1 when its protected contract must change. These annotations are proposals to document and test, not universal laws or observed causal effects.

As a worked fragment, a tool adapter requires an authorization boundary because schema compatibility does not grant permission. A sandbox complements that boundary because approval and execution confinement address different obligations. A workspace profile specializes the authorization policy; delegation alternates with handoff because it retains rather than transfers control; and an evaluator–optimizer conflicts with protected acceptance when it can rewrite the evidence that authorizes termination. Adding a sandbox therefore does not repair a missing authorization decision.

The nine cases in Table 13 can be encoded as follows; the inventory IDs retain their meanings from the online inventory and Appendix A.2.

1. Delegation → handoff:
   $\rho =$*alternates with*, $\phi =$*control owner* (A1: W04/W07).
2. Deferred loading → tool extension:
   $\rho =$*specializes*, $\phi =$*on-demand visibility* (A1: P04/P08/P09).
3. Mediation → publish–subscribe:
   $\rho =$*alternates with*, $\phi =$*routing topology* (A1/A3: C01–C04).
4. Sandbox → policy floor:
   $\rho =$*complements*, $\phi =$*decision vs. confinement*
   (A2: P22/P23/P25).

5. Evaluator–optimizer → verify-on-stop:
   $\rho =$*conflicts with*, $\phi =$*reviewer can edit checks* (A2: W05/P17).
6. Session branching → event history:
   $\rho =$*requires*, $\phi =$*recoverable ancestry* (A1/A3: P13/P14/P26).
7. Compaction → reflection memory:
   $\rho =$*alternates with* or *complements*, $\phi =$*context-revision method* (A2/A3: P27/R02).
8. Task subclass → base harness:
   $\rho =$*specializes*, $\phi =$*inherited contract preserved* (A1/A3: E01).
9. Regularized search → multi-objective evolution:
   $\rho =$*complements*, $\phi =$*different risks* (A2/A3: E03/E04).

4 Three Major Hypotheses

The inventory and the A^2 taxonomy lead to three hypotheses about what makes a harness durable under repeated change. They concern, respectively, the asset that should survive revision, the authority that must constrain an agent maintainer, and the conditions under which self-modification can be trusted. These hypotheses turn the position into an empirical program without treating the responsibility records or the source audit as causal evidence. We develop each hypothesis from a proposed mechanism to observable consequences, and then consider what the present evidence can and cannot establish. Section 6 translates these arguments into prospective protocols.

4.1 H1: Inspectable Commitments Are the Durable Asset

Rapid revision makes prompts and implementations unstable reference points. H1 therefore locates the durable engineering asset in the design commitment that successive realizations are expected to preserve. We use *commitment* to denote the conjunction of a pointcut, a contract, permitted variation, a test obligation, and a negative control. A prompt or implementation can realize such a commitment, but neither is synonymous with it. The hypothesis is not that one record schema is final; it is that an explicit and checkable commitment will survive realization changes better than an unnamed or merely documented one.

The proposed mechanism is a common criterion of preservation across revisions. A pointcut identifies where a mechanism must act, while the contract states the obligation imposed there. The variation boundary distinguishes routine substitution from redesign, and the tests and negative control distinguish preservation from both loss of function and superficial resemblance. In the A^2 framework, these elements make the Assets axis available to repeated review and verification. They also expose a distinction that component names conceal: replacing an implementation need not replace its obligation, whereas changing the obligation constitutes redesign even if the implementation looks unchanged.

This account can be evaluated by following predeclared revisions of harness–record pairs. Independent reviewers or executable checks assess whether each required obligation remains intact after a change episode. For k obligations, we define contract retention as

$$R = \frac{1}{k}\sum_{j=1}^{k} \mathbf{1}(o_j \text{ is preserved}), \qquad D = 1 - R,$$

where D measures contract drift. Retention and drift concern semantic preservation rather than textual similarity. An explicitly authorized contract redesign is reported separately,

rather than silently counted as retention. The test obligation must also continue to exercise the revised realization, and the negative control must remain outside the pointcut; otherwise an apparently stable contract may reflect a weakened measurement instrument.

The 52 records, nine analytical cases, and exploratory source audit establish the feasibility of specifying such commitments and using them to distinguish retrieval from structural and contract evidence. They do not establish that the commitments improve engineering outcomes. That question requires the matched comparison of complete records, mechanism-only descriptions, and ordinary documentation in Section 6.1. Within that scope, H1 would be rejected if complete records failed to exceed the prespecified retention margin, failed to reduce drift, or produced an advantage too small to meet the declared practical threshold.

4.2 H2: Agent-Led Evolution Requires Mechanized Separation of Concerns

H2 concerns control rather than representation. When an agent participates in evolving its own harness, we hypothesize that separation of concerns must be enforced in the runtime. Guards, acceptance gates, checkpoints, rollback, and independently controlled evaluation divide what the agent may propose from what it may authorize. Documentation, comments, and prompt instructions can explain this division, but they do not themselves impose it.

The argument begins with an asymmetry between human and agent maintenance. A human maintainer can be held accountable through an external review process; an agent maintainer requires an analogous boundary in the execution path. The proposer should not be able to rewrite the permission floor, admit its own candidate, erase the evidence required for recovery, or replace the final evaluator. The Activities axis of A^2 distinguishes design, verification, and evolution; H2 adds that agent-led evolution also requires distinct control rights over those activities. The E-family records offer candidate mechanisms for such a boundary, ranging from bounded adaptation to regularized and multi-objective search, but their presence in the inventory says nothing about their comparative effectiveness.

The hypothesis yields observable consequences beyond the presence of a named gate. Relevant measurements include unauthorized protected-contract changes, attempted and successful bypasses, the proportion of protected actions covered by an enforced gate, recovery after induced failure, and the provenance linking a promoted change to its proposer, authority, and evaluation evidence. These measurements require capability-matched controls; otherwise a larger runtime could receive credit merely for possessing more tools or a larger budget.

Current support for H2 is analytical and source-located. Existing systems show that candidate enforcement and evolution mechanisms can be implemented, while the audit demonstrates why a mechanism name cannot establish its authority boundary. Neither observation shows that external enforcement improves on a minimal runtime guided by instructions. Section 6.2 therefore compares instruction-only, capability-matched minimal, and externally enforced runtimes. H2 would be rejected within the tested scope if the minimal runtime were non-inferior on both contract-preserving success and unauthorized effects under the preregistered margins. A record that merely describes a gate would not satisfy the proposed mechanism.

4.3 H3: Explicit Variation Boundaries Improve Self-Modification

H3 joins the assets of H1 to the control separation of H2. It predicts that, relative to unconstrained self-modification, agent-led revision within explicit variation boundaries will produce less contract drift, fewer regressions, faster recovery from failed changes, and more

complete change attribution. The expected benefit arises from restricting routine search to permitted variation while reserving protected-contract changes for a separate, authorized redesign path.

Under this account, a bounded proposer can explore adapters, policies, prompts, or routing choices without silently rewriting the obligations those choices are meant to serve. An acceptance gate evaluates candidates against the same tests and negative controls, while checkpoints and rollback make unsuccessful candidates reversible. Changes that cannot be absorbed by the declared seam follow the A3-to-A1 return in A^2. H3 is thus the paper's central hypothesis about closed-loop evolution, rather than a conclusion inferred from existing E-family systems.

The prediction can be examined over successive revision rounds. Contract drift uses the H1 measure at each round. Regression rate is the proportion of previously passing held-out checks that fail after an accepted modification. Recovery time comprises the attempts and elapsed time between detecting a failed modification and restoring a state that satisfies the protected checks, with unrecovered episodes censored at the budget. Attribution completeness is the proportion of accepted changes linked to a proposer, diff, authority, evaluation result, and rollback point. Contract drift and regression rate are co-primary outcomes; recovery and attribution remain secondary because faster or better-documented failure cannot compensate for a violated contract.

No part of the present audit compares bounded and unconstrained self-modification, so H3 remains causal-open. Its current motivation is limited to the conceptual distinction between variation and redesign, the availability of candidate E-family mechanisms, and descriptive evidence that cue locations change across releases. The two-arm, multi-round protocol in Section 6.3 supplies the necessary intervention. H3 would be rejected within scope if bounded modification failed to meet either co-primary improvement margin, or if it increased drift or regression beyond the corresponding margin. Inconclusive intervals would leave the hypothesis unresolved rather than support it.

Read together, the hypotheses separate three questions that are easily conflated: what a record preserves, who may change it, and whether that division improves repeated self-modification. The inventory and audit can locate and help specify the mechanisms under study. Longitudinal and controlled intervention studies must establish their claimed benefits.

5 Structural Drift: Software versus Harness Engineering

Harness engineering inherits the assets and activities of software engineering, but three changes complicate their transfer (Table 15). A harness can close the maintenance loop through the system itself, agents can consume and revise design patterns, and a mechanism's value can change with the underlying model. None of these changes is absolute: software engineering already includes automated maintenance and self-adaptive systems [49], and conventional design assets also age. We therefore treat them as shifts in the default setting rather than as a dichotomy between two fields. Each is an analytical proposition with distinct risks and empirical consequences, not an effect established by the exploratory audit.

5.1 Closed-Loop Evolution Changes the Maintenance Problem

In conventional software maintenance, developers propose changes and a separately administered process admits a release. A closed-loop harness can instead use its own traces to propose, probe, select, and reuse successor logic. OO-Harness evolves subclasses and routes tasks among them [25]; Harness^2 carries adaptation experience across iterations [53];

Table 15 Structural drift from software engineering to harness engineering. Each row names an assumption that software patterns make, how it shifts for harnesses, and what a responsibility record must add. The rows are proposed shifts of emphasis; each carries a rejection test in the text.

Dimension	Software engineering	Harness engineering	What the record must add
Evolution loop (Section 5.1)	Developers typically propose changes; a separately administered process admits a release.	The harness can use its own traces to propose, probe, select, and reuse successor logic.	An evolution envelope: variation searched within an epoch, the contract revised only across epochs under separate authority.
Pattern consumer (Section 5.2)	Human designers, testers, and maintainers read patterns and apply judgment.	Agents design, verify, and evolve the harness; humans hold the named authority.	Machine-checkable pointcuts and tests, stated authority, context visibility, and control ownership.
Utility lifetime (Section 5.3)	An asset's value changes mainly with requirements, dependencies, or implementations.	A mechanism may also gain or lose value when only the model changes.	A boundary between replaceable implementations and protected obligations, with a controlled removal test.

and RRSI and Harmony Harness constrain repeated search or expose functionality–security trade-offs [51, 57]. These systems demonstrate system-level adaptation, distinct from model training with AI feedback [4], but they do not establish safe autonomous maintenance in general.

Our position is that this loop should be bounded rather than abandoned. A harness is modified repeatedly as models, tools, and tasks change, and its traces provide direct evidence about where it fails. The central design question is consequently which parts an agent may edit and on what evidence a candidate may be admitted. A pattern record supplies one possible boundary by separating protected contracts from permitted variation. For closed-loop self-improvement, we specialize that record to an evolution envelope $\mathcal{D} = (C, \mathcal{V}, \mathcal{A}, \mathcal{E}, \mathcal{L})$, comprising the protected contract, permitted variation, authority, acceptance evidence, and lifecycle rule. A change is admissible only inside $\mathcal{V}$, under $\mathcal{A}$, while preserving C and satisfying $\mathcal{E}$; otherwise it returns to A1 for contract redesign.

Protection within an epoch does not imply immutability. During a *variation epoch*, the loop proposes and admits changes inside $\mathcal{V}$ while C, $\mathcal{A}$, and $\mathcal{E}$ remain fixed. Accumulated evidence may eventually show that the declared variation cannot absorb a required change. At that *epoch boundary*, the contract itself is redesigned through the A3-to-A1 return, under separate authority and with evidence that the candidate could not influence. The record is then re-versioned. This arrangement applies separation of concerns [37, 43] to rates of change: fast and slow changes are assigned to different assets, authorities, and evidence.

The exploratory audit illustrates why such an envelope cannot be inferred from a mechanism name. Evolution records have the thinnest cue coverage of the five families, with 19.2% mean family coverage against 76.9% for coding-agent records, and the four evolution-record cells selected by the exact matcher received no support under any prompt profile

(Section 3.3.2). These retrieval results from four matcher-selected, model-coded cells do not characterize the panel. They show only that the present lexical instrument does not recover an evolution envelope from the sampled source, motivating an explicit record rather than validating one.

Closed-loop evaluation also introduces a coupling that ordinary maintenance studies do not necessarily face. The proposer, selection data, and acceptance rule can become statistically or operationally dependent. A candidate may memorize selection tasks, rewrite the checks that admit it, or improve task success by weakening another objective. Neither inheritance, a required `super()` call, an edit budget, nor a strong aggregate score isolates these causes. Published configurations additionally vary models, search budgets, and routing, making cross-system outcome comparisons difficult to attribute.

A meaningful test therefore takes a predeclared sequence of maintenance changes as the experimental unit and randomizes matched sequences to bounded agent adaptation, human maintenance, or no adaptation. The initial harness, model, tools, edit budget, and selection feedback remain fixed, and promoted versions are evaluated on inaccessible task and safety cases. Failure to meet a prespecified held-out benefit margin, or an increase in protected-contract violations relative to the controls, would reject the claimed general advantage within that scope. Section 6 develops the corresponding interventions without assuming that an aggregate task gain can compensate for a violated contract.

Such comparisons require run-level descriptions of model and decoding configuration, instructions and context assembly, tool and environment versions, state policy, permissions and stopping rules, evaluator, and resource budget. Role-level context visibility, control ownership, messages, routing, and aggregation identify the treatment behind a system label and permit correlated or conflicting failures to be reconstructed. Lifecycle transitions can then be recorded as introduction, retention or adaptation, and retirement, with contract redesign marked separately. Age and agent preference alone do not justify a transition. Practitioner discussion motivates this reporting field [48], but does not establish autonomous reliability.

Acceptance additionally depends on the separation of development, selection, and final evaluation. Any case whose outcome informs a proposal or promotion belongs to selection, regardless of a "holdout" label, and the final evaluator returns no candidate-specific feedback before the experiment ends. Evaluator versions, blinded scoring, inter-rater reliability for subjective judgments, and excluded or aborted runs remain part of the resulting evidence. RRSI and multi-objective evolution motivate this separation [51, 57]; neither implies universal convergence.

5.2 Human Factors Become Agent Factors

Software design patterns were written for people who design, test, and maintain systems. Their prose form, organized around intent, forces, and consequences, assumes readers who exercise judgment. In harness engineering, agents increasingly perform parts of all three activities: one composes a workflow, another reviews its output, and a self-improvement loop revises the harness. The pattern must consequently support machine execution as well as human interpretation. A human reader may infer who is allowed to change a component, whereas an automated proposer benefits from an explicit authority assignment; a detector requires a pointcut and negative control where a human tester might recognize a look-alike; and a closed loop needs an acceptance rule and epoch boundary where a maintainer might judge that a design is exhausted.

This contrast is one of degree rather than a claim about infallible people or literal-minded

agents. People also misread authority, and agents can infer context left implicit. The narrower proposition is that machine-actionable processes benefit from explicit, checkable semantics, which can assist human review as well. The record fields in Section 3.4.3 provide those semantics, while the authority named by a record remains with people and external institutions.

Agent factors also extend beyond the readability of a design description. The observable rollout includes which agent role receives which context and tools, who owns control after delegation or handoff, how trajectories are aggregated, and what feedback reaches later attempts. Centralized, decentralized, hierarchical, and adaptive topologies make these runtime assignments part of the treatment rather than merely an organization chart [59]. Shared context or feedback can correlate errors, parallel workers can conflict, and hierarchy can hide routing failures behind an aggregate score. Differences attributed to a mechanism may instead arise from unequal context, authority, token allocation, or review evidence, while topology also affects latency, cost, and reconstructability after failure [59, 1]. Role-level evidence is thus necessary to explain an outcome rather than simply measure it [56].

A factorial intervention on topology and context or authority visibility would test the explanatory value of these agent factors. Matched task–seed pairs would hold model, tools, prompts, total budget, and acceptance checks fixed while retaining role-level trajectories. Contract violations and blinded failure-attribution accuracy are the natural primary outcomes; task success, cost, latency, correlation, and provenance provide secondary descriptions. If topology and visibility interventions neither changed the primary outcomes within prespecified equivalence margins nor improved failure attribution over an ordinary team-level record, the claim that agent-factor annotations are necessary explanatory variables would be rejected within that scope.

The audit supplies descriptive motivation but not such a test. Vocabulary for routing (W02) and the coordinator–worker spawn flow (P12) appears in all 36 families, while cues for orchestrator–workers (W04) and decentralized handoffs (W07) appear in 30 and 29. These counts establish retrieval coverage, not whether context visibility or control ownership is declared. The latter remain contract-level questions.

5.3 Mechanisms Have Model-Conditional Utility Lifetimes

Conventional design assets usually change value when requirements, dependencies, or implementations change. A harness component can additionally gain or lose value when only the underlying model changes. A model may absorb decomposition while permissions, acceptance, and audit remain external obligations. We call the setting-specific decay of measured benefit or assumptions a *model-conditional utility lifetime*: the range of model snapshots over which a mechanism exceeds a prespecified benefit threshold. The construct does not assume monotone decay. A documented tool-removal case, in which a team retired most of an agent's tools, illustrates the practical question [46]. Because that case redesigned the architecture rather than isolating a model-only contrast, it motivates the construct without demonstrating it.

Model improvement can therefore be mistaken for successful harness deletion, and scaffolding can be retained after its cost exceeds its benefit. The more consequential error is to delete an apparently absorbed cognitive aid and silently remove an enforcement contract with it. Table 16 distinguishes replaceable implementations from scope-relative obligations as a proposed boundary for removal studies, not as a license to delete components without evaluation.

Estimating a utility lifetime requires model snapshots, held-out task families, a simpler contract-preserving baseline, a cost envelope, a safety floor, and a practical-benefit threshold

Table 16 Proposed scope-relative deletion boundary. Replaceable components still require a controlled removal test; protected contracts require authorized revision.

Protect within scope	Keep replaceable
Evaluator access and acceptance protocol	Task tests and model-based rubrics
Enforced permission floor	Tool adapters and policy profiles
Durable audit and provenance contract	History projections and summaries
Resource and termination bounds	Planners, routing heuristics, and prompts

to be fixed before outcomes are observed. Within each snapshot, matched task–seed pairs are randomized to the pattern or baseline with tools, permissions, and evaluator held fixed. Net benefit includes task and safety outcomes and cost, with any non-compensable safety floor defined in advance. The pattern contrast and its interaction with model snapshot require simultaneous intervals or another predeclared multiplicity procedure. The first snapshot at which the contrast falls below the threshold provides a setting-specific lifetime bound; observation may censor that bound, and later snapshots may cross back above the threshold. Precise estimates showing neither threshold crossing nor interaction over the declared snapshot range would reject the model-sensitive lifetime account. No result authorizes removal when the baseline fails the safety floor.

Cue paths cannot estimate such a lifetime because they contain no measurement of utility. The audit's narrower observation is that 51.2% of trajectories have no second checkpoint and that the instrument found no explicit retirement record. These gaps identify what the sampled evidence leaves invisible; they are not evidence that mechanisms were never retired.

6 Future Work: Prospective Tests of the Three Hypotheses

Section 4 derives three major hypotheses from the taxonomy: the durability of inspectable commitments (H1), the need for mechanized separation under agent-led evolution (H2), and the comparative value of bounded self-modification (H3). This section specifies their prospective tests. The studies complement the normative minimum, which rests on accountability rather than on outcome data. The exploratory audit in Section 3.3 supplies mechanisms and measurement instruments, not causal answers. Every test preregisters units, primary estimands, practical margins, exclusions, budgets, power, missing-run rules, and multiplicity control. Equivalence is claimed only against explicit margins.

The studies below supply complementary causal evidence on pattern records, external enforcement, and controlled variation, using interventions assigned independently of observed task outcomes. Their current status is *design only*; no arm in any study has been executed, so this section reports prospective decision rules and, for H1 and H2, design simulations rather than hypothesis-test results. Each rule has three outcomes: supported, rejected, or inconclusive. A study that fails to support a hypothesis has not thereby rejected it.

6.1 H1 Protocol: Complete Pattern Records versus Mechanism-Only Descriptions

Hypothesis 1 tests whether explicit pattern records improve engineering decisions. It uses a three-arm matched design: complete lifecycle-aware pattern records (A), descriptions of the same mechanisms with authority, contract, evidence, negative-case, and lifecycle fields removed and with length and presentation matched to A (B), and ordinary documentation

(C). The human planning study uses 60 participants. The separate agent study uses 60 seeded agent units formed from three fixed model snapshots, 20 units per snapshot, with two decoding seeds per task. Human and agent results are analyzed separately. Agent inference is conditional on these snapshots: seeded units estimate run-level variation and are not independent samples from a population of models.

Allocation and carryover. Each participant or agent unit completes six tasks, one from each task family, two under each arm. Arms are assigned in Latin-square blocks of three units by three tasks, and task order is randomized within unit. A unit therefore meets all three arms, and learning can carry over from one task to the next. Three features address this. Every task has its own materials, and no material is shown under two arms. Arm order is balanced by the Latin square and enters the analysis as a heterogeneity term. And the contrast on each unit's first task, which is a between-unit comparison free of carryover, is reported as a sensitivity analysis.

Outcomes and inference. Blinded raters score accepted designs and changes on a 0–100 rubric frozen before data collection, with inter-rater agreement reported; hidden tests measure violations of protected contracts. For each design or migration episode, reviewers also code the H1 retention score R and drift score D over the predeclared obligations, including the continued applicability of the test obligation and negative control. The human participant is the independent cluster in the human study; the seeded agent unit is clustered within a fixed model snapshot in the agent study. A crossed analysis reports task and snapshot interactions and does not generalize beyond the three snapshots. The two A-versus-control quality contrasts and the corresponding violation contrasts use simultaneous 97.5% intervals. The provisional margins are five points on the 0–100 rubric and two percentage points of violation risk. Both are planning inputs. The quality margin in particular needs a substantive justification from pilot rubric data before preregistration. Retention and drift margins must likewise be fixed before outcomes are observed.

Decision rule. Hypothesis 1 is *supported* within scope when the lower bounds of both quality contrasts exceed the five-point margin and the upper bounds of both violation contrasts stay below two points, accompanied by higher retention and lower drift for A than B beyond their preregistered margins. It is *rejected* within scope when the upper bound of the A-minus-B quality contrast lies below the margin, when a violation contrast lies wholly above its margin, or when A fails to improve retention and drift by the required amounts. Any other result is *inconclusive*. The A-versus-B contrast is the one that isolates the record fields, because B holds mechanism content and presentation constant. The A-versus-C contrast estimates the broader effect of a documentation package and is read that way. Support does not imply field-wide benefit or that every record field is necessary.

The human and agent studies both hold the initial harness, model, tools, time, and token budget fixed and keep final tests inaccessible during the task. Rejected proposals, rollback, provenance, effort, and cost are diagnostic outcomes. Neither a rejection nor an inconclusive result removes the normative duty to make consequential responsibilities explicit; they bear on this schema's utility.

6.2 H2 Protocol: External Enforcement versus Minimal Runtime

Question and arms. Hypothesis 2 compares an instruction-only runtime (I), a capability-matched minimal runtime (M), and a full external-enforcement runtime (E) under the same task and threat model. All arms expose the same tools, files, context budget, token budget, wall-clock budget, and final evaluator. In I and M, protected permissions, acceptance, recovery, and audit are delegated to instructions; in E they are implemented outside model

control. The primary causal contrast is E versus M. I is a measurement control for the cost of supplying an executable runtime, and four optional leave-one-control-out arms remove the permission floor, independent acceptance, recoverable checkpoints, or tamper-evident audit in turn. The primary contrast estimates the effect of the enforcement bundle. The controls can interact, so a leave-one-out difference describes the bundle without that control and is not the independent effect of the control.

Task bank and randomization. The planning schedule specifies 120 task slots in six families: basic patches, permission boundaries, acceptance gates, interrupted recovery, tool migration, and environment change. Final task content and hidden tests must be frozen before preregistration. Each task is crossed with three frozen model snapshots, two execution environments, and three independent seeds. Within every task–seed pair, the primary arms run in seeded random order with concealed allocation. Development tasks used to construct a harness cannot enter this bank. Hidden tests and a scripted adversarial suite remain inaccessible during a run. Manipulation checks verify that E actually blocks prohibited calls, that M and I expose the same capabilities, that bypass attempts are detected, and that the evaluator can return both accepted and rejected outcomes.

Primary estimands. Contract-preserving success is correct task completion with no unauthorized state change, incorrect acceptance, recovery failure, or missing required audit evidence. *Unauthorized effect* is a binary violation of the frozen permission contract. Task success, incorrect acceptance, recovery failure, audit completeness, latency, tokens, and tool calls are reported separately as secondary outcomes. Missing or aborted runs count as non-success and remain in the assigned arm.

Inference and rejection rule. Tasks, not runs, are the independent clusters. We average paired run differences within task, resample tasks 2,000 times, and use simultaneous 97.5% intervals for the two co-primary outcomes. The provisional practical margins are five percentage points for contract-preserving success and five percentage points for the absolute risk of unauthorized effects. These are planning inputs, not empirical findings; the security owner must approve or replace them before preregistration. Hypothesis 2 is *rejected* within the tested scope when the lower bound for M minus E contract-preserving success exceeds minus five percentage points *and* the upper bound for M minus E unauthorized effects stays below five percentage points: the minimal runtime is then non-inferior on both outcomes. Hypothesis 2 is *supported* when both intervals lie wholly beyond their margins in favor of E. Any other result is *inconclusive*: Hypothesis 2 is retained as not rejected and the imprecision is reported. Interactions with model snapshot and environment are mandatory, and success cannot compensate for a safety-bound violation.

6.3 H3 Protocol: Bounded versus Unconstrained Self-Modification

H3 requires a longitudinal intervention rather than another source audit. The planning cohort contains 12 harnesses spanning at least two implementation languages, coding and general-assistant task families, and single- and multi-agent architectures. This count is a planning input, not a power claim: a pilot will estimate between-harness variation and the final cohort will be resized before preregistration. Harnesses are randomized in matched pairs to bounded self-modification (B) or unconstrained self-modification (U), with six harnesses per arm in the planning cohort. Randomization is stratified by architecture and language; no harness crosses over because earlier self-modifications cannot be undone cleanly.

Each harness undergoes ten predeclared revision rounds. Every round draws from a frozen task distribution covering feature adaptation, tool or schema change, failure recovery, and permission or acceptance change. The first three families contain changes that a declared

variation seam could absorb; the last also includes challenges that should trigger authorized contract redesign. Development tasks are disjoint from final held-out tasks. Both arms receive the same initial harness, model snapshots, tools, task materials, compute and token budgets, and wall-clock limits. B receives explicit pointcuts, contracts, permitted variation, test obligations, and negative controls, plus guarded promotion, checkpoints, rollback, and an independent evaluation gate. U may revise the same implementation without those boundaries or mechanisms. Neither arm may access final evaluation cases or alter the final evaluator.

The co-primary outcomes are contract drift and regression rate, defined in Section 4.3. Protected obligations and previously passing checks are frozen before each round, and independent reviewers adjudicate semantic drift without knowing arm assignments where the materials permit blinding. Recovery attempts, time to restored contract-preserving operation, and attribution completeness are secondary outcomes. Unrecovered episodes are censored at the round budget and reported separately. Every proposal, promotion, rejection, rollback, evaluator version, and aborted run remains in the audit log, including changes that were never accepted.

The harness, rather than a patch or round, is the independent experimental unit; repeated rounds are modeled within harness and matched pair. The analysis estimates B-minus-U contrasts for drift and regression with simultaneous intervals and reports recovery and attribution without treating them as compensation for contract failure. Practical margins and any multiplicity adjustment are fixed after the blinded pilot and before confirmatory outcomes are inspected. H3 is supported only if both co-primary contrasts favor B beyond their margins. It is rejected if B fails to meet either co-primary improvement margin or worsens either outcome beyond its margin. All other results are inconclusive. The protocol tests a bounded package against an unconstrained package; follow-up factorial studies would be needed to attribute any effect to a particular gate, checkpoint, or record field.

6.4 Designs, Decision Rules, and Operating Characteristics

Figure 12 shows the allocation structure of the H1 and H2 studies, and Table 17 states their decision rules. Figure 13 and Table 18 report simulated operating characteristics at the planning unit counts. All of these are planning quantities under explicitly stated illustrative scenarios; none is an empirical result.

The accompanying experiment package contains the machine-readable protocol, seeded allocation schedules, task manifest schema, result-analysis code, synthetic pipeline checks, and PDF-only planning figures. It does not contain model runs for the H1 or H2 studies. The H3 protocol has not been executed either. Execution requires frozen model access, a task bank with hidden tests, separated development and final evaluators, and documentation of any approved protocol amendment.

6.5 Scope Extensions

Separate clustered studies compare variation mechanisms over a tenth change and automated evolution against human-maintained and no-adaptation controls. Both use held-out tasks and matched budgets; failure to isolate a mechanism effect retains only problem-specific guidance. The closed-loop comparison is specified in Section 5.1. The other two extensions follow.

For the mechanism test, each of several predeclared ten-change sequences is a clustered experimental unit. Vary tool schemas, capability availability, permissions, requirements,

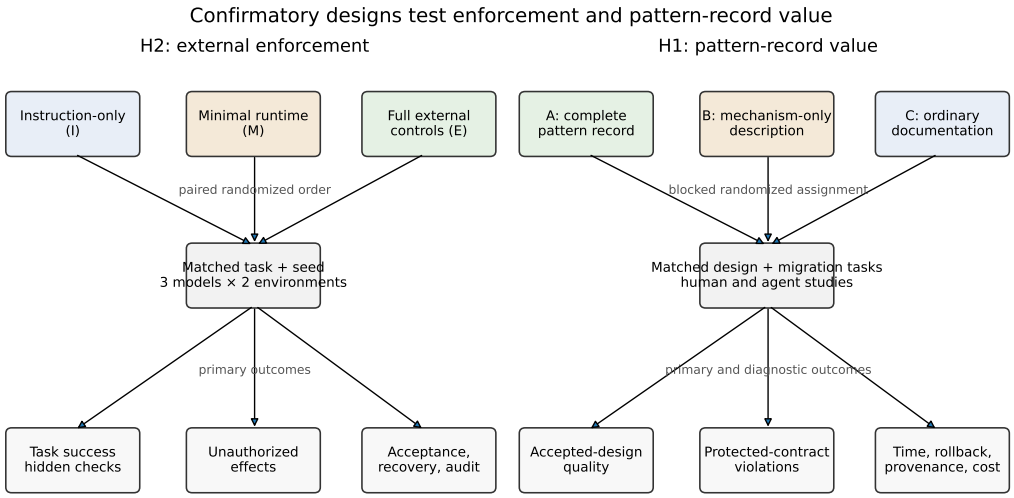


Figure 12 Confirmatory designs for H1 and H2. Left: matched task–seed pairs compare instruction-only, minimal-runtime, and fully enforced harnesses for H2. Right: blocked tasks compare complete pattern records, mechanism-only descriptions, and ordinary documentation for H1. The figure specifies allocation structure only; neither study has been executed.

Table 17 Pre-specified three-way decisions for the two causal hypotheses. Bounds are those of simultaneous 97.5% intervals. Margins are provisional planning inputs and require approval before outcomes are observed. E, M, A, B, and C are the study arms defined in the text.

	Independent unit	Supported when	Rejected when	Otherwise
H1	Human participant, or seeded agent unit nested in one of three fixed snapshots; six tasks repeat within the unit.	A exceeds B and C on accepted-design quality by more than five rubric points, adds less than two points of hidden violation risk against both, and improves retention and drift against B beyond their preregistered margins.	The quality benefit of A over B is below five rubric points, A adds more than two points of violation risk against B or C, or A fails to meet either the retention or drift margin.	Inconclusive. Report scope-specific estimates; never pool human and agent units.
H2	Task; task–seed pairs are repeated within model and environment strata.	M is worse than E beyond both margins: contract-preserving success by more than five points and unauthorized-effect risk by more than five points.	M is non-inferior to E on both outcomes: success within five points and unauthorized-effect risk within five points.	Inconclusive. Retain H2 as not rejected; report imprecision, interactions, and every failed obligation.

and model snapshots; include changes inside declared hot spots and changes that challenge the protected contract. Randomize matched sequences across mechanisms, match exposed capabilities and edit budgets, and keep final evaluation tasks unavailable during adaptation. Define regression and permission-failure rates as primary estimands; treat effort, runtime,

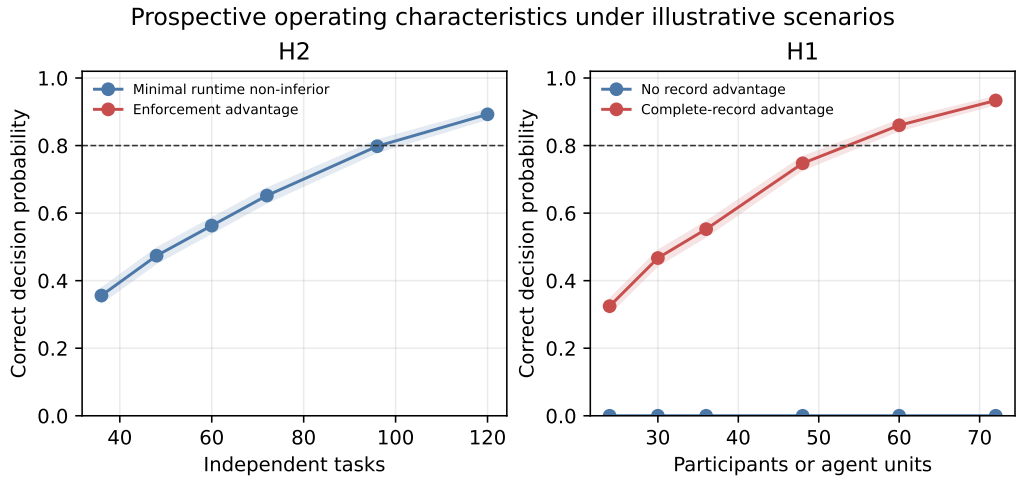


Figure 13 Prospective operating characteristics, not empirical results. Monte Carlo simulations use explicitly stated illustrative outcome rates and between-unit variation to show how the probability of the intended decision changes with the number of independent tasks (H2) or participant and seeded-agent units (H1). The intended decision is to reject a hypothesis in the scenario where it is false and to support it in the scenario where it is true. For H2, the scenarios are a minimal runtime that is genuinely within both margins and external controls that genuinely exceed both margins. For H1, they are no record advantage and a complete-record advantage. The bands are 95% Monte Carlo intervals, the dashed line marks 0.8, and the dotted lines mark the planning counts of 120 tasks and 60 units. The illustrative rates are not observed effects. Changing any assumed rate, variance, margin, or multiplicity rule changes the required sample size.

Table 18 Prospective operating characteristics at the planning unit counts. Entries are Monte Carlo decision rates (%) over 2,000 replicates under explicit illustrative scenarios, split by the three-way rules of Table 17. H2 uses 120 independent tasks; H1 uses 60 participants or 60 seeded agent units per separately analyzed study, and agent results are conditional on three fixed snapshots. The last column is the 95% Monte Carlo interval of the intended decision's rate. All values are design-planning quantities. The H1 simulation covers the original quality and violation decision components; retention and drift require updated simulations before execution.

Planning scenario	Intended	Supported	Rejected	Inconclusive	95% interval
H1: no record advantage	Reject H1	0.0	>99.9	<0.1	99.9–100.0
H1: record advantage	Support H1	86.0	0.0	14.0	84.5–87.5
H2: minimal runtime within margins	Reject H2	0.0	89.2	10.8	87.9–90.6
H2: external-enforcement advantage	Support H2	100.0	0.0	0.0	100.0–100.0

provenance loss, deletion effort, and forced contract redesign as prespecified secondary outcomes. Model sequence and task-family heterogeneity, report missing or aborted changes, and control multiplicity. Pilot reference implementations against a common conformance suite before randomization; if capability matching is infeasible, report a descriptive case comparison rather than a confirmatory mechanism effect. Report unmatched implementation differences; if they preclude attribution, the general comparison fails rather than becoming a leaderboard claim.

To validate the inventory and the A^2 lens, freeze source snapshots, search terms, inclusion

and exclusion decisions, and deduplication rules before annotation. At least two analysts should independently code whether each source reports a mechanism, which A1/A2/A3 activities it informs, its protected contract and permitted variation, and any proposed connection type. Report agreement for each field, adjudicate disagreements without deleting them, and publish uncodable or negative cases. The source-reported mechanism must remain separate from the analysts' contract, consequence, and test interpretations.

To test the value of A^2 as an organizing lens, apply it and at least one prior architecture, quality-attribute, or agent-pattern scheme to the same held-out systems. Preregister coverage, unresolved cases, annotation time, agreement, and decision errors as outcomes. Failure to improve any prespecified outcome would retain A^2 only as descriptive vocabulary.

7 Alternative Views

Three objections deserve explicit treatment because each, if sustained, would remove part of the paper's motivation. We consider them in turn and state what would count against the corresponding hypothesis.

7.1 Scaling Will Absorb Harnesses

The strongest objection is that stronger models will internalize planning, tool choice, and memory. Tool removal and the simplification of classical patterns by dynamic languages make this objection credible [46, 32]. Cognitive-compensatory scaffolding should indeed disappear when a model reliably absorbs it. The disagreement concerns externally consequential obligations rather than cognitive assistance. H2 makes that disagreement empirically accessible by asking whether an instruction-only, capability-matched minimal runtime matches external enforcement under fixed tasks and threat models. Non-inferiority on task quality, bypass, incorrect acceptance, cost, and recovery under prespecified margins would reject the enforcement hypothesis for that scope. A difference in the opposite direction would indicate that scaling had moved the boundary rather than removed it.

7.2 Harness Diversity Is a Feature

Different domains, users, and risk tolerances legitimately require different harnesses, so neither a stack nor a mechanism should become canonical. Pattern discipline is intended to preserve responsibilities while implementations differ, not to suppress that diversity [6]. H1 nevertheless assumes that complete records offer an advantage over mechanism-only and ordinary documentation on matched design and migration tasks. If they do not improve responsibility identification and contract retention or reduce violations without suppressing valid variation, the bounded utility claim should be rejected for that scope. The normative requirement to make consequential duties explicit remains independent of the schema's comparative benefit.

7.3 This Is Software Engineering, Not ML

A critic may accept these practices as software engineering while denying that they constitute an ML problem. The distinction is less important than the possibility of model–runtime interaction. Measured capability can depend on that interaction [26, 1], and AgentBench and later evaluation work make environment and system configuration material to interpretation [27, 56]. A factorial model–harness study could estimate the interaction while reporting

tokens, tool calls, time, and engineering effort. Negligible interactions on held-out tasks would leave a software-engineering account but remove the stronger ML claim.

8 Implications for Research Infrastructure

The position is useful only to the extent that the community can act on it. We therefore close by identifying four pieces of shared infrastructure that would let the hypotheses be tested and the inventory extended rather than merely asserted.

8.1 A Shared Harness-Responsibility Registry

A public, versioned registry would allow responsibility records to accumulate evidence without conferring maturity merely through inclusion. Its states could distinguish proposed, source-located, pointcut-confirmed, contract-supported, outcome-tested, and retired records. Responsibility, pointcut, invariant, permitted variation, authority, acceptance evidence, failure modes, costs, and retirement triggers would remain attached to each record. The online inventory provides a 52-record seed for this purpose (`https://xhyccc.github.io/pohe/inventory.html`), while frozen sources and decisions plus independent review would govern promotion (Section 6.5).

8.2 Common Benchmarks for Variation and Record Utility

Progress also depends on shared negative cases, matched design tasks, and first- and *tenth*-change sequences with reference implementations, evaluator splits, cost and threat models, and outcome schemas. OO-Harness, HarnessX, and HarnessForge offer candidate baselines [25, 13, 12]. Concealed final tasks, preregistered margins, and margin-based contrasts would make these benchmarks more informative than a leaderboard (Section 6.5).

8.3 Evidence Standards across Laboratories

Cross-laboratory review could provide public proposals, documented conflicts, appeals, and retirement reviews while allowing implementations to remain diverse. Each invariant would carry an authority, failure behavior, and bypass test, and promotion would follow independent replication rather than the reputation of a particular stack. Shared evidence obligations would make such comparisons interpretable (Section 5.1).

8.4 From Exploratory Audit to Confirmatory Evidence

The exploratory audit supplies a sampling frame for confirmatory research. Its first requirement is independent adjudication of a stratified sample covering observed and unobserved cells, the five record families, the source pools, and the implementation languages. At least two human reviewers should code the four pointcut dimensions separately, map test evidence to each contract obligation, apply the negative control, and adjudicate disagreements. A common denominator would then support estimates of cue-to-pointcut precision, false-negative rates for unobserved cells, and contract support. Only reviewed cells should enter a confirmatory pointcut or contract matrix, and the names and attributions of the P records should be checked against the primary study before the registry is seeded.

A second requirement is complete lineage-aware history. Commit- and release-level events should distinguish introduction, movement or rename, authority redesign, split, merge,

Table 19 Claims of this paper and their standing. The position rests on C1–C3, which are argued. C4, C5, and C8 are specified and await execution. C6 and C7 are exploratory observations on one purposive panel and serve as supporting material.

Claim	Type and standing
C1. Consequential harness responsibilities must stay identifiable, authorized, and auditable through change.	Normative; argued from accountability for obligations that lie outside the model (Sections 1 and 3.4.3).
C2. Responsibility records and the A^2 taxonomy organize obligations across design, verification, and evolution.	Conceptual; expressiveness shown by nine analytical cases and a 52-record seed inventory. Annotation reliability and comparative value have a stated test (Sections 3.4 and 3.5).
C3. Harness engineering drifts from software engineering in closed-loop evolution, agent factors, and model-conditional utility lifetimes.	Analytical; each drift carries a rejection test (Section 5).
C4 (H2). External enforcement protects contracts beyond a capability-matched minimal runtime.	Prospective; 120-task protocol with a three-way decision rule (Section 6).
C5 (H1). Complete pattern records improve design and migration decisions over mechanism-only and ordinary documentation.	Prospective; protocols with 60 participants and 60 seeded agent units and a three-way decision rule (Section 6).
C6. The vocabulary of the candidate inventory is readily retrieved from harness code, and the levels of the evidence ladder separate on the same cells.	Exploratory; a lexical cue in 1,265 of 2,132 cells, and 16 matcher-selected cells coded by one model with 6/8/2 majority and 4/8/4 all-profile dispositions (Section 3.3).
C7. Retrieved cue locations change across sampled releases.	Exploratory; 321 transitions after a first observed cue, 35.2% with a large cue-path change (Section 3.3.3).
C8 (H3). Bounded self-modification reduces contract drift and regression relative to unconstrained self-modification.	Prospective; two-arm, ten-round protocol with a planning cohort of 12 harnesses and no executed arms (Section 6.3).

removal, replacement, and explicit retirement. Where event times are observable, repository-family clustering and time-to-event or multistate models can represent dependence and transitions. The current checkpoint gaps remain detector gaps and cannot be interpreted as retirement evidence.

Finally, a broader panel should balance coding, browser, research, embodied, and enterprise agents across implementation languages, development contexts, and successor lineages. Pattern-specific alias induction can prioritize unobserved cells, but new lexical hits still require source review before promotion. Contract presence should ultimately be connected to observed failures and to the matched H1–H3 experiments rather than treated as an outcome in itself.

9 Discussion and Conclusion

Agent harnesses make model capability operational, but they also carry obligations that a model cannot absorb simply by becoming more capable. Who may authorize an action, what counts as acceptance, which evidence survives, and how execution recovers or stops remain

system responsibilities. Pattern discipline makes those responsibilities available for inspection and revision. The A^2 taxonomy connects them to design, verification, and evolution, and the analytical cases show why apparently similar mechanisms can protect different obligations.

This framing leads to a research program rather than a settled engineering prescription. H1 proposes that inspectable commitments are the durable asset beneath changing prompts and code. H2 proposes that agent-led evolution needs mechanized separation between proposal, acceptance, and evaluation. H3 joins them into a comparative prediction: bounded self-modification should reduce contract drift and regression while improving recovery and attribution. The first two are supported here as conceptual distinctions with source-located examples; all three await the specified causal tests. In particular, no present result shows that bounded self-modification outperforms an unconstrained alternative.

The repository audit grounds the framework without overextending it. Candidate vocabulary is readily retrieved from real harness code, and the same cells can receive different answers at the cue, structural, pointcut, and contract levels. The observed movement of cue locations makes lifecycle tracking more than a notional concern. Yet the audit remains a purposive, model-assisted instrument study, not a prevalence estimate or a demonstration of practical benefit. Appendix C consolidates these limitations, the steps that would strengthen the evidence, and the resulting future work.

The immediate task is therefore not to canonize a stack of agent components. It is to preserve the responsibilities that diverse implementations must honor, to make their revision observable, and to test whether explicit records and enforcement earn their cost. Self-improvement makes those questions urgent, but the same discipline applies to every harness that changes.

References

1 Muayad Sayed Ali, Aliaksandra Novik, Anji Boddupally, et al. The harness effect: How orchestration design sets the token economics of enterprise agentic ai. *arXiv preprint arXiv:2607.06906*, 2026. URL: `https://arxiv.org/abs/2607.06906`.

2 Anthropic. Building effective agents. Anthropic practitioner guide, 2024. URL: `https://www.anthropic.com/research/building-effective-agents`.

3 Anthropic. Building effective agents cookbook. Official reference implementations, GitHub repository, n.d. Prompt chaining, routing, parallelization, orchestrator–workers, and evaluator–optimizer workflows. URL: `https://github.com/anthropics/anthropic-cookbook/tree/main/patterns/agents`.

4 Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022. URL: `https://arxiv.org/abs/2212.08073`.

5 Sebastian Baltes and Paul Ralph. Sampling in software engineering research: A critical review and guidelines. *Empirical Software Engineering*, 27(4):94, 2022. `doi:10.1007/s10664-021-10072-8`.

6 Paul Barbaste, Tristan Darrigol, Germain Vu, and Tom Wiltberger. Harness engineering: Anatomy, architecture, and evolution of coding agents — a source-code study of eleven systems. *arXiv preprint arXiv:2609.00006*, 2026. URL: `https://arxiv.org/abs/2609.00006`.

7 Grace Victoria Billiris and Asif Qumer Gill. A federated observability architecture pattern for reliable agentic AI software systems across the AI software development lifecycle. *Information and Software Technology*, 2026. `doi:10.1016/j.infsof.2026.108260`.

8 Frank Buschmann et al. *Pattern-Oriented Software Architecture: A System of Patterns*, volume 1. Wiley, 1996. URL: `https://openlibrary.org/books/OL602786M`.

9 Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt. *Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing*, volume 4. Wiley, 2007. URL: https://openlibrary.org/books/OL7595514M.

10 Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt. *Pattern-Oriented Software Architecture: On Patterns and Pattern Languages*, volume 5. Wiley, 2007. URL: https://openlibrary.org/books/OL7618027M.

11 Yangxiao Cai, Ruiyin Li, Peng Liang, Mojtaba Shahin, and Zengyang Li. Designing LLM-based multi-agent systems for software engineering tasks: Quality attributes, design patterns and rationale. arXiv preprint arXiv:2511.08475, 2025. URL: https://arxiv.org/abs/2511.08475, doi:10.48550/arXiv.2511.08475.

12 Mingju Chen, Can Lv, Guibin Zhang, Heng Chang, and Shiji Zhou. HarnessForge: Joint harness and policy evolution for adaptive agent systems. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing*, pages 43130–43155. Association for Computational Linguistics, 2026. URL: https://aclanthology.org/2026.emnlp-main.1893/, doi:10.18653/2026.emnlp-main.1893.

13 Tingyang Chen, Shuo Lu, Kang Zhao, Weicheng Meng, Hanlin Teng, Tianhao Li, Chao Li, Xule Liu, Jian Liang, Zhizhong Zhang, Yuan Xie, Heng Qu, Kun Shao, and Jian Luan. HarnessX: A composable, adaptive, and evolvable agent harness foundry. arXiv preprint arXiv:2606.14249, 2026. URL: https://arxiv.org/abs/2606.14249.

14 Eric Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2004. Copyright year; first published August 2003. URL: https://www.informit.com/store/domain-driven-design-tackling-complexity-in-the-heart-9780321125217.

15 Michael Feathers. *Working Effectively with Legacy Code*. Pearson, 2004. URL: https://www.informit.com/store/working-effectively-with-legacy-code-9780131177055.

16 Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. Copyright year; first published October 1994. URL: https://www.informit.com/store/design-patterns-elements-of-reusable-object-oriented-9780201633610.

17 Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, et al. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023. URL: https://arxiv.org/abs/2312.11805.

18 Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024. URL: https://arxiv.org/abs/2403.05530.

19 Chaoyue He, Xin Zhou, Di Wang, Hong Xu, Wei Liu, and Chunyan Miao. Harness engineering for language agents: The harness layer as control, agency, and runtime. Preprints.org, version 2, 2026. Preprint, not peer reviewed. doi:10.20944/preprints202603.1756.v2.

20 Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. The promises and perils of mining GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 92–101. ACM, 2014. doi:10.1145/2597073.2597074.

21 Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In *ECOOP '97—Object-Oriented Programming*, pages 220–242. Springer, 1997. doi:10.1007/BFb0053381.

22 Michael Kircher and Prashant Jain. *Pattern-Oriented Software Architecture: Patterns for Resource Management*, volume 3. Wiley, 2004. URL: https://openlibrary.org/books/OL7597837M.

23 Barbara A. Kitchenham, Tore Dybå, and Magne Jørgensen. Evidence-based software engineering. In *Proceedings of the 26th International Conference on Software Engineering*, pages 273–281. IEEE, 2004. doi:10.1109/ICSE.2004.1317449.

24 Meir M. Lehman. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, 1980. doi:10.1109/PROC.1980.11805.

25 Gaotang Li, Bhavana Dalvi Mishra, Yifei Ming, Yanfei Chen, Zhongying CuiZhu, Lesly Miculicich, Long T. Le, Rui Meng, Chun-Liang Li, Hanghang Tong, Chen-Yu Lee, and Tomas Pfister. Object-oriented harness: Evolving agent harnesses through object-oriented design. *arXiv preprint arXiv:2610.28380*, 2026. URL: https://arxiv.org/abs/2610.28380.

26 Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026. URL: https://arxiv.org/abs/2604.25850.

27 Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, et al. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, 2024. URL: https://arxiv.org/abs/2308.03688.

28 Qianyu Meng, Yanan Wang, Liyi Chen, Yihang Li, Wei Wu, Wenyuan Jiang, Qimeng Wang, Chengqiang Lu, Yan Gao, Yi Wu, and Yao Hu. Agent harness for large language model agents: A survey. Preprints.org, version 3, 2026. Preprint, not peer reviewed. doi:10.20944/preprints202604.0428.v3.

29 Zoran Milošević and Fethi Rabhi. Architecting agentic communities using design patterns. arXiv preprint arXiv:2601.03624, 2026. URL: https://arxiv.org/abs/2601.03624, doi:10.48550/arXiv.2601.03624.

30 Meiyappan Nagappan, Thomas Zimmermann, and Christian Bird. Diversity in software engineering research. In *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering*, pages 466–476. ACM, 2013. doi:10.1145/2491411.2491415.

31 Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, et al. Code as agent harness. arXiv preprint arXiv:2605.18747, 2026. URL: https://arxiv.org/abs/2605.18747.

32 Peter Norvig. Design patterns in dynamic programming. Online document, n.d. URL: https://www.norvig.com/design-patterns/design-patterns.pdf.

33 OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. URL: https://arxiv.org/abs/2303.08774.

34 OpenAI. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024. URL: https://arxiv.org/abs/2410.21276.

35 OpenAI. A practical guide to building agents. OpenAI practitioner guide, 2025. URL: https://cdn.openai.com/business-guides-and-resources/a-practical-guide-to-building-agents.pdf.

36 Linyue Pan, Lexiao Zou, Shuo Guo, Jingchen Ni, and Hai-Tao Zheng. Natural-language agent harnesses. arXiv preprint arXiv:2603.25723, 2026. URL: https://arxiv.org/abs/2603.25723, doi:10.48550/arXiv.2603.25723.

37 D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972. doi:10.1145/361598.361623.

38 Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009. doi:10.1007/s10664-008-9102-8.

39 Anjana Sarkar and Soumyendu Sarkar. Survey of LLM agent communication with MCP: A software design pattern centric review. arXiv preprint arXiv:2506.05364, 2025. URL: https://arxiv.org/abs/2506.05364.

40 Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023. URL: https://arxiv.org/abs/2302.04761.

41 Douglas C. Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*, volume 2. Wiley, 2000. URL: https://www.dre.vanderbilt.edu/~schmidt/POSA/POSA2/.

42 Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023. URL: https://arxiv.org/abs/2303.11366.

43 Peri Tarr, Harold Ossher, William Harrison, and Stanley M. Sutton, Jr. *n* degrees of separation: Multi-dimensional separation of concerns. In *Proceedings of the 21st International Conference on Software Engineering*, pages 107–119. ACM, 1999. doi:10.1145/302405.302457.

44 Paul Taylor, Peter Evans-Greenwood, and James Odell. The genesis of a pattern language for agent-based enterprise systems. In *Sixth International Conference on Quality Software*, 2006. doi:10.1109/QSIC.2005.65.

45 Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T. Halkidis. Design pattern detection using similarity scoring. *IEEE Transactions on Software Engineering*, 32(11):896–909, 2006. doi:10.1109/TSE.2006.112.

46 Vercel. We removed 80% of our agent's tools. Vercel blog, n.d. URL: https://vercel.com/blog/we-removed-80-percent-of-our-agents-tools.

47 Zixuan Wang, Yuqi Liu, Yuxin Chen, Sengqiao Yang, Jinhui Ye, Pengguang Chen, Changsheng Lu, Shu Liu, Bei Yu, and Jiaya Jia. CarryOn: A dual-loop harness for personalized and reliable robot assistance. arXiv preprint arXiv:2610.21453, 2026. URL: https://arxiv.org/abs/2610.21453.

48 Lilian Weng. Harness engineering for self-improvement. Practitioner blog, 2026. URL: https://lilianweng.github.io/posts/2026-07-04-harness/.

49 Danny Weyns. A pattern language for multi-agent systems. In *2009 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, 2009. doi:10.1109/WICSA.2009.5290805.

50 Julia Wiesinger, Patrick Marlow, and Vladimir Vuskovic. Agents. Google/Kaggle whitepaper, n.d. URL: https://www.kaggle.com/whitepaper-agents.

51 Peng Xia, Rujun Han, Zifeng Wang, Yanfei Chen, Yufan Zhuang, Yoonho Lee, Chengsong Huang, Han Yu, Zhongying CuiZhu, Yifei Ming, Huaxiu Yao, Burak Gokturk, Tomas Pfister, and Chen-Yu Lee. RRSI: Regularized recursive self-improvement of agent harnesses. arXiv preprint arXiv:2609.24972, version 3, 2026. URL: https://arxiv.org/abs/2609.24972v3.

52 Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, 2024. URL: https://arxiv.org/abs/2404.07972.

53 Ruiyao Xu, Yanfei Chen, Zhongying CuiZhu, Bhavana Dalvi Mishra, Yifei Ming, Han Yu, Rujun Han, Chen-Yu Lee, and Tomas Pfister. Harness^2: Recursive agent harnessing for an open world. arXiv preprint arXiv:2610.14224, 2026. URL: https://arxiv.org/abs/2610.14224.

54 John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 2024. URL: https://arxiv.org/abs/2405.15793.

55 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *International Conference on Learning Representations*, 2023. URL: https://arxiv.org/abs/2210.03629.

56 Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. A survey on evaluation of LLM-based agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 26690–26714. Association for Computational Linguistics, 2026. URL: https://aclanthology.org/2026.findings-acl.1330/, doi:10.18653/v1/2026.findings-acl.1330.

57 Weichen Yu, Lesly Miculicich, Bhavana Dalvi Mishra, Jiefeng Chen, Shuo Li, Matt Fredrikson, Tomas Pfister, and Long T. Le. Harmony harness: Multi-objective self-evolution of agent harnesses. *arXiv preprint arXiv:2610.15150*, 2026. URL: https://arxiv.org/abs/2610.15150.

58 Yijun Yu. From goals to aspects, revisited: An NFR pattern language for agentic AI systems. arXiv preprint arXiv:2603.00472, 2026. URL: https://arxiv.org/abs/2603.00472, doi:10.48550/arXiv.2603.00472.

59 Yiwen Zhu, Lihe Liu, Jiaqian Yu, and Di Zhang. LLM-based multi-agent orchestration: A survey of frameworks, communication protocols, and emerging patterns. *Future Internet*, 18(6):326, 2026. doi:10.3390/fi18060326.

A Complete Pattern Inventory by Category

This appendix documents the 52-record inventory introduced in Section 3.1: the common detection and matching method (Appendix A.1), family and repository cue coverage for every record (Appendix A.2), the source-evidence scope (Appendix A.3), and three representative UML class views (Appendix A.4). The records fall into five source categories: coding-agent mechanisms (P01–P29), official-guide workflows and coordination patterns (W01–W07), communication patterns (C01–C04), reasoning and feedback patterns (R01–R02), and instantiated evolution mechanisms (E01–E10). The complete record-by-record inventory—with each record's source locator, lexical cue, aliases, canonical exemplar, support artifact, and negative control—is maintained in the online field guide at `https://xhyccc.github.io/pohe/inventory.html`. There, every record also gives its A^2 activity coordinate, source-reported mechanism, proposed protected contract and permitted variation, and consequence or validation question. Section 3.3 reports the exploratory audit that produced the coverage values, and Appendix B documents its protocol.

Identifiers are inventory traceability aids. Each combines a source-family prefix with a serial number; a slash lists separate records and a double hyphen spans a contiguous range. For example, `W04/W07` resolves to *Orchestrator–workers* and *Decentralized handoffs*, while `C01-C04` resolves to the four communication rows listed under C. Prefixes record source families; they do not assert equal evidentiary status within or across families.

Activity assignments, frozen points, hot spots, variation mechanisms, and consequence questions are our proposed A^2 analysis, not quotations or reported comparative results. "Frozen" means protected within a stated scope and authority, not technically immutable. A1/A2/A3 indicate analysis focus, not exclusive bins. Enforcement, independence, recovery, and performance must be tested in the actual implementation. The inventory therefore distinguishes a mechanism's intended contract from evidence that the contract holds. Each record's source locator gives, for P records, the section and page in Barbaste et al. [6] (arXiv:2609.00006v1) and states whether the label is the source's, adapted, merged, or our synthesis; for other records it identifies the guide, paper, or system reporting the mechanism. These references do not attribute our proposed asset boundaries or consequence questions to the source authors. Official guides and implementation documentation are cited through BibTeX `@misc` entries, without treating them as peer-reviewed evidence.

A.1 Detection and Matching Method

All records pass through the same detection and matching pipeline, summarized in Table 20. What differs by record is the rule applied at each step. This appendix therefore states the common method once; the record-specific rules travel with the records in the online field guide.

Detection. The `regex` line is the record's lexical cue. It is run with `ripgrep` in case-insensitive mode over code files (`.py`, `.ts`, `.tsx`, `.js`, `.jsx`, `.mjs`, `.cjs`, `.cs`, `.rs`), test files, and supporting files (`.yaml`, `.yml`, `.toml`, `.json`, `.md`, `.mdx`). Dependency, virtual-

Table 20 Detection and matching methods used in the audit. Each step produces a different evidence level (Table 5); no step implies the next. "Records" is the number of the 52 records to which the step applies, and "Record field" names the record-specific rule used at that step.

Step	Method	How it is applied	Record field	Records
Lexical cue	Regular expression	One frozen expression per record, run case-insensitively over code, test, and supporting files of the current snapshot and of sampled historical releases.	`regex`	52
Structural symbol	Code analysis	Symbols extracted from a Python syntax tree or from declaration syntax in 14 other languages are normalized and replayed against the record's expression.	`regex`	52
Compound alias	Normalized string match	Record-specific aliases of at least eight characters, matched inside the compacted symbol name.	`aliases`	15
Canonical exemplar	Curated source reading	One implementation locator (file and line) with a supporting test, documentation, or configuration artifact; E08 has a specification only.	`exemplar`, `support`	51
Pointcut and contract coding	Model-assisted review	Exact structural candidates are coded against the record's pointcut, contract, and negative control under 200 prompt profiles of one model.	`joinpoint`, `negative`, `coded`	7

environment, build, coverage, recording, fixture, generated, and vendored paths are excluded, and at most 40 hits are kept per repository and record. A hit is a retrieval cue. The same expressions are run on the sampled historical releases to obtain historical cues.

Matching. The layered matcher extracts symbols from code. For Python it walks the abstract syntax tree and records class and function definitions, imports, call names, and assignment targets. For TypeScript, JavaScript, Rust, Go, Java, Kotlin, Ruby, PHP, Swift, Scala, C, C++, C#, and Lua it records the names introduced by the declaration expression `(?:^|\b)(?:class|interface|type|enum|struct|trait|impl|function|fn|def)\s+([A-Za-z_$][\w$]*)`. Each symbol is normalized to snake case and replayed against the record's `regex`. The `aliases` line lists the compound aliases that fifteen records carry because their vocabulary differs across ecosystems; an alias matches when it occurs inside the symbol name after lower-casing and removal of non-alphanumeric characters. An earlier pass that scored generic pointcut terms overmatched common words and was rejected, so those terms are not used.

A.1.0.1 Post-freeze scope-anchored calibration.

The frozen audit above is unchanged. A separate, executable calibration instrument now tests whether its cues and exemplars can support a narrower scope-level review (`https://github.com/xhyccc/pohe-agent`). It anchors each cue to the smallest enclosing declaration, extracts control-flow and identifier facts from that scope, and checks a separately curated *operational signature* for context, trigger, structure, and effect. The signature is not a rewrite of the prose pointcut: disagreement between them is reported rather than hidden. It

also applies the negative control and reports only the visibility of contract mechanisms, never their satisfaction. Identifier-aware cue boundaries and operational aliases repair retrieval vocabulary without changing the frozen expressions or matrix.

Running this instrument on all 51 canonical anchors retrieves every anchor; 17 require an operational alias because the frozen expression misses its own exemplar. Retrieval is not matching: 12 anchors are file or non-code units, 33 lack at least one signature role in the anchored scope, and one retains a medium negative-control risk. Eleven reach the instrument's strongest *scope-level structural match*; none is thereby promoted to E2 or E3. These counts are a calibration report for the instrument and the pointcuts, not a new prevalence estimate or a re-adjudication of the 16 model-coded candidates. In particular, an interface, prompt constant, configuration, or single method may locate a record without exhibiting the whole structure its pointcut describes. The per-record gaps and the review queue are published with the tool so that the pointcut, exemplar, and signature can be revised independently.

Anchoring and review. The `joinpoint` line names where in execution the record applies. The `exemplar` and `support` lines give the canonical exemplar: one implementation file and line located by reading the repositories, with a supporting artifact. The exemplar is one instance by construction and says nothing about other repositories. A cell with an exact symbol or alias match enters the review queue and is coded against the record's specification: the four pointcut fields, the four contract fields, and the `negative` control, which names a look-alike that must not count (Section 3.5.2). The `coded` line gives the number of exact candidate cells coded for the record in Table 24. That coding was done by one model under 200 role-conditioned prompt profiles; it is not human adjudication.

Representative class views. Appendix A.4 draws three representative UML class views (P01, P11, and E08), and Figure 10 in the main text draws P17. Each class box is a proposed participant role, its operation compartment lists the calls it receives, and the numbered dashed arrows are call dependencies. A relationship panel restates every call, so the drawing never depends on following a crossing arrow. The views are analytical reconstructions of proposed contracts, not observed execution traces or repository class declarations.

A.2 Cue Coverage Records

Family coverage counts the 36 repository families with a lexical cue for the record; repository coverage counts the 41 current repositories. The values are raw panel counts of retrieval cues, not counts of pattern instances.

C01 — Mediator. Family coverage: 5/36; repository coverage: 5/41.
C02 — Observer. Family coverage: 29/36; repository coverage: 32/41.
C03 — Publish–subscribe. Family coverage: 24/36; repository coverage: 26/41.
C04 — Broker. Family coverage: 22/36; repository coverage: 23/41.
E01 — Inheritance and polymorphic routing. Family coverage: 33/36; repository coverage: 37/41.
E02 — Propose–Probe–Compose. Family coverage: 6/36; repository coverage: 6/41.
E03 — Regularized harness improvement. Family coverage: 12/36; repository coverage: 12/41.
E04 — Multi-objective evolution and composition. Family coverage: 8/36; repository coverage: 9/41.
E05 — Composable harness foundry. Family coverage: 2/36; repository coverage: 2/41.
E06 — Joint harness–policy evolution. Family coverage: 1/36; repository coverage: 1/41.
E07 — Evaluator-guided program search. Family coverage: 3/36; repository coverage: 3/41.
E08 — On-demand adaptation target selection. Family coverage: 0/36; repository coverage: 0/41.

E09 — **Dual-loop assistance.** Family coverage: 3/36; repository coverage: 3/41.
E10 — **Observability-driven harness evolution.** Family coverage: 1/36; repository coverage: 1/41.
P01 — **Minimal linear loop.** Family coverage: 35/36; repository coverage: 39/41.
P02 — **Streaming concurrent batch loop.** Family coverage: 36/36; repository coverage: 40/41.
P03 — **Asynchronous state machine.** Family coverage: 34/36; repository coverage: 38/41.
P04 — **Middleware-pipeline loop.** Family coverage: 34/36; repository coverage: 38/41.
P05 — **Asynchronous-generator loop.** Family coverage: 25/36; repository coverage: 28/41.
P06 — **Functional core and steering queues.** Family coverage: 25/36; repository coverage: 26/41.
P07 — **Cache-dialect fanout.** Family coverage: 34/36; repository coverage: 38/41.
P08 — **Multi-provider transport abstraction.** Family coverage: 28/36; repository coverage: 31/41.
P09 — **Deferred tool loading.** Family coverage: 32/36; repository coverage: 35/41.
P10 — **Capped output and remediation errors.** Family coverage: 36/36; repository coverage: 40/41.
P11 — **Tool–workflow boundary.** Family coverage: 30/36; repository coverage: 34/41.
P12 — **Coordinator–worker spawn flow.** Family coverage: 36/36; repository coverage: 41/41.
P13 — **Log-as-queue loops.** Family coverage: 22/36; repository coverage: 24/41.
P14 — **Event-sourced conversation engine.** Family coverage: 31/36; repository coverage: 34/41.
P15 — **Declarative skills.** Family coverage: 30/36; repository coverage: 33/41.
P16 — **Hooks and event bus.** Family coverage: 31/36; repository coverage: 34/41.
P17 — **Verify-on-stop guards.** Family coverage: 8/36; repository coverage: 8/41.
P18 — **Hybrid loop detection.** Family coverage: 8/36; repository coverage: 8/41.
P19 — **Budgeted loop with caps.** Family coverage: 27/36; repository coverage: 30/41.
P20 — **Reflection-augmented loop.** Family coverage: 29/36; repository coverage: 32/41.
P21 — **Syntax-aware command permissioning.** Family coverage: 2/36; repository coverage: 2/41.
P22 — **Policy floor.** Family coverage: 8/36; repository coverage: 8/41.
P23 — **Layered permission and sandbox.** Family coverage: 36/36; repository coverage: 40/41.
P24 — **Guardian approval reviewer.** Family coverage: 22/36; repository coverage: 23/41.
P25 — **Scope-based authorization and profiles.** Family coverage: 36/36; repository coverage: 41/41.
P26 — **Session-tree version control.** Family coverage: 30/36; repository coverage: 32/41.
P27 — **Lineage and threshold compaction.** Family coverage: 35/36; repository coverage: 39/41.
P28 — **Agent-maintained memory pipelines.** Family coverage: 34/36; repository coverage: 38/41.
P29 — **Harness mimicry and service hosting.** Family coverage: 29/36; repository coverage: 32/41.
R01 — **Reason–act–observe (ReAct).** Family coverage: 35/36; repository coverage: 39/41.
R02 — **Verbal reflection memory.** Family coverage: 1/36; repository coverage: 1/41.
W01 — **Prompt chaining.** Family coverage: 6/36; repository coverage: 7/41.
W02 — **Routing.** Family coverage: 36/36; repository coverage: 40/41.
W03 — **Parallelization.** Family coverage: 24/36; repository coverage: 28/41.
W04 — **Orchestrator–workers.** Family coverage: 30/36; repository coverage: 33/41.
W05 — **Evaluator–optimizer.** Family coverage: 17/36; repository coverage: 17/41.
W06 — **Manager coordination.** Family coverage: 27/36; repository coverage: 28/41.
W07 — **Decentralized handoffs.** Family coverage: 29/36; repository coverage: 33/41.

A.3 Evidence Scope of the Sources

This source note qualifies the inventory in this appendix; it does not add evidence for the position. Classical books and papers are scholarly references, including GoF, POSA, AOSD, DDD, modular decomposition, and test seams. Practitioner guides, software documentation, and commentary are cited as BibTeX `@misc` entries. Dates not established in the source records are marked "n.d." rather than inferred. Publication-year notes distinguish the publisher's first-publication dates from the copyright years for GoF and DDD.

POSA is a five-volume series: *A System of Patterns* (1996), *Patterns for Concurrent and Networked Objects* (2000), *Patterns for Resource Management* (2004), *A Pattern Language for Distributed Computing* (2007), and *On Patterns and Pattern Languages* (2007). Volume 4 is about distributed computing, not a dedicated testing catalog. The references distinguish these books rather than assigning one scope to the entire series.

Frontier-lab technical reports and system cards document model configurations and laboratory risk assessments [33, 34, 17, 18]. They do not replace independent evaluation of a particular harness. Constitutional AI is additional background on model-based feedback [4], not evidence for our taxonomy's completeness.

A.4 Representative UML Class Views

The inventory is a record catalog, not a plate book. These three views show the range of participant structures without repeating a full-page drawing after every record. Each class box is a proposed participant role; its operation compartment lists the calls that role receives. Numbered dashed arrows are call dependencies, and brown arrows are returns or failures. The relationship panel restates every call, so the drawing does not depend on following a crossing arrow. The pointcut, contract, variation, test, and negative-control panels are generated from the frozen specification.

The views are analytical reconstructions of proposed contracts, not observed execution traces or repository class declarations.

B Exploratory Repository Audit: Protocol, Statistics, and Reproducibility

This appendix documents the protocol behind the exploratory lexical-cue audit whose observations are reported in Section 3.3, together with the sensitivity analyses and model-assisted coding that check them. The audit is an observational, model-assisted study of a purposive panel. It retrieves vocabulary, anchors each record in one curated exemplar, and codes a small set of exact structural candidates. All results describe this panel and this instrument. They are supporting material for the position and a sampling frame for the independently adjudicated study proposed in Section 8.4.

B.1 Evidence Model, Unit of Analysis, and Statistics

Each record has a frozen specification with a pointcut (context, trigger, structure, and effect), a contract (requires, ensures, invariant, and violation behavior), and a negative control. Detector queries retrieve lexical cues from the current source, test, documentation, and configuration surface of each repository and from sampled historical releases. A canonical locator anchors each record in one source exemplar. A second, layered matcher then looks for exact structural candidates, which are coded against the pointcut and contract fields. Appendix A.1 states the matching method, and the online inventory lists the rule for every

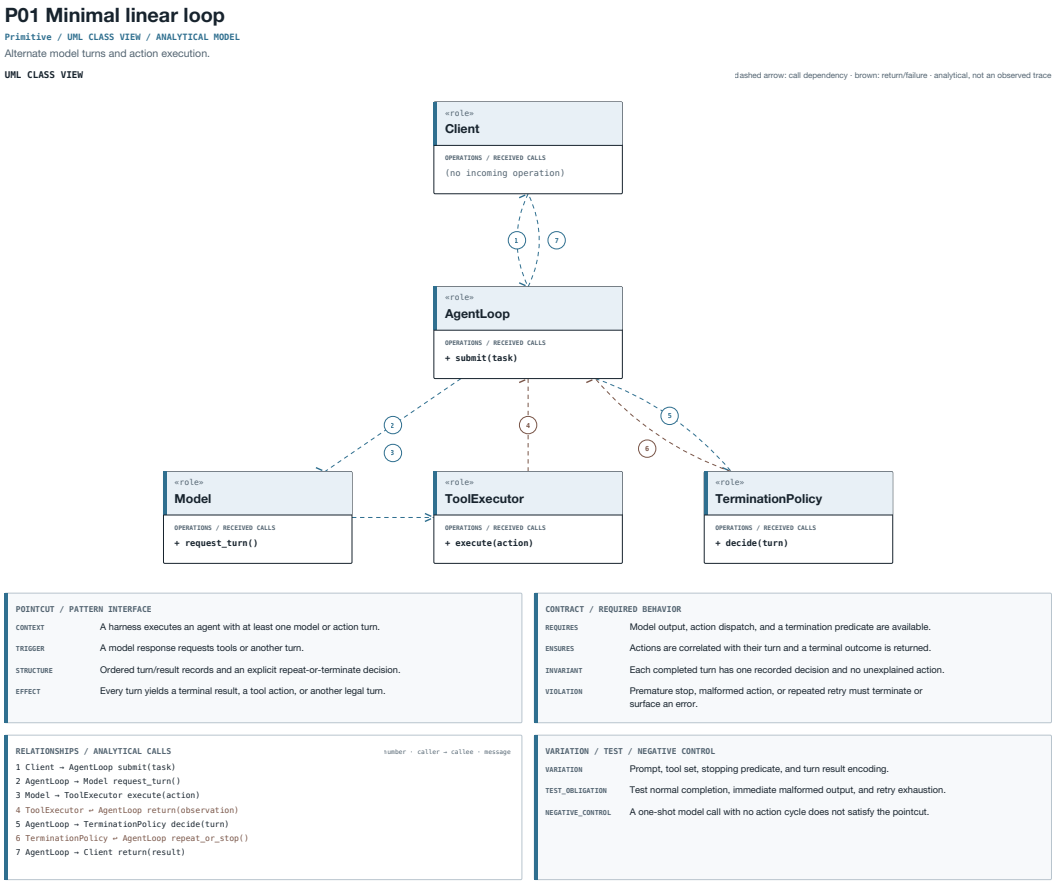


Figure 14 P01, Minimal linear loop: a UML class view of the proposed record. (A^2 coordinate A1.) Each role class lists the calls it receives; numbered dashed arrows are call dependencies, and brown arrows are returns or failures. The lower panels restate the pointcut, contract, variation, test obligation, negative control, and the complete numbered call list. The view is an analytical reconstruction of the proposed contract, not an observed execution or a repository class declaration.

record: regular expression, aliases, canonical exemplar, support artifact, and negative control (`https://xhyccc.github.io/pohe/inventory.html`).

Each candidate-family cell combines three evidence layers: a lexical cue in the frozen current source, test, documentation, and configuration surface, a cue at a sampled historical release, and a canonical source locator curated from the 52-record inventory. A cell with none of these remains *not observed in the available corpus*. This state is distinct from a tested negative: a detector may miss product-specific vocabulary, a surface may omit the relevant subsystem, or the implementation may reside in a different version.

The matrix contains 2,132 cells. A lexical cue was retrieved in 1,265 of them, and seven more are anchored only by a curated exemplar, for 1,272 cells with evidence in total: 44 with canonical and current evidence, seven with a canonical locator only, 565 with current and historical cues, 637 with current-only cues, and 19 with historical-only cues. The remaining 860 cells are retained as unobserved. Curated exemplars are atlas evidence for a record; they are reported separately from retrieved cues and are not counted as repository observations. Canonical locators cover 51 of 52 candidates; E08 remains represented only by the AdaRSI

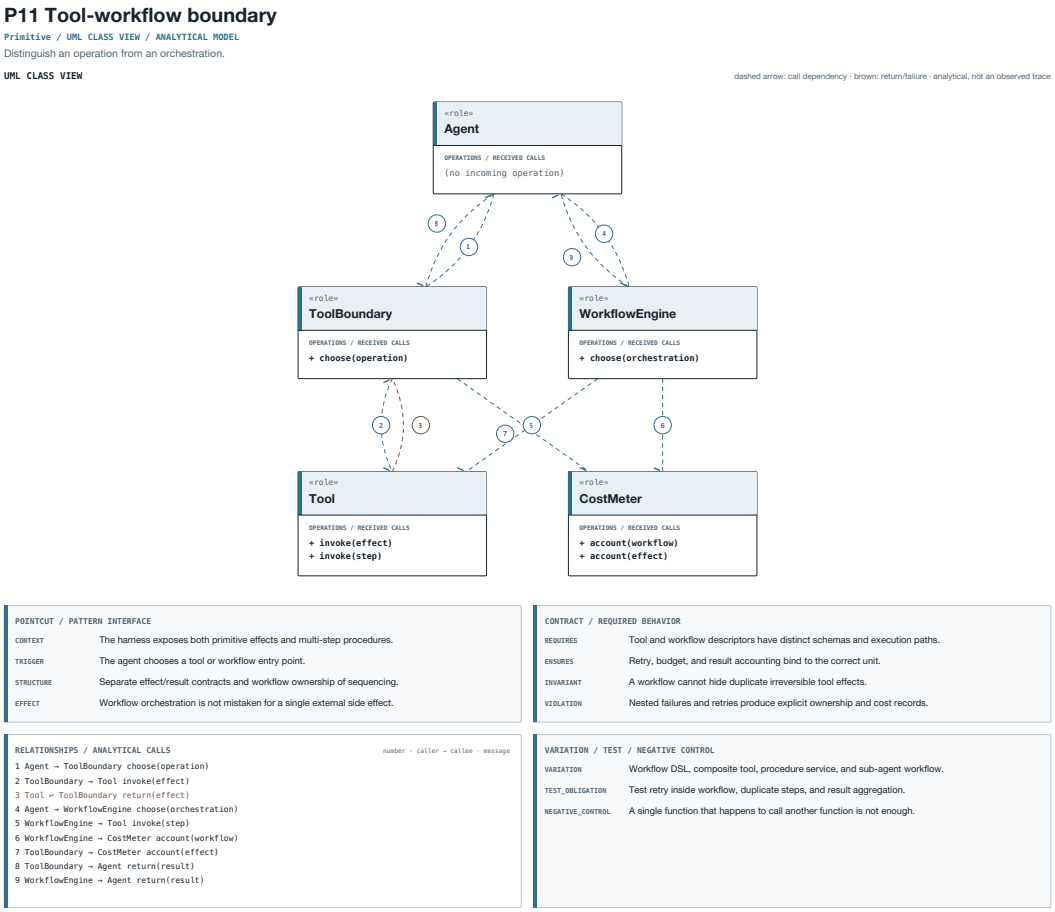


Figure 15 P11, Tool–workflow boundary: a UML class view of the proposed record. (A^2 coordinate A1.) Each role class lists the calls it receives; numbered dashed arrows are call dependencies, and brown arrows are returns or failures. The lower panels restate the pointcut, contract, variation, test obligation, negative control, and the complete numbered call list. The view is an analytical reconstruction of the proposed contract, not an observed execution or a repository class declaration.

specification.

The repository family is the independent unit for coverage, and repository-level values are descriptive. Family-level coverage is reported as a raw panel proportion. The panel is purposive and fully observed, so a proportion has no sampling uncertainty within the panel and is not generalized beyond it. Figure 6A also draws two-sided 95% Wilson score intervals; they indicate the resolution of a proportion over 36 units and are not population estimates. Observed-pattern rarefaction uses 5,000 random orders of the 36 families. Differences among record categories and source pools are screened with exploratory Kruskal–Wallis diagnostics. Because the 52 candidate summaries share one family evidence base, nominal p-values are not confirmatory; a future confirmatory analysis must preserve candidate–family dependence through permutation or hierarchical partial pooling. Co-occurrence is screened for every candidate pair with phi and a two-sided Fisher test across the 36 family indicators, followed by Benjamini–Hochberg correction over 1,326 tests. Lifecycle sequences use a repository-cluster bootstrap with 2,000 resamples (seed 20261107), because states within

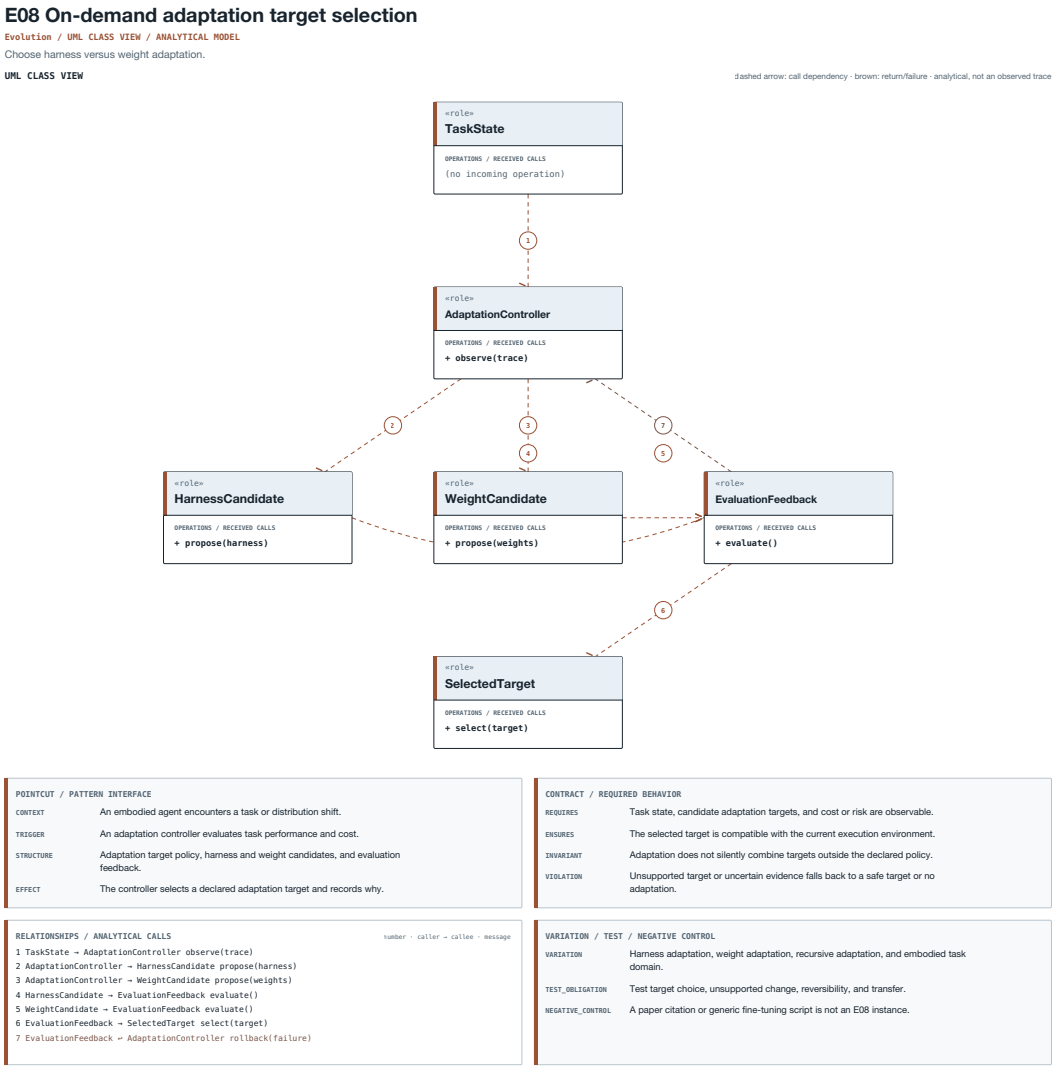


Figure 16 E08, On-demand adaptation target selection: a UML class view of the proposed record. (A^2 coordinate A3.) Each role class lists the calls it receives; numbered dashed arrows are call dependencies, and brown arrows are returns or failures. The lower panels restate the pointcut, contract, variation, test obligation, negative control, and the complete numbered call list. The view is an analytical reconstruction of the proposed contract, not an observed execution or a repository class declaration.

one realization are repeated observations, and transition probabilities are row-conditional. Because repositories of one family share history, we repeat the bootstrap with the family as the cluster and report both intervals (Table 11). A missing detector cue is recorded as a signal gap, not as retirement, unless a later source or migration record explicitly documents removal. Prompt-profile sensitivity is summarized with pairwise agreement, Fleiss' κ, Gwet's AC1, and Krippendorff's α. The profiles share one model, so these coefficients describe sensitivity to prompts and do not carry their classical independent-rater interpretation.

One held-out retrieval check covered seven predeclared candidates and reported precision and recall of 0.954 on its operational labels. Those labels, queries, and adjudications were produced by the same underlying AI system, creating construction leakage. The result is a

Table 21 Sensitivity to family aggregation. The union rule combines all repositories in a family. A one-repository-per-family sample estimates how much this aggregation changes descriptive coverage. Intervals are the 2.5th–97.5th percentiles over 5,000 seeded samples.

Measure	Family union	One repo/family
Median family richness	35.0	34.7 [33.5, 35.0]
Mean pattern-family coverage	61.8%	60.2%
Patterns present in all families	6	4.0
Repository-count/richness correlation	$r = 0.187$	

procedural consistency check, not an independently validated error model for the remaining 45 candidates or for a broader software population.

B.2 Audit Procedure and Sensitivity Analyses

B.2.0.1 Procedure.

The audit has three steps. Each record has a frozen specification with a pointcut, a contract, and a negative control (Appendix A). A frozen regular expression retrieves lexical cues from the current source, test, documentation, and configuration surface of each repository and from sampled historical releases. A curated exemplar anchors each record in one source location. A layered matcher then selects exact structural candidates, which one model codes against the specification under 200 prompt profiles (Appendix B.3). The first step is retrieval, the second is curation, and the third is model-assisted coding. None is independent human adjudication, and the panel is purposive, so every number in Section 3.3 describes this panel and this instrument. A cue is a retrieval signal, not a pattern instance, and an unobserved cell (860 of 2,132) is kept distinct from absence.

B.2.0.2 Coverage screens.

The exploratory Kruskal–Wallis screen of category coverage gave $H = 17.22$, $p = 0.00175$, and $\epsilon^2 = 0.281$ (five categories, $n = 52$ candidates). Source-pool richness did not differ under the same screen ($H = 1.28$, $p = 0.528$; three pools, $n = 36$ families). Several fully covered records have broad expressions: concurrency terms for P02, delegation terms for P12, sandbox terms for P23, scope terms for P25, and routing terms for W02. Their coverage shows that this vocabulary is common in harness code; it is not a count of pattern instances. The gradient across categories, languages, and source pools describes detector yield. It reflects detector breadth, record granularity, code layout, missing subsystems, and corpus composition as well as architecture, and is not a prevalence estimate. Activity coordinates are our assignments and are confounded with source family, so the activity contrast in Table 10 is descriptive.

B.2.0.3 Family size.

Of the 36 families, 32 contain one repository, three contain two, and one contains three. A seeded sensitivity analysis selected one repository per family: median family richness changed only from 35.0 to 34.7, but the number of candidates present in every sampled family fell from six to about four (Table 21). The repository-count and richness correlation was $r = 0.187$. Family-union coverage therefore describes the combined family surface, not a repository-average prevalence estimate.

Table 22 Sensitivity of retrieval coverage to evidence thresholds. All rows count detector or atlas evidence, not adjudicated pointcut matches. The canonical row reflects one atlas exemplar per supported pattern by construction.

Retrieval definition	Cells	Share
Any retrieval evidence	1,272	59.7%
At least one current cue	1,246	58.4%
At least two current cues	1,177	55.2%
At least five current cues	1,049	49.2%
At least one code path	1,077	50.5%
At least two code paths	932	43.7%
At least two code paths plus support	857	40.2%
Canonical atlas exemplar	51	2.4%

B.2.0.4 Evidence threshold.

Almost all observed cells rest on raw current cues, the entry level of retrieval evidence. Raising the threshold shrinks the set (Table 22); requiring at least two current code paths plus support retains 857 of 2,132 cells. Canonical and support locators give one anchor per record by construction, so they measure atlas coverage. A stricter threshold is still a retrieval rule: it removes some single-keyword hits, and how much precision it gains is unknown until cells are adjudicated.

B.2.0.5 Co-occurrence and trajectories.

All co-occurrence pairs reuse the same family matrix and detector vocabulary, so the 22 flags are exploratory. They may reflect detector vocabulary, shared architecture, or lineage, and do not identify dependency or causation. For the trajectories, resampling families instead of repositories moves the interval bounds by at most two percentage points. Of the 1,272 trajectories, 651 (51.2%) have no additional observed checkpoint, so sequence topology is not a complete lifetime reconstruction. Only 29 carry a commit-level introduction record, so the remaining first observations mark detector entry rather than a dated origin, and the 398 changed trajectories are a lower bound. The labels track cue paths, a level below adjudicated architectural redesign. Sparse checkpoints and changing detectability can produce the same picture, so the sequence counts are not statements about adoption, utility, or architectural lifecycle, nor transition probabilities for unseen repositories. The absence of an explicit retirement record does not show that no mechanism was retired.

B.3 Profile Coding of the Exact Candidates

This subsection gives the details behind the observations in Section 3.3.2. To test whether some unobserved cells are query artifacts, we applied a second matcher over Python AST symbols, multi-language declaration syntax, test, documentation, and configuration artifacts, and path context. An unrestricted first pass overmatched general words, so it was rejected. A stricter pass retained exact symbol-level regex matches or compound aliases, and it retained 16 exact structural candidates. These cells were selected by the matcher rather than sampled from the 2,132-cell matrix, so their dispositions cannot estimate precision, false-negative risk, or contract prevalence. An AI-assisted source triage prioritized eight of them as a candidate-recall overlay: one P17, two P18, four P21, and one R02 realization (Table 23).

Table 23 Matcher-selected candidates and one-model profile disposition. C/A/R denotes confirmed, ambiguous, and rejected outputs from 200 role-conditioned profiles of one model. These eight cells form a provisional candidate-recall overlay kept separate from the frozen matrix; profile-majority-supported cells enter the overlay.

Pattern	Repository	Votes C/A/R	Disposition	Structural role reviewed
P17	DeerFlow	54/146/0	Unresolved	Post-completion acceptance verdict; checker exceptions do not change completed status, and no enforced stop gate is shown
P18	Pydantic AI	200/0/0	Supported	Live repeated-call/result/text detector with nudge and terminal error
P18	Roo Code	200/0/0	Supported	Consecutive identical tool calls blocked at a configured threshold
P21	Gemini CLI	88/112/0	Unresolved	Parsed commands feed confirmation policy, but the coarse fallback remains disputed
P21	OpenClaw	105/51/44	Supported	Parsed command steps feed allowlist and execution-approval decisions; contract depth varies across profiles
P21	Qwen Code	200/0/0	Supported	AST shell safety classification feeds the permission manager
P21	OpenCode	149/51/0	Supported	Shell scan resolves permission resources before execution; all four contract fields reach majority present coding
R02	CarryOn	200/0/0	Supported	Versioned reflection memory is loaded into later recovery execution

One model evaluated all 16 selected candidates under 200 role-conditioned prompt profiles using the same frozen pointcut and contract fields. The 3,200 judgments are correlated prompt perturbations, not independent ratings or an effective sample size of 3,200. Twelve cells were invariant across profiles. Pairwise profile agreement averaged 0.882 (range 0.750–1.000); Fleiss' κ was 0.798 (cell-bootstrap 95% CI 0.633–0.946), Gwet's AC1 was 0.833, and Krippendorff's α was 0.798. The profiles were invariant on all 16 context and trigger judgments but on only 12 effect judgments. Instability therefore concerns whether a mechanism changes control flow, not whether the relevant code is present. All profiles completed all cells without missing profile–cell pairs. The 95% interval resampled the 16 cell summaries over 10,000 replicates. The agreement coefficients describe sensitivity to these prompt profiles only; their classical independent-rater interpretation does not apply. They are not estimates for the full inventory and do not substitute for human validation.

With supported and rejected dispositions requiring more than 50% of profile outputs, six cells were profile-majority supported: P18 in Pydantic AI and Roo Code, P21 in Qwen Code, OpenCode, and OpenClaw, and R02 in CarryOn. OpenCode received 149 confirmed and 51 ambiguous votes. OpenClaw crossed the majority threshold with 105 confirmed, 51 ambiguous, and 44 rejected votes. Eight cells had unanimous rejections. Two remained unresolved: P17 in DeerFlow (54 confirmed, 146 ambiguous) and P21 in Gemini CLI (88 versus 112). Five of the six supported cells recorded all four contract fields under majority

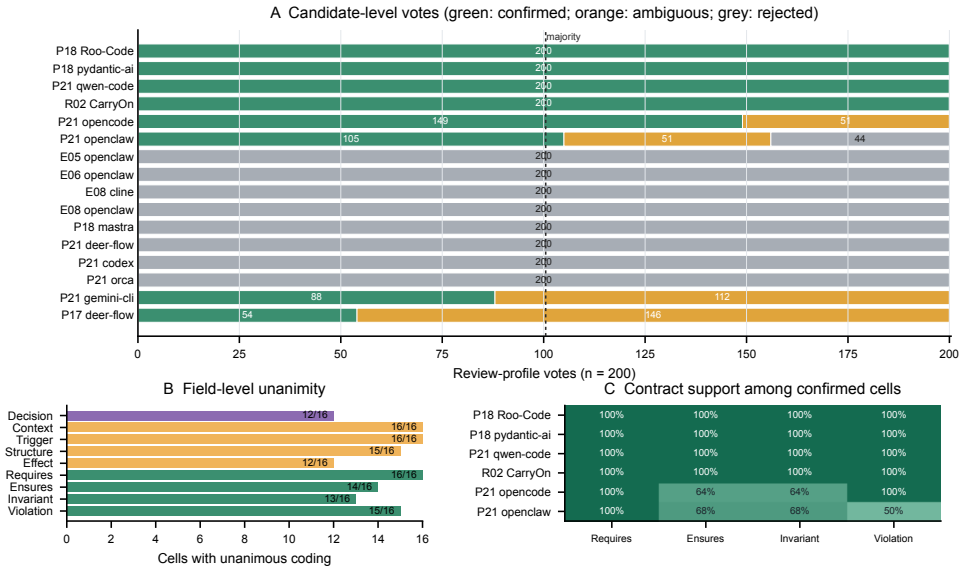


Figure 17 One-model profile sensitivity analysis of exact layered candidates. (A) Output composition for each of the 16 candidates; the dashed line marks the profile-majority threshold of 101 outputs. (B) Number of cells with invariant coding for each decision, pointcut, and contract field. (C) Percentage of profiles marking each contract field present among the six majority-supported pointcuts. Cells without a strict majority remain unresolved and are not promoted. The profiles share one model and are not human raters.

coding. Contract completeness still varies at the individual-profile level, so this does not establish reusable responsibility contracts without further review. Requiring invariance would remove OpenCode and OpenClaw from the supported set, leaving four supported, eight invariantly rejected, and four unresolved cells. The other four supported cells and all eight rejected cells are invariant. The first 100 profiles produced the same 6/8/2 pointcut dispositions, but all six supported cells were then contract-complete. Across all 200 profiles, OpenClaw's violation support fell to 49.5%, reducing contract-complete supported cells to five of six. Pairwise agreement changed from 0.892 to 0.882 and Fleiss' κ from 0.814 to 0.798. The added evidence lenses therefore changed contract support without changing the pointcut-level consensus.

The eight-cell AI-assisted set is a candidate-recall overlay, not independently confirmed coverage. Under the profile-majority rule, six supported candidates raise audited candidate coverage from 1,272 to 1,278 of 2,132 cells (59.9%), leaving 854 unobserved. This overlay does not rewrite the frozen matrix. The unresolved cells remain visible because their near-split votes concern different enforcement boundaries and no separate adjudication was recorded.

These are dispositions of one model. Unanimity under 200 prompts shows that a disposition is stable to prompt wording; it does not show that the disposition is correct, and a unanimous rejection is not a tested negative. The 16 cells illustrate stable cases and failure modes; retrieval precision and contract prevalence are questions for independently adjudicated samples.

Table 24 One-model prompt-profile sensitivity analysis of exact layered candidates. One model coded all 16 selected cells under 200 role-conditioned prompts. These correlated outputs are not human or independent raters. C/A/R denotes the model's confirmed, ambiguous, and rejected outputs. Supported and rejected dispositions require more than 50% of profiles; all others remain unresolved. Contract depth is the number of the four contract fields marked present; ranges indicate variation across review profiles.

Pattern	Repository	Votes C/A/R	Disposition	Contract	Interpretation
E05	openclaw/openclaw	0/0/200	Rejected	0/4	Declared effect or boundary absent
E06	openclaw/openclaw	0/0/200	Rejected	0/4	Declared effect or boundary absent
E08	cline/cline	0/0/200	Rejected	0/4	Declared effect or boundary absent
E08	openclaw/openclaw	0/0/200	Rejected	0/4	Declared effect or boundary absent
P17	bytedance/deer-flow	54/146/0	Unresolved	2/4 [2–3]	No absolute majority
P18	RooCodeInc/Roo-Code	200/0/0	Supported	4/4	Profile-majority point-cut match
P18	mastra-ai/mastra	0/0/200	Rejected	1/4	Declared effect or boundary absent
P18	pydantic/pydantic-ai	200/0/0	Supported	4/4	Profile-majority point-cut match
P21	QwenLM/qwen-code	200/0/0	Supported	4/4	Profile-majority point-cut match
P21	anomalyco/opencode	149/51/0	Supported	4/4 [2–4]	Profile-majority point-cut match
P21	bytedance/deer-flow	0/0/200	Rejected	0/4	Declared effect or boundary absent
P21	google-gemini/gemini-cli	88/112/0	Unresolved	2/4	No absolute majority
P21	openai/codex	0/0/200	Rejected	0/4	Declared effect or boundary absent
P21	openclaw/openclaw	105/51/44	Supported	3/4 [1–4]	Profile-majority point-cut match
P21	stablyai/orca	0/0/200	Rejected	0/4	Declared effect or boundary absent
R02	JIA-Lab-research/CarryOn	200/0/0	Supported	4/4	Profile-majority point-cut match

B.4 Detector Surface Check

An exploratory incremental scan of 17 additional source, test, and configuration extensions found 90 cue lines but no additional observed cell. The added extensions therefore did not change this detector's matrix. Pattern-specific vocabulary and structure, source-surface omissions, and genuine absence all remain candidate explanations for the unobserved cells; separating them takes adjudicated matching with known recall, which the next study supplies.

B.5 Reproducible Detection and Gated Iteration

The detector is a retrieval instrument, so its defects limit any claim drawn from cue coverage. We therefore rebuilt the cue path as a deterministic runner and audited it with a small gated loop rather than tuning the frozen matrix. The runner canonicalizes its query, codebook, and manifest inputs by SHA-256, emits cues sorted by stable identifier with no generation timestamp, and writes an observability ledger for decoded, oversized, excluded,

and unsupported files together with a truncation ledger that records matched, emitted, and omitted lines for every candidate and artifact type. Two repeats of the same input produce byte-identical outputs.

The iteration loop evaluates the immutable baseline and explicitly versioned candidate policies on labeled development fixtures. A candidate is retained only when repeated runs are byte-identical, no labeled positive loses its cue, every labeled near miss stays unmatched, and no frozen output hash changes. The holdout is scored only after selection. Table 25 summarizes the ranked defects. The fixture audit verifies three implementation defects: `checkpoint_` identifiers are missed because the frozen boundary requires a non-word character after the term; the bare `route` alternative matches configuration strings; and a documented API mentioned only in a comment yields a cue. A pattern-final underscore fix is a no-op on these fixtures because the frozen boundary already rejects the tested near miss.

Table 25 Ranked detector-audit outcomes on labeled fixtures. A candidate is the first configuration to clear every gate. Fixture precision and recall describe six development cells; they are not corpus estimates.

Candidate	Dev. precision	Dev. recall	Decision
Frozen baseline 0.1.0	0.750	0.750	reference
Boundary guards 0.2.0–0.2.2	0.750	0.750	rejected
Comment guard 0.2.3	0.750	0.750	rejected
Router forms 0.2.4	1.000	0.750	rejected
Checkpoint continuation 0.2.5	0.800	1.000	rejected
Combined 0.2.6	1.000	1.000	accepted

The accepted configuration recovers the checkpoint positive and rejects the configuration-string near miss, and it leaves the frozen detector unchanged. On three post-selection holdout cells it reached precision 0.500 and recall 1.000: it recovered the checkpoint positive and kept a prefix near miss out, but it retained the comment-only false positive. These are non-generalizing fixture checks. Applying the candidate to the frozen corpus would require a new detector version, a full re-scan, and a re-audit of every cue-derived statistic, none of which this audit performs. Independent human adjudication of the fixtures is also still open. The audit therefore documents a verified reproducibility and regression path for the instrument, not an improved evidence estimate.

B.6 Reproducibility

The companion research artifact contains the frozen repository manifests, detector outputs, pointcut specifications, realization timelines, statistical tables, figure source data, and code. Repository working trees are excluded.

C Limitations and Future Work

We consolidate here the limitations acknowledged in the main text and the future work they motivate. The four parts state what the paper actually did, what that establishes, where the evidence could be strengthened, and what should come next.

C.1 What We Did

This paper developed a position about agent-harness responsibilities, organized it through the A^2 taxonomy, and examined nine analytical cases drawn from a source-indexed inventory of

52 records. An exploratory audit searched for the records' vocabulary in 41 frozen repositories grouped into 36 families, located canonical source exemplars, and followed cue locations through sampled releases. Sixteen matcher-selected candidates were then coded by one model under 200 role-conditioned prompt profiles. A separate post-freeze calibration instrument examined scope-anchored retrieval and operational signatures. The paper also specified, but did not execute, prospective studies of record utility, external enforcement, and bounded self-modification.

C.2 What This Yields

The work yields a common language for separating a mechanism from the obligation it protects, a set of records that make those obligations inspectable, and an audit instrument that keeps retrieval, source location, structural matching, contract evidence, and causal outcomes apart. The cases show that the taxonomy can express distinctions hidden by component names, and the repository observations show that the candidate vocabulary and its changing locations can be studied in real harness code. These are conceptual, methodological, and descriptive contributions. They do not establish pattern prevalence, independent correctness of the model coding, or the practical value of a particular record schema.

C.3 How the Study Could Be Stronger

The repository panel was purposive, and its counts describe only the selected families. Lexical coverage is vulnerable to vocabulary drift, detector breadth, construction leakage, and missing source surfaces; a missing cue is not evidence of absence. The exact candidates were selected by the matcher rather than sampled from the full matrix, and the 200 profiles share one model, so their agreement measures sensitivity to prompt framing rather than independent-rater reliability. Sparse historical checkpoints and file- or interface-level anchors further limit lifecycle and structural inference. Independent human adjudication of a stratified sample, provenance-balanced repository sampling, richer cross-language structural analysis, and complete lineage-aware histories would address these limitations more directly than additional lexical hits.

C.4 Future Work

The next stage is interventional as well as observational. The H1 protocol should test whether complete records improve contract retention and design decisions over mechanism-only and ordinary documentation. The H2 protocol should compare external enforcement with capability-matched minimal and instruction-only runtimes under concealed tasks and threat models. The H3 protocol should randomize harnesses to bounded or unconstrained self-modification over repeated revision rounds and measure contract drift, regression, recovery, and attribution. Cross-model replication can test the sensitivity of source coding, but cannot substitute for independent human review or for executing these protocols. Together, these studies would show where pattern discipline earns its cost and where simpler documentation or runtime designs are sufficient.