16 — Learning & Self-Improvement

🕒 state of the art 2026-07 · revised 2026-07-28 · 📖 ~8 min read

Learning objectives

By the end of this chapter, you should be able to:

1. **Explain** why self-improving learning breaks the assumption of *static* scaffolding — and how it inverts the book's expiration clause;
2. **Describe** the stages of the closed skill-capture cycle (trigger, curation, isolation, portable format, indexed re-encounter, maintenance against entropy);
3. **Compare** the two competing designs for applying what was learned — autonomous × human promotion — and **locate** a real harness on the dimension's maturity ladder;
4. **Evaluate** the dimension's risks (superstition, entropy, contamination, prompt injection as permanent learning) and the engineering that prevents them.

The problem

The twelve dimensions of chapters 02–13 describe *static* scaffolding: someone — the harness author, the user, a plugin — writes the instructions, tools and policies, and the agent consumes them. This chapter documents the emerging dimension that breaks that assumption: the agent that **writes its own scaffolding** — capturing learned procedures as reusable skills.

The dimension was promoted to supplementary status in the benchmark template (dimension 13) on the strength of one piece of evidence: the **Hermes Agent** (Nous Research) implements the full cycle, and reading the code confirms each stage (Appendix A).

The state of the art

The closed cycle: the six stages

The reference mechanism, verified in the Hermes code (detailed evidence in Appendix A), closes the cycle in six stages:

1. **Autonomous trigger** — the learning review fires on its own, in the background, without the user asking (with a manual trigger as a complement).
2. **Curation by an isolated fork** — a clone of the agent, with a curatorial prompt that defines what to capture and — most importantly — **anti-patterns of what NOT to learn**. Without that list, the system would degenerate into accumulated superstition.
3. **Isolation of the meta-work** — the curator fork has restricted tools and persistence turned off, so as not to contaminate the real session.

4. **Writing in a portable format** — the skill becomes a `SKILL.md` under strict standards, with the context constraint shaping the format of the knowledge.
5. **Cheap re-encounter** — a compact index always in the system prompt; the full content only enters the context on demand. Learning indexed, not dumped.
6. **Maintenance against entropy** — a periodic curator consolidates, archives by inactivity, and protects what is pinned. Memory that only grows becomes noise; the curator is the knowledge's garbage collector.

The maturity ladder in the evaluated cohort

Harness	Score 13	What it has
Hermes	**3**	The complete closed cycle (Appendix A), with autonomous application
gemini-cli	**3** (retro)	Auto Memory: an extractor agent with anti-noise gates ("Default to NO SKILL", 5 blocking questions) producing SKILL.md + memory patches — but with **human promotion via inbox** (`/memory inbox`); dedupe, write sandbox, dedicated evals
IronClaw	2	Automatic skill extraction (`learning.rs`) with usage/confidence metrics and versioning
OpenClaw	1	Dreaming (autonomous memory consolidation); Skill Workshop with a proposal queue
OpenHarness	1 (retro)	Auto-extraction of facts per turn, with usage-based staleness (60 days) — facts, not procedures
Codex CLI	1	Automatic memories with pruning (facts, not procedures)
Goose	1	chatrecall (semantic recall of past conversations)
opencode, the rest	0 (retro)	Skills are consumption/distribution; nothing is written from experience

The ladder is sharp: **memory of facts** (level 1) → **extraction of procedures** (level 2) → **curated cycle with anti-patterns and maintenance** (level 3). What sets level 3 apart is not capturing more — it is the engineering of *not* capturing wrongly and of pruning what has aged.

The two competing designs at level 3

Level 3 already has **two competing designs**, with the divergence exactly where it matters: *who applies what was learned*. Hermes applies it autonomously (with the curator cleaning up afterwards); gemini-cli requires human promotion (inbox — nothing enters the context without `/memory inbox`). It is the classic autonomy × control trade-off of chapter 07, reappearing in the newest dimension: Hermes bets that anti-patterns are enough to prevent bad learning; gemini-cli bets they are not. The coming rounds will tell which scales better.

Why this changes the book's thesis

The expiration clause (chs. 01, 14) says: every harness component is a prosthesis for a current model limitation, and expires when the model improves. Self-improving learning **inverts the clause**: instead of waiting for the model to render the scaffolding unnecessary, the model+harness pair *writes new scaffolding for itself*. Each

learned skill is a piece of harness generated at runtime, specific to the user and the environment — something no harness author could have written at the factory.

This creates a third path in the taxonomy:

1. **Factory scaffolding** — written by the harness author; expires as models evolve.
2. **Boundary scaffolding** — sandbox, permissions, interfaces; does not expire (it is about the world).
3. **Self-generated scaffolding** — skills written by the agent; it *grows* with use, and its quality depends on the curation engineering, not on the model's raw capability.

The risks: the mirror of the promises

The risks are the mirror of the promises: without anti-patterns, superstition; without curation, entropy; without isolation of the meta-work, contamination; and — pointed out by the IronClaw evaluation (prompt-write safety; cf. ch. 07) — without a protected write boundary, **prompt injection becomes permanent learning**: an attacker who convinces the agent to "learn" a malicious skill persists in procedural memory. A mature dimension 13 will require a mature dimension 6.

EXECUTIVE SUMMARY

The dimension is the newest in the template and the least converged: two harnesses at level 3 with opposite designs on who applies the learning, and the rest of the cohort somewhere between memory of facts and nothing. What is already engineering consensus among those who got there: the central piece is not the capture mechanism, but the **anti-patterns of what not to learn** and the **maintenance** (consolidate, archive, never delete). **What to steal** today: an anti-pattern list in the curatorial prompt; isolation of the meta-work in a fork without persistence; a compact index with content on demand; a periodic curator as garbage collector; a write boundary protected against prompt injection.

Retroactive re-evaluation of the code cohort pending; the dimension leaves "supplementary" status when ≥3 harnesses reach level 2+.

> **See also**: the living collection Awesome Harness Engineering — Skills & MCP gathers more consultable resources for this dimension, curated by problem.

Check your understanding

1. Why is the **anti-pattern** list ("what NOT to learn") described as the central piece of the curatorial engineering, rather than the capture mechanism itself? What happens to a system that captures without it?
2. Locate on the maturity ladder a harness that extracts facts automatically with usage-based staleness but does not capture procedures. What score does it receive, and what would it need to climb one level?

3. Hermes and gemini-cli are both at level 3, but diverge on *who applies* what was learned. Reconstruct the autonomy × control trade-off in this context: what is each design's bet?
4. Explain the sentence "a mature dimension 13 will require a mature dimension 6": why is prompt injection qualitatively more serious in a harness that learns than in a static harness?

Appendix A — Hermes Agent

> Per-repository evidence, with paths — supplementary material (online version), expanded each benchmark round. Full evaluation: `../../benchmark/avaliacoes/hermes-agent.md`.

Hermes's closed cycle (evidence: `agent/background_review.py` and related)

The mechanism, verified in the code of the evaluated fork:

1. Autonomous trigger. Every ~10 tool-calling iterations (`skill_nudge_interval`, in `agent/turn_finalizer.py`), the harness fires a background review — without the user asking. There is also the manual `/learn` trigger.

2. Curation by an isolated fork. A clone of the agent runs in a separate thread with a snapshot of the conversation and a curatorial prompt (`_SKILL_REVIEW_PROMPT`) that is the central piece of the engineering. It instructs the curator to be active ("a pass that does nothing is lost learning"), defines an order of preference (update an existing skill > create a new one; new skills only class-level, never "fix-bug-1234") and — most importantly — lists **anti-patterns of what NOT to learn**: environment-dependent failures, negative claims about tools ("the browser doesn't work"), transient errors, one-off narratives. Without that list, the system would degenerate into accumulated superstition.

3. Isolation of the meta-work. The fork has a restricted tool whitelist (`memory` + `skills`), memory and persistence turned off — so the curation does not contaminate the real session — and inherits the parent's cached prompt prefix (~26% reduction in the cost of the review).

4. Writing in a portable format. The skill becomes a `SKILL.md` compatible with **agentskills.io** in `~/.hermes/skills/<categoria>/<nome>/` (with `references/`, `templates/`, `scripts/`), under strict standards — description ≤60 characters *because the index in the system prompt truncates at 60*: the context constraint shaping the format of the knowledge.

5. Cheap re-encounter. The compact index (name + description) is always in the system prompt; the full content only enters the context when the agent calls `skill_view` — learning indexed, not dumped.

6. Maintenance against entropy. A periodic **curator** (`agent/curator.py`) runs when the agent is idle: it consolidates skills into umbrellas, archives by inactivity (90 days — archive, never delete), and protects pinned skills. Memory that only grows becomes noise; the curator is the knowledge's garbage collector.