# Harness Alignment and Harness Drift: Why Intent, Unlike Correctness, Resists Automation

**Tatsuya Shimomoto**
ORCID: 0009-0002-6168-4162

Version: v1 (2026-06-07)
Companion repository: Agent Knowledge Cycle — https://github.com/shimo4228/agent-knowledge-cycle (concept DOI: 10.5281/zenodo.19200726)

DOI: 10.5281/zenodo.20578272 (concept, all versions; v1: 10.5281/zenodo.20578273)

## Abstract

By early 2026, the discourse around agent harnesses — the configuration layer of skills, rules, prompts, and documentation shaping an LLM agent's behavior — has named two activities: harness engineering, the reactive practice of ensuring an agent never repeats a mistake, and harness optimization, autonomous search over harness code for benchmark score. Both run against fixed, checkable criteria. The activity on the other side — keeping the harness aligned with what its operator now wants, where the criterion moves — is performed daily and has, to the author's knowledge, no established name.

The paper defines harness alignment — the continuous, human-gated activity of keeping an agent's harness aligned with the operator's evolving intent — and its failure mode, harness drift. The three defining properties — continuous, human-gated, bidirectional — follow from a single root: intent, unlike correctness, cannot be automated the same way — it has no verifier outside the operator, and moves as the operator's judgment sharpens; verifying intent sharpens the judgment doing it, so the loop moves its own target. An automated check freezes intent into a specification, reducing its automatable part to correctness work. A four-domain search found no established term covering all three; an audit shows 2026 drift coinages severed from the classical lineage that harness drift bridges. A six-phase cycle (Research, Extract, Curate, Promote, Measure, Maintain) operationalizes it over a memory structure whose boundary separates freely-writable records from gated behavior-shaping artifacts.

Failure has two layers: artifact-side harness drift, and a human-side twin — gate complacency, deskilling, delegation-feedback divergence — anchored in the automation-complacency and ironies-of-automation literatures — structural inference, not measurement. Two running instances, differing in substrate, model, and knowledge genre, are offered as portability evidence, not efficacy. All of it comes from months-old practice — provisional judgments offered for testing, not settled practice.

**Keywords**: harness alignment; harness drift; intent alignment; LLM agents; agent harness; human-in-the-loop; software evolution; automation complacency

## 1. Introduction

An agent's harness can pass every correctness check and still drift from what its operator now wants. The harness — the layer around a language model that Meta-Harness defines as "the code that determines what information to store, retrieve, and present to the model" (Lee et al., 2026), and that in current practice comprises an agent's skills, rules, prompts, and documentation — has acquired its own engineering discourse, and by early 2026 that discourse has named two activities.

The first is harness engineering. Mitchell Hashimoto, writing in February 2026: "I don't know if there is a broad industry-accepted term for this yet, but I've grown to calling this 'harness engineering.' It is the idea

that anytime you find an agent makes a mistake, you take the time to engineer a solution such that the agent never makes that mistake again" (Hashimoto, 2026).[1] The testimony captures a vocabulary still crystallizing, and a practice that is reactive and correctness-driven: a mistake occurs, and the harness is changed so that the mistake cannot recur. The second is harness optimization. Meta-Harness describes "an outer-loop system that searches over harness code for LLM applications" and names the result "automated harness engineering" (Lee et al., 2026): the search is autonomous, and its criterion is a benchmark score. A unified review of the field traces "a historical progression from weights to context to harness," treating memory, skills, and protocols as "three distinct but coupled forms of externalization" with harness engineering as the unification layer that coordinates them (Zhou et al., 2026).[2]

Both named activities sit on the same side of harness work: the automatable side. Each runs against a criterion that is fixed and checkable — a failing test, a benchmark score. The activity on the other side — keeping the harness aligned with what its operator now wants, where the criterion itself moves — is performed daily wherever an operator maintains an agent's skills and rules over months, and it has, to the author's knowledge, no established name in the literature.

Harness alignment — the continuous, human-gated activity of keeping an agent's harness aligned with the operator's evolving intent — is this paper's subject. The term is in operational use in the Agent Knowledge Cycle (AKC) repository, where it was established by a recorded design decision (Shimomoto, 2026a, ADR-0017); the paper's task is to define it against the literature and to argue for the distinction it marks. The thesis: harness alignment is not a variant of harness engineering or harness optimization. Those automate against fixed, checkable criteria, while intent alignment cannot be automated the same way — intent has no verifier outside the operator, and it moves as the operator's judgment sharpens — so the activity is necessarily continuous, human-gated, and bidirectional. A six-phase cycle operationalizes it today in two independent instances. Its failure mode, harness drift, names what accumulates when the activity stops.

Two scoping notes bound what follows. First, the practice is young. The discourse cited above is measured in months — Hashimoto in February 2026, Meta-Harness in March, the externalization survey in April — and the daily work it names, operating an agent's harness seriously enough that its alignment becomes a maintenance problem, is about as recent. No one yet holds established best practices for this layer — Hashimoto's hedge above, not knowing whether an industry-accepted term exists, is the condition of the layer itself — and what any operator holds today, the author included, are provisional judgments formed while the ground moves.

Second, the paper's evidential base is one such set of judgments. The AKC repository is a single operator's public, dated decision record: architecture decision records (ADRs) stating what was decided, when, and why. The argument is offered as a position, not an empirical study. ADR citations in what follows locate where a provisional judgment was recorded and what it responded to; they do not assert that the judgment is settled. The paper's claim is that even this early, a structure is visible — stable across months of one practice, and unnamed in the literature.

The gate this paper argues for was not arrived at from caution: the judgments behind it did not come from holding back on automation. The practice behind them automated past the defaults of the harness it runs on — autonomous extraction and distillation of session knowledge, scheduled batch runs, and, in the second of the two instances presented in Section 5, an agent that runs the cycle over its own logs — and

[1] Hashimoto names two concrete forms the practice takes: "[b]etter implicit prompting (AGENTS.md)" and "[a]ctual, programmed tools. For example, scripts to take screenshots, run filtered tests, etc etc" (Hashimoto, 2026).

[2] The survey is explicit that harness engineering "is not a fourth kind of externalization alongside memory, skills, and protocols. It is the runtime environment within which these forms of externalization operate and interact" (Zhou et al., 2026).

the human gate marks where pushing automation further cost more than it returned (Shimomoto, 2026a, ADR-0005, ADR-0010).[3] The non-automation claims of Section 3 are a boundary found by overshooting it, not a conservatism that never tested it. The same provenance holds for the companion paper cited in Section 3 — an accountability argument that, where responsibility cannot be redirected, a human bearer must be named in advance: that framework, too, was extracted from the operational history of an agent built to run autonomously (Shimomoto, 2026c).

The asymmetry in the existing vocabulary is, on this paper's reading, not an accident. Activities acquire names when they acquire tooling and measurements: a practice that runs against a checkable criterion can be scripted, benchmarked, productized — and therefore discussed. The remainder that resists that automation stays practiced but unnamed. On this reading, the naming gap the paper addresses is an instance of its own thesis: what was named first is what could be automated.

The paper proceeds as follows. Section 2 defines harness alignment, introduces its three properties and its failure mode, and states what the activity is not. Section 3 argues why the three properties follow from a single root — the part of harness work that cannot be automated the way correctness can. Section 4 locates the absence: each component of the definition is already named in prior literature, but no single established term covers all three properties at once. Section 5 presents the cycle that operationalizes the activity, the memory boundary it rests on, and the two instances in which it runs; Section 6 presents its failure modes on two layers. Section 7 relates the cycle to the agent-memory literature, and Section 8 concludes with open questions.

## 2. Definition: what harness alignment is, and is not

### 2.1 The definition and its three properties

Harness alignment is the continuous, human-gated activity of keeping an agent's harness — its configuration layer of skills, rules, prompts, and documentation — aligned with the operator's evolving intent (Shimomoto, 2026a, ADR-0017). Three properties define the activity, and all three are introduced here because Section 3 derives them jointly:

- **(a) The alignment target is operator intent, itself evolving.** The target is not a specification, a test suite, or a benchmark; it is what the operator currently wants from the agent. And the target moves: the operator's judgment about what good agent behavior looks like sharpens through use, so the intent the harness must track this month is not the intent it tracked last month. Because the loop's own running is what moves the target (Section 3), this paper labels the property *bidirectional*.[4]
- **(b) The loop is human-gated.** Every change to the artifacts that shape future behavior passes through an explicit human approval gate before taking effect: the proposing system produces a diff or proposal and stops, and the operator reviews, edits if needed, and commits (Shimomoto, 2026a, ADR-0005). Nothing is auto-applied.

[3] Concretely: in early 2026 the harness's default posture asked for permission on nearly every tool invocation — an automatic mode arrived later — and the author ran it with those approvals bypassed, retaining confirmation only for a handful of irreversible operations. The gate of Section 5 is what that subtraction left visible: with correctness-level approvals removed, the writes still worth a human's review were the ones that shape future behavior. The configuration practice — the allowlist, and a block list kept to a handful of irreversible operations — is documented in a contemporaneous write-up (Shimomoto, 2026d).

[4] An example small enough to see whole, from the drafting of this paper (June 2026). Reviewing a draft, the author noticed that footnotes could carry a second, denser layer for LLM readers while the body stays lean for humans — a new preference about what a paper should be, formed mid-read. The author's harness holds a rules file governing how papers are drafted; left unchanged, it would from that moment have been misaligned with the operator's intent — not wrong, just no longer what the operator wants. It was updated the same day, through explicit operator instruction and review. The first version of the new rule then mis-guessed a detail — where footnote definitions should sit — and was corrected by the operator hours later: intent has no verifier outside the operator, even, and especially, when the agent's guess is plausible.

- **(c) The activity is continuous.** Alignment is sustained through a recurring cycle, not configured once.[5]

The term extends intent alignment, which originates with Christiano (2018): "When I say an AI A is aligned with an operator H, I mean: A is trying to do what H wants it to do."[6] Christiano separates this from correctness in the same source — alignment is "the problem of getting your AI to try to do the right thing, not the problem of figuring out which thing is right." Harness alignment extends the notion in two directions: from the agent's *behavior* to the *artifacts that shape behavior*, and across *time* — from a property an agent has at a moment to an activity an operator performs over months. One disclaimer bounds the term: the alignment target is operator intent, not model values. This is not an AI-safety value-alignment claim.[7]

The activity has a named failure mode. Harness drift is the gradual uncoupling of the harness from operator intent when the alignment activity does not run: skills go stale, rules stop matching practice, documentation diverges from code (Shimomoto, 2026a, ADR-0017). Section 6 develops it in lineage with the drift vocabularies of classical software engineering and of 2026 agent research.

### 2.2 What harness alignment is not

Harness engineering is reactive and lives on the correctness axis: a mistake occurs, and the harness is changed — through better prompting or programmed tooling — so that the agent never repeats it (Hashimoto, 2026). Harness optimization is autonomous and lives on the score axis: an outer loop searches over harness code to maximize a benchmark score (Lee et al., 2026). Harness alignment is neither, and the difference is not tempo or tooling but the nature of the criterion.

Harness engineering runs against a fixed, checkable criterion — the mistake that must never recur — which is what makes the practice scriptable. Harness optimization runs against another — the benchmark — which is what makes the search automatable. Harness alignment's criterion is operator intent, which lives in the operator and moves.[8] Engineering and optimization belong to the correctness axis; harness alignment occupies the intent axis. The axes are complementary, not competing: a harness can be optimized and drifted simultaneously — scoring better on a fixed benchmark while sliding away from what its operator now wants (Shimomoto, 2026a, ADR-0017).

Section 3 takes this criterion contrast as its root and derives the three properties from it.

## 3. Why these three properties: the part that resists automation

Harness engineering and harness optimization automate against fixed, checkable criteria — a failing test, a benchmark score. Intent alignment — alignment whose criterion is what an operator wants — cannot be automated the same way, because intent has no verifier outside the operator and moves as the operator's judgment sharpens through use. The companion repository carries this as a standing claim: correctness can be automated — "tests, types, linters, and review tools all check whether an output passes specific criteria" — while alignment "cannot be automated to the same degree, because intent itself moves as the operator's judgment sharpens through use" (Shimomoto, 2026a) — a comparative hedge, not an impossibility theorem. Christiano draws the same line: "I use alignment as a statement about the motives of the assistant, not

[5] *Continuous* is this paper's fixed label for property (c). Lehman's phrase "continual change" appears in Section 3 as his own term, with a different driver.

[6] The coinage is explicit in the source: "When precision is needed, I'll refer to this property as 'intent alignment.'" (Christiano, 2018).

[7] The remaining risk is recorded on the repository side as well: the word collides with AI-safety vocabulary, and even with the disclaimer "the risk of out-of-context quotation remains" (Shimomoto, 2026a, ADR-0017).

[8] The companion repository's layer comparison puts the contrast as a question pair: the harness layer asks "[i]s this output correct?"; the alignment activity asks whether the harnesses themselves are still valid (Shimomoto, 2026a).

about their knowledge or ability" (2018) — an output check inspects the output, not the trying.[9]

A structural argument — this paper's own, not a proof — says why the asymmetry is no artifact of current tooling. Any automated intent-check would have to freeze intent into a stated criterion; a frozen criterion is a specification, and checking against a specification is correctness work. On this argument, the automatable part of intent alignment reduces, piece by piece, to correctness work — and what remains is the moving criterion itself, exactly the part that resists the same automation. The residue is not zero work; it is the activity this paper names, and the three properties are its shape. The derivation runs (c), (b), (a), from the most mechanical consequence outward.

**Continuous — property (c).** Because the criterion moves, no one-time configuration stays aligned: the correction must run as long as the intent does. Lehman's first law of software evolution is the structural parallel: a used program "undergoes continual change or becomes progressively less useful" (Lehman, 1980, Law I).[10] "Continual change" is Lehman's term, and his driver is the world the program models; here it is the operator's intent, which moves even when the world does not.

**Human-gated — property (b).** Because the moving criterion lives only in the operator, every change to behavior-shaping artifacts passes a human gate: the gate is where intent enters the loop. The gate also carries a verification argument — the operator is the one verifier whose failures are not correlated with the generator's; an argument, not a formal proof (Shimomoto, 2026a, ADR-0005).[11] In Liu and van der Schaar's taxonomy (2025), such a gate is what they "term extrinsic metacognition" — a human evaluator at the decision point — extrinsic here by design, not by immaturity: automating the evaluator would re-import the freezing move.[12]

The same remainder is reached on a second, structurally distinct ground — an accountability argument from the same practice. *Distributing Accountability, Not Capability* (Shimomoto, 2026c; DOI 10.5281/zenodo.20353789) asks not whether intent has a verifier but whether consequences have a bearer, and arrives at the same location: capability distribution does not produce accountability distribution, and where responsibility cannot be redirected in principle, that framework's answer is a human bearer "named before deployment rather than discovered after an incident."[13] An epistemic axis and a normative axis leave the same un-automatable remainder: a human named in advance, built into the structure where behavior-shaping writes occur.

**Bidirectional — property (a), the evolving target.** Because verifying intent is an act of judgment, the verifying changes the verifier: deciding what is worth keeping, what deserves to become a rule, and whether the agent followed it sharpens — and so moves — the judgment doing the verifying. The loop's running changes its own target. Christiano's base definition treats the operator's wants as static (2018); the companion repository records the dynamics as a first-class theme — the cycle changes the human too (Shimomoto, 2026a, ADR-0009, ADR-0012).

---

[9] The definition is explicitly about the trying, not the outcome: an aligned agent can still make errors, and fixing them is outside the definition (Christiano, 2018).

[10] Lehman's mechanism for the law: "The program has become a part of the world it models, it is embedded in it" (Lehman, 1980).

[11] The repository's form of the argument: "[w]hen the entity proposing a change and the entity checking it are the same system, the check inherits the proposer's blind spots — the generator–verifier gap" (Shimomoto, 2026a, ADR-0005).

[12] The contrast class is defined in the same source: "Intrinsic metacognitive learning, then, occurs when agents independently assess their learning, update metacognitive knowledge, and adapt learning plans to optimize long-term performance without relying on external mechanisms" (Liu & van der Schaar, 2025).

[13] To risk-management and management-system standards, that framework positions itself as supplying "the judgment layer they presuppose" — the per-deployment decisions required before such controls become applicable (Shimomoto, 2026c).

These derivations support the definition, not replace it; whether an existing term already names the intersection is Section 4′s question.

## 4. No single established term: locating the absence

Each component of harness alignment is already named in prior literature; the absence is at the intersection. Before the term was established, a four-domain literature search — software evolution; information-systems, organizational, and safety science; machine learning and AI safety; HCI and knowledge engineering — was run on 2026-06-06 and is recorded in the companion repository (Shimomoto, 2026a, ADR-0017). It found, to the author's knowledge, no single established term that simultaneously covers the three properties: an evolving operator-intent target, a human-gated loop, and continuous alignment.[14] A counter-example sweep re-running the search against newer literature precedes deposit of this paper. The nearest candidates and their gaps:

| Term | Source | What it names | What it misses |
|---|---|---|---|
| Intent alignment | Christiano (2018) | An agent "trying to do what H wants it to do" | Intent treated as static; no configuration layer |
| Continual change (Law I) | Lehman (1980) | Continual change as intrinsic and feedback-driven | Driver is the world, not operator intent |
| Architectural drift | Perry & Wolf (1992) | Divergence "due to insensitivity about the architecture" | The intended architecture is assumed fixed |
| Practical drift | Snook (2000), as characterized in secondary literature | Practice uncoupling from written procedure | Names the failure process, not the counter-activity |
| Harness optimization | Lee et al. (2026) | Autonomous search over harness code | Score-driven; no operator intent in the loop |
| Agent drift | Rath (2026) | Behavioral-level degradation (developed in Section 6) | Names a failure, not an activity; no classical lineage |
| Extrinsic metacognition | Liu & van der Schaar (2025) | The human evaluator's role in an agent's learning loop | Names the evaluator's role, not a continuous intent-alignment activity on the harness |

[14] Two further candidates were rejected on different grounds: "alignment drift" circulates informally but has no canonical citation to build a reference relation on, and "configuration drift" is DevOps vendor vocabulary whose nearest research neighbor — code-documentation inconsistency detection — covers documentation only and carries no intent-evolution dimension (Shimomoto, 2026a, ADR-0017).

[15] A pre-deposit sweep (June 2026) checked the names themselves. "Harness drift" appears once outside this program, in a different sense: a benchmark-disclosure audit uses it for results produced on the same benchmark under different scaffolds circulating under the same name (Moghadasi & Ghaderi, 2026) — a cross-paper comparability defect, not a configuration layer's uncoupling from operator intent. No non-author use of "harness alignment" as a defined term was found. The nearest neighbors by mechanism rather than by name: DoubleAgents, a system whose coordination policies are human-approved and continuously revised — the three properties, mechanically, but as a named system for one socially embedded task, not a named

Harness alignment names that intersection, and harness drift names its failure mode.[15] A nameable intersection is not yet a runnable one — Section 5 shows the activity running, Section 6 how it fails, and Section 7 where it sits in prior art.

## 5. Operationalization: the cycle and its instances

Section 3′s argument has a physical address in the implementation. The companion repository operationalizes harness alignment as a six-phase cycle whose load-bearing element is a single structural commitment: a memory boundary separating freely-writable records from gated behavior-shaping artifacts (Shimomoto, 2026a). This section presents the cycle, the boundary, the attention cost they impose, and the two running instances.

The cycle exists because of property (c): a moving criterion needs a recurring loop, not a one-time configuration. Its six phases divide into three that add or elevate knowledge — Research (signal-first intake: admit only information that would change the next action), Extract (capture reusable patterns from sessions), and Promote (elevate recurring patterns to always-loaded rules, through the gate)[16] — and three that audit and keep it honest — Curate (check for redundancy, staleness, and silence), Measure (check compliance observably rather than by impression), and Maintain (keep documentation roles clean and content fresh). The phases close into a feedback loop — Measure and Maintain detect harness drift; Curate and Promote correct it through the gate (Shimomoto, 2026a) — the structure Lehman said evolution intrinsically has: "evolution is an intrinsic, feedback driven, property of software" (Lehman, 1980).

The boundary is where property (b) becomes architecture. Appending to the episode log (the raw record of sessions) and updating the knowledge store derived from it require no approval — records are disposable and reproducible.[17] Adding or editing a rule, a skill, or an identity document (the agent's persistent self-description)[18] — anything that shapes future behavior — requires explicit human sign-off, with no auto-approve-after-N-days path and no approved-by-the-LLM-itself path, because behavior-shaping artifacts also shape what the agent later extracts and promotes from its own records: an unchecked bad change there is "a seed, not a one-off error" (Shimomoto, 2026a, ADR-0005).[19] What can be verified without the operator runs unattended; every change that shapes behavior passes the gate, and intent enters the loop with it.

The gate's work — reviewing and committing every behavior-shaping write — is the part Section 3 argued cannot be automated the same way, so it is paid for in operator attention, which does not scale with the model — a claim about 2026-era human-agent work, not a timeless law (Shimomoto, 2026a, ADR-0010). The constraint disciplines where the cycle spends judgment,[20] and it returns in Section 6 as the amplifier of the gate's own failure mode.

The cycle runs today in two independent instances: the author's daily operation of a commercial coding harness (Anthropic's Claude Code), and Contemplative Agent, a CLI agent that "runs a six-phase knowledge

---

activity over the agent's own configuration layer (Long et al., 2025); and bidirectional human-AI alignment, which names mutual human-AI adaptation as a framework but does not localize a gate on behavior-shaping writes (Shen et al., 2024).

[16] Promote is the load-bearing move within the loop: rules are loaded every session and shape behavior reliably, where skills trigger probabilistically — "Promotion moves knowledge from the probabilistic layer to the deterministic layer" (Shimomoto, 2026a, docs/akc-cycle.md).

[17] In the repository's terms, the episode log is write-through and immutable, the knowledge store is a pure function of it, and regenerating both from scratch is a supported recovery path (Shimomoto, 2026a, ADR-0005).

[18] The identity document is not an exotic artifact: agent frameworks ship it as a persona file — OpenClaw, for instance, loads SOUL.md, "[p]ersona, tone, and boundaries," at the start of every session (OpenClaw, 2026).

[19] Approval also leaves an audit trail: the version history of the gated artifacts is a record of every intentional behavioral change. For batch or scheduled runs, the gate's output is "a patch file or a pull request — never a direct write" (Shimomoto, 2026a, ADR-0005).

[20] Concretely: signal-first intake bounds what is read, promotion to rules prevents the same judgment from being re-made every session, and compliance measurement replaces manual re-audit (Shimomoto, 2026a, ADR-0010).

cycle over its own logs — every promotion from logs → patterns → skills → rules passes through a human approval gate" (Shimomoto, 2026b).[21] The first instance is self-attested — the author is also the operator, and the repository that defines the cycle records its own operation (Shimomoto, 2026a, ADR-0018). The two are offered as evidence of portability — the same six phases and the same gate surviving a change of substrate (plain-Markdown rules versus a code re-implementation),[22] model, and knowledge genre — not as an efficacy evaluation; the demonstration is ongoing.

A cycle that can run can also fail to run. Section 6 names what accumulates when it does not — on two layers.

## 6. Failure modes: harness drift, and its human twin

Harness alignment fails on two layers that compound but are distinct: on the artifact side, the harness uncouples from intent; on the human side, the operator's part of the loop degrades. This section treats them in turn and keeps them separate.

### 6.1 The artifact side: harness drift

Harness drift is the gradual uncoupling of the harness from operator intent when the cycle does not run: skills go stale, rules stop matching practice, documentation diverges from code (Shimomoto, 2026a, ADR-0017). The name stands in explicit lineage with three drift vocabularies.

In classical software architecture, Perry and Wolf (1992) name architectural drift — divergence "due to insensitivity about the architecture" — and distinguish it from architectural erosion, which is "due to violations of the architecture": drift comes from inattention, erosion from violation.[23] Harness drift is drift in precisely this sense; no one violates the harness — it uncouples because no one is looking. In safety science, practical drift — the slow, steady uncoupling of practice from written procedure, as characterized in secondary literature (Snook, 2000) — describes the same shape at the level of organizations and their procedures. In 2026 agent research, Rath (2026) coins agent drift: "the progressive degradation of agent behavior, decision quality, and inter-agent coherence over extended interaction sequences," introduced with three manifestations — semantic drift ("progressive deviation from original intent"), coordination drift ("breakdown in multi-agent consensus mechanisms"), and behavioral drift ("emergence of unintended strategies"). Agent drift names degradation at the behavior level; harness drift names the uncoupling at the configuration layer that shapes behavior.

A bibliographic observation connects the lineages. The agent-drift paper's reference list — twelve entries, audited directly — contains no classical software-evolution literature at all: no Lehman, no Perry and Wolf, no Parnas, no Snook (Rath, 2026). A pre-deposit audit extended the check to three further drift coinages

[21] The re-implementation maps the cycle's phases onto code rather than installing its Markdown artifacts, with "[n]o fine-tuning, no labeled training data" (Shimomoto, 2026b). The relationship also runs the other way: several of the cycle's early decision records, the gate among them, were adapted from Contemplative Agent's engineering history (Shimomoto, 2026a, ADR-0018).

[22] The plain-text form is itself doing design work: the cycle ships as Markdown rules installable with a single file copy, which keeps the entire behavioral surface reviewable by the person who must gate it and portable to any harness that reads plain text. The skills and rules that teach the cycle are also scaffolding designed to dissolve — as operator and agent internalize it, the explicit files become unnecessary and the loop runs through conversation alone: "Success is measured not by rule count, but by whether the cycle runs naturally without explicit invocation" (Shimomoto, 2026a, docs/akc-cycle.md). The scaffolding is person-dependent — a new operator-agent pair needs it again — and that is property (a) made visible: what remains when the scaffold fades is the judgment the loop sharpened.

[23] The other half of Perry and Wolf's pair: architectural erosion's violations "often lead to an increase in problems in the system and contribute to the increasing brittleness of a system — for example, removing load-bearing walls often leads to disastrous results" (Perry & Wolf, 1992).

— constraint drift, memory drift, belief deviation — and found the same absence.[24] Four audited lists are consistent with, though do not by themselves establish, a broader disconnection between the 2026 agent-drift coinage and the classical drift lineage. Harness drift, defined with explicit lineage to both bodies, bridges that gap by reference rather than by reinvention.

### 6.2 The human side: the failure twin

The bidirectional property has an honest twin: a loop that can sharpen judgment can also erode it. The companion repository records this failure twin as three human-side modes, held as structural inference — properties of a coupled human-agent loop, not measured phenomena — under an explicitly experimental status (Shimomoto, 2026a, ADR-0014). Each mode is presented below at that same claim strength, and located in the empirical literature where it is already documented.

**Gate complacency.** A stream of well-formed, usually-correct proposals trains the operator to approve by default: the click still happens, but the judgment behind it thins to a reflex. This is an instance of automation complacency, operationally defined as "poorer detection of system malfunctions under automation control compared with manual control" (Parasuraman & Manzey, 2010). Two findings from that literature map directly onto the gate. First, complacency is not passive laziness but "an active reallocation of attention away from the automation to other manual tasks in cases of high workload" — the busier the operator, the stronger the pull. Second, it is reliability-dependent: in the studies Parasuraman and Manzey integrate, operators monitoring constant-reliability automation detected 33% of its failures where operators of variable-reliability automation detected 82%, and detection varied inversely with reliability.[25] A consistently good agent is precisely the condition under which the gate's quality decays.[26] The attention constraint that Section 5 priced as the cycle's running cost is also this mode's amplifier: "Under a constrained attention budget, the cheapest action is to trust the proposal, and a trustworthy agent makes that cheap action feel safe" (Shimomoto, 2026a, ADR-0014).

**Deskilling.** A human who only reviews the agent's output lets the faculty that review depends on atrophy. This is Bainbridge's (1983) irony of automation: "physical skills deteriorate when they are not used," so "a formerly experienced operator who has been monitoring an automated process may now be an inexperienced one" — while the monitoring arrangement itself asks the awkward thing: "the automatic control system has been put in because it can do the job better than the operator, but yet the operator is being asked to monitor that it is working effectively."[27]

**Delegation-feedback divergence.** The agent keeps acting while the feedback that would correct it no longer reaches a human positioned to use it. This is the most dangerous mode because, unlike a stalled

[24] Each audited through two independent channels (the Semantic Scholar reference graph and the arXiv HTML reference list): constraint drift — "the loss, distortion, weakening, or relaxation of constraints as they pass through memory, delegation, communication, tool use, audit, and optimization" (Li et al., 2026); memory drift, with semantic, procedural, and goal subforms (Lam et al., 2026); and belief deviation over long horizons (Liu et al., 2026). None cites Lehman, Perry and Wolf, Parnas, or Snook. Their pre-2000 references reach control theory, cognitive psychology, and information-flow security — the papers do reach back historically, never into software evolution. The second paper cites the agent-drift coining paper itself: the drift vocabulary is propagating within the 2026 agent literature while remaining severed from the classical lineage.

[25] Parasuraman and Manzey (2010) note the economics: "Given that highly reliable automation will fail only very rarely, then the rational strategy would be to monitor it also only very infrequently" — complacency at a reliable gate is not negligence but a rational reallocation, which is why the defense has to be structural.

[26] The adjacent concept, automation bias — omission and commission errors that follow automated advice — is distinct: complacency concerns monitoring, bias concerns decision-following (Parasuraman & Manzey, 2010).

[27] Bainbridge sets the bounds: vigilance — "it is impossible for even a highly motivated human being to maintain effective visual attention towards a source of information on which very little happens, for more than about half an hour" — and the take-over paradox: "the operator needs to be more rather than less skilled, and less rather than more loaded, than average" (Bainbridge, 1983).

loop, a diverged loop still produces output — output no human is meaningfully steering (Shimomoto, 2026a, ADR-0014).

The empirical citations locate the first two modes as agent-knowledge instances of phenomena the automation literature has documented for decades; they do not show the modes have been measured on this cycle. The claim stays bounded — the record "does not claim AKC prevents these failures" (Shimomoto, 2026a, ADR-0014) — and the defenses are structural rather than exhortative. Three structures resist the modes: the gate is a circuit-breaker, not a delay — no auto-approve path exists for the loop to route around, so divergence has an arrest point; Curate and Promote are active judgment acts rather than passive assent, so the cycle's normal operation exercises exactly the faculty deskilling erodes; and the gate forces articulation — a diff reviewed, possibly edited, and committed under the operator's name — which is structurally heavier than a yes/no click.

The two layers compound — a complacent gate accelerates harness drift — but they are not the same failure, and they are recorded separately (Shimomoto, 2026a, ADR-0017). A cycle whose individual operations sound familiar invites the prior-art question. Section 7 answers it.

## 7. Relationship to existing frameworks

Run as isolated operations, the cycle's phases have named precedent. Skill induction: Voyager builds an "ever-growing skill library of executable code" (Wang et al., 2023), and Agent Workflow Memory induces "commonly reused routines, i.e., workflows" (Wang et al., 2024) — analogues of Extract and Promote, but closed autonomously, without a human gate. Procedural-memory refinement: ReMe is a "dynamic procedural memory framework" (Cao et al., 2025), and LangMem lets developers "control what gets stored" and optimize agent behavior through "prompt refinement" (LangChain, 2025) — analogues of Curate, but run platform-side.[28] Reflection: Generative Agents "synthesize those memories over time into higher-level reflections" (Park et al., 2023). Memory hierarchies: MemGPT provides "the appearance of large memory resources through data movement between fast and slow memory" (Packer et al., 2023). Framework vocabulary: CoALA (Sumers et al., 2023) and the externalization survey's map of memory, skills, and protocols unified under harness engineering (Zhou et al., 2026).

The delta is not a new operation but the activity those operations compose into: a loop that is human-gated where the prior art closes autonomously, that targets the operator's judgment where the prior art optimizes the agent or its context, and that binds on operator attention where the prior art binds on context, consistency, or capability (Shimomoto, 2026a, ADR-0013).[29] These works were identified as prior art for positioning, not consulted during the cycle's construction (Shimomoto, 2026a, ADR-0013).

## 8. Conclusion

Harness alignment — the continuous, human-gated activity of keeping an agent's harness aligned with the operator's evolving intent — is not a variant of harness engineering or harness optimization. Those automate against fixed, checkable criteria; intent alignment cannot be automated the same way, because intent has no verifier outside the operator and moves as the operator's judgment sharpens through use. The three defining properties are the shape of that residue, a six-phase cycle operationalizes the activity in two running instances, and harness drift names what accumulates when the cycle stops.

Four open questions bound the contribution. First, external adoption of both terms is n=0 at publication: they may remain internal shorthand with citations, never adopted externally as the bridge Section 6

[28] ReMe's own contrast term for what it moves beyond is a "static append-only archive" of experience (Cao et al., 2025).

[29] The companion repository's positioning record states the delta plainly: in the agent-memory literature "the human is, at most, an annotator" (Shimomoto, 2026a, ADR-0013).

proposes between the agent-drift coinage and the classical lineage (Shimomoto, 2026a, ADR-0017). Second, the human-side failure twin is a structural inference awaiting measurement: Section 6′s empirical anchors locate the modes in established literatures but do not measure them on this cycle (ADR-0014). Third, whose attention budget: the cycle is designed from the primary operator's perspective, and generalization to multi-operator settings is unresolved (ADR-0010). Fourth, the 2026 sources that carry the contrast — Meta-Harness and agent drift — are unreviewed preprints whose vocabulary may shift, dating the contrast; the classical anchors, Lehman and Perry and Wolf, carry the durable weight.

Every claim above carries the scope set in Section 1: a structure named from inside a months-old practice, on the record of a single operator, offered for other operators to test against their own. The vocabulary asymmetry the paper began from is likely to persist: activities that can be scored will keep acquiring names, tools, and benchmarks faster than activities whose criterion lives in a person. Harness alignment names what is on the other side of that line — an activity already performed daily, wherever an agent's harness is kept aligned with an operator whose intent keeps moving because their judgment keeps sharpening.

## References


Bainbridge, L. (1983). Ironies of Automation. *Automatica* 19(6): 775–779. DOI: 10.1016/0005-1098(83)90046-8

Cao, Z., et al. (2025). *Remember Me, Refine Me: A Dynamic Procedural Memory Framework for Experience-Driven Agent Evolution.* arXiv:2512.10696.

Christiano, P. (2018). *Clarifying "AI alignment".* AI Alignment (ai-alignment.com). https://ai-alignment.com/clarifying-ai-alignment-cec47cd69dd6 (archived: https://web.archive.org/web/20250516125926/https://ai-alignment.com/clarifying-ai-alignment-cec47cd69dd6)

Hashimoto, M. (2026). *My AI Adoption Journey.* mitchellh.com. https://mitchellh.com/writing/my-ai-adoption-journey

Lam, C., et al. (2026). *Governing Evolving Memory in LLM Agents: Risks, Mechanisms, and the Stability and Safety Governed Memory (SSGM) Framework.* arXiv:2603.11768.

LangChain (2025). *LangMem.* https://langchain-ai.github.io/langmem/

Lee, Y., Nair, R., Zhang, Q., Lee, K., Khattab, O., & Finn, C. (2026). *Meta-Harness: End-to-End Optimization of Model Harnesses.* arXiv:2603.28052.

Lehman, M. M. (1980). Programs, Life Cycles, and Laws of Software Evolution. *Proceedings of the IEEE* 68(9): 1060–1076. DOI: 10.1109/PROC.1980.11805

Li, T., et al. (2026). *Safe Multi-Agent Behavior Must Be Maintained, Not Merely Asserted: Constraint Drift in LLM-Based Multi-Agent Systems.* arXiv:2605.10481.

Liu, T., & van der Schaar, M. (2025). Position: Truly Self-Improving Agents Require Intrinsic Metacognitive Learning. In *Proceedings of the 42nd International Conference on Machine Learning* (PMLR 267). arXiv:2506.05109.

Liu, Z., et al. (2026). *Meta-Cognitive Memory Policy Optimization for Long-Horizon LLM Agents.* arXiv:2605.30159.

Long, T., et al. (2025). *DoubleAgents: Human-Agent Alignment in a Socially Embedded Workflow.* arXiv:2509.12626.

Moghadasi, M. N., & Ghaderi, F. (2026). *What Twelve LLM Agent Benchmark Papers Disclose About Themselves: A Pilot Audit and an Open Scoring Schema.* arXiv:2605.21404.

OpenClaw (2026). *Agent workspace.* OpenClaw documentation. https://docs.openclaw.ai/concepts/agent-workspace

Packer, C., et al. (2023). *MemGPT: Towards LLMs as Operating Systems.* arXiv:2310.08560.

Parasuraman, R., & Manzey, D. H. (2010). Complacency and Bias in Human Use of Automation: An Attentional Integration. *Human Factors* 52(3): 381–410. DOI: 10.1177/0018720810376055

Park, J. S., et al. (2023). *Generative Agents: Interactive Simulacra of Human Behavior.* arXiv:2304.03442.

Perry, D. E., & Wolf, A. L. (1992). Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes* 17(4): 40–52. DOI: 10.1145/141874.141884

Rath, A. (2026). *Agent Drift: Quantifying Behavioral Degradation in Multi-Agent LLM Systems Over Extended Interactions.* arXiv:2601.04170.

Shen, H., et al. (2024). *Position: Towards Bidirectional Human-AI Alignment.* arXiv:2406.09264.

Shimomoto, T. (2026a). *Agent Knowledge Cycle (AKC).* GitHub / Zenodo. Concept DOI: 10.5281/zenodo.19200726 (version DOI, v2.2.0: 10.5281/zenodo.20565806)

Shimomoto, T. (2026b). *Contemplative Agent (Version 2.5.0).* GitHub / Zenodo. Concept DOI: 10.5281/zenodo.19212118 (version DOI, v2.5.0: 10.5281/zenodo.20555864)

Shimomoto, T. (2026c). *Distributing Accountability, Not Capability: Phase Separation and the LLM Workflow Quadrant in Autonomous AI Agent Architectures* (v1). Zenodo working paper. Concept DOI: 10.5281/zenodo.20353789 (version DOI, v1: 10.5281/zenodo.20353790)

Shimomoto, T. (2026d). *Stop Using Default Settings — 10 Claude Code Configs That Actually Work.* dev.to. https://dev.to/shimo4228/stop-using-default-settings-10-claude-code-configs-that-actually-work-243l

Snook, S. A. (2000). *Friendly Fire.* Princeton University Press. [Cited as characterized in secondary literature; book text not directly verified.]

Sumers, T. R., Yao, S., Narasimhan, K., & Griffiths, T. L. (2023). *Cognitive Architectures for Language Agents.* arXiv:2309.02427.

Wang, G., et al. (2023). *Voyager: An Open-Ended Embodied Agent with Large Language Models.* arXiv:2305.16291.

Wang, Z. Z., Mao, J., Fried, D., & Neubig, G. (2024). *Agent Workflow Memory.* arXiv:2409.07429.

Zhou, C., et al. (2026). *Externalization in LLM Agents: A Unified Review of Memory, Skills, Protocols and Harness Engineering.* arXiv:2604.08224.