Verifier Autonomy Audit

AI Factory Audit — Plan Task 2.2: Verifier Autonomy

Date: 2026-05-09 **Auditor:** Martin Kleppmann (CodeCraft) **Classification:** AUDIT-ONLY — read-only, no mutation, no live invocation

---

VERDICT SUMMARY (up front)

Autonomy verdict: ABSENT

The `/verify-fix-loop` skill is fully specified and internally consistent, but it has zero wiring into any automated trigger path. CEO is the de-facto verifier for every task that reaches `mc.js ready`. The skill exists only as a manually-invoked slash command.

---

1. End-to-End Trace of `/verify-fix-loop`

Source: `~/.claude/skills/verify-fix-loop/SKILL.md`

Flow map

```
Caller (John / human) invokes: /verify-fix-loop mc_id=<N> spec_path=<path>
    |
    ▼
SKILL orchestrates in main conversation thread (not a sub-agent itself)
    |
    ├─ mkdir -p /tmp/verify-fix-loop-<mc_id>/    (EVIDENCE_DIR)
```

```
    │
    ▼
LOOP (max 3 iterations):
    │
    ├─ Step A: Task(subagent_type=verifier OR general-purpose+persona)
    │     prompt = verifier brief template (inline in SKILL.md)
    │     verifier writes: EVIDENCE_DIR/verifier-loop<N>.md  (mandatory)
    │                      /tmp/verifier-feedback-<mc_id>.md (if CONFIDENCE=FEEDBACK)
    │
    ├─ Step B: Parse STATUS + CONFIDENCE from verifier output
    │
    ├─ Step C: Branch
    │     PERFECT / VERIFIED → write SUMMARY.md (SUCCESS), exit
    │     PARTIAL            → if high_stakes: ESCALATE; else: SUCCESS_WITH_NOTES, exit
    │     FAILED             → ESCALATE (harness broken)
    │     FEEDBACK:
    │         if high_stakes or budget exhausted → ESCALATE
    │         else →
    │
    ├─ Step D: Task(subagent_type=fix-builder OR general-purpose+persona)
    │     reads /tmp/verifier-feedback-<mc_id>.md
    │     applies prescribed edits to spec_path via Edit tool
    │     returns APPLIED:<N> / PARTIAL:<N>/<M> / COULD_NOT_APPLY:<reason>
    │
    └─ LOOP_INDEX += 1 → back to Step A
```

Domain escalation policy

- `docs`, `system`, `refactor`, `polish` — loops up to MAX_LOOPS (default 3)
- `security`, `finance`, `legal`, `deploy`, `infra`, `unknown` — ESCALATE on first FEEDBACK (no autonomous correction)

Loop budget

- Default MAX_LOOPS = 3
- Hard cost cap: $5 per skill invocation
- Per-loop cost estimate: $0.40–0.60 (Sonnet)
- Worst case: 3 × $0.60 = $1.80

Termination conditions

1. CONFIDENCE in {PERFECT, VERIFIED} → SUCCESS
2. CONFIDENCE == PARTIAL + not high_stakes → SUCCESS_WITH_NOTES
3. Budget exhausted (LOOP_INDEX == MAX_LOOPS with FEEDBACK) → ESCALATE
4. High-stakes domain with FEEDBACK on first iteration → ESCALATE
5. Any FAILED confidence → ESCALATE (harness broken)
6. fix-builder returns COULD_NOT_APPLY → ESCALATE
7. MC status changes to done/cancelled mid-loop → ABORT silently
8. Cost estimate exceeds $5 → ESCALATE before next iter

Entry points (who can call this)

The SKILL.md lists trigger phrases: "verify-fix-loop", "auto-verify and fix", "verifier loop", "ne idi preko mene", "loop until pass". All trigger phrases are designed for **human invocation** in a conversation. No programmatic entry points exist.

---

2. Auto-Invocation Analysis — The Central CEO Question

pi-orchestrator.js

Grep result: ZERO matches for `verify-fix-loop`, `verifier`, `fix-builder` in `~/system/kernel/pi-orchestrator.js`.

The orchestrator's post-completion flow (`reportCompletion` function, lines ~3781–3930) does:

- Hallucination detection (regex-based `detectHallucination`)
- Proof-of-work check (GOTCHA file or response length)
- qa-19 Check #20 (endpoint verification, if configured)
- Postflight marker write to `~/system/state/postflight-cleared-<id>.json`

None of these steps call the verifier, fix-builder, or verify-fix-loop skill. The "postflight" referenced in pi-orchestrator is a file marker write, NOT the `/task-postflight` skill.

task-postflight skill

Grep result: ZERO matches for `verify-fix-loop`, `verifier`, `fix-builder` in `~/.claude/skills/task-postflight/SKILL.md`.

The `/task-postflight` skill dispatches Angie Jones (Proveo) for AC-checklist QA, not the atomic-claim verifier. These are parallel, non-overlapping verification patterns:

- Proveo = human-readable AC checklist with pass/fail verdicts per item
- Verifier = atomic claim decomposition with machine-verified proof citations

Hooks directory

Grep result: Only archive files matched. No active hook in `~/.claude/hooks/` references `verify-fix-loop`, `verifier`, or `fix-builder`.

Active hooks audited:

- `liveness-claim-validator.sh` — PostToolUse on Write/Edit; checks for bare liveness claims in memory/spec/agent files. Not related to verifier dispatch.
- `mc-ready-gate.sh` — wrapper for `mc.js ready`; runs ZAKON #30 direct-probe gate + evidence-contract-validator. Does NOT invoke verify-fix-loop.
- `evidence-contract-validator.sh` — validates verdict JSON schema + sha256 chain. Shell-based, no agent dispatch.
- `cross-session-claim-gate.sh`, `session-task-lock-gate.sh`, `plan-completeness-gate.sh`, `pre-dispatch-gate.sh` — none reference verifier.

Daemon fleet

Grep result: ZERO matches for `verify-fix-loop`, `verifier`, `fix-builder` in `~/system/daemons/`.

LaunchAgents

Grep result: ZERO matches in `~/Library/LaunchAgents/`.

VERDICT: ABSENT

The verify-fix-loop and its constituent agents (verifier, fix-builder) have zero automated entry points. The only invocation path is a human typing a trigger phrase in a Claude Code conversation. CEO is always in the loop because there is no loop without CEO.

---

3. Tool-Surface Security Check

Verifier (read-only)

Definition file: `~/.claude/agents/verifier.md` **Declared tools:** `tools: Read, Grep, Glob, Bash`

The `tools:` field includes Bash. This is the critical point.

The agent definition does NOT use a tool whitelist that removes Write/Edit/Task at the API level. It relies entirely on prompt-level enforcement ("Enforcement is prompt-only — this rule is yours to honor. You are the gatekeeper."). The verifier.md explicitly states this.

Permitted Bash commands (per prompt whitelist in verifier.md):

- cat, head, tail, wc, ls, file, stat
- diff, git read-only subcommands
- grep, rg, find (via tool preferred)
- jq, node -e (read-only expression)
- node ~/system/tools/mc.js show (read-only subcommands only — NEVER add|start|done|ready|update|pause|cancel)
- gh pr view, gh issue view, gh api -X GET
- sqlite3 -readonly, psql SELECT only
- curl -sI (HEAD), curl -s GET (never POST/PUT/DELETE)
- bash -n, shellcheck, node --check (dry-run linters)

Escape paths documented:

- The prompt says "NEVER run: rm, mv, cp (to non-/tmp/), chmod, chown, ln" and "Redirections that write outside /tmp/verifier-* or /tmp/<task_id>-evidence/: >, >>, tee to other paths".
- This is prompt-level enforcement only. A model following instructions could still run `bash -c "echo foo > ~/system/some-file.txt"` — the agent framework does not block it at the API tool-call level.
- The `tools: Bash` declaration gives the agent full shell access; the prompt whitelist is self-enforced.
- Feedback file writes are permitted to `/tmp/verifier-feedback-<TASK_ID>.md` specifically.

Verdict on verifier tool isolation: Prompt-enforced, not API-enforced. Read-only is a behavioral constraint, not a structural constraint. The risk is manageable for a trusted model, but not cryptographically bounded.

Fix-builder (write-only, scoped)

Definition file: `~/.claude/agents/fix-builder.md` **Declared tools:** `tools: Read, Edit, Grep, Glob`

The fix-builder tool list explicitly excludes:

- Write (no new file creation)
- Bash (no test runs, deploys, builds, git ops)
- Task (no further dispatch)

This is stronger isolation than the verifier: the `tools:` field at the agent definition level excludes Bash and Write. If the agent framework enforces declared tools as a whitelist, fix-builder genuinely cannot run shell commands or create new files. It can only read existing files (Read, Grep, Glob)

and apply edits to existing files (Edit).

Gap: Fix-builder cannot create new files even when feedback prescribes it. The skill handles this: "If the feedback prescribes creating a new file, mark that fix as COULD_NOT_APPLY" — the loop escalates. This is a by-design limitation, not a bug.

Verdict on fix-builder tool isolation: Structurally scoped (Bash and Write excluded from tools declaration). This is the correct pattern. The verifier should be refactored to match this approach.

---

4. Synthetic Dry-Trace

Selected task: MC #99389 — "Refactor /mehanik skill to progressive-disclosure pattern" (status: review, owner: pi-orchestrator)

This task was marked `mc.js ready` (now `review`) after pi-orchestrator completed it.

What WOULD have happened if /verify-fix-loop were auto-invoked:

```
Step 0: trigger fired when pi-orchestrator called mc.js ready #99389
         → /verify-fix-loop mc_id=99389 spec_path=~/.claude/skills/mehanik/SKILL.md
            domain=docs (inferred from skill file path)
            max_loops=3

Step A (iter 1): dispatch verifier
  - verifier reads ~/.claude/skills/mehanik/SKILL.md
  - verifier reads MC #99389 ACs via mc.js show 99389
  - verifier decomposes ACs into atomic claims:
      (a) SKILL.md exists and is < N lines (tier-1 constraint)
      (b) references/agent-brief.md exists
      (c) references/failure-modes.md exists
      (d) Skill tool callable post-refactor
  - verifier probes each atom with Read/Glob/Bash

Step B: parse CONFIDENCE
  If all files exist and SKILL.md is within limits → PERFECT → SUCCESS
  If any reference file missing → FEEDBACK

Step D (if FEEDBACK): dispatch fix-builder
  - fix-builder reads /tmp/verifier-feedback-99389.md
  - applies Edit to create missing sections or correct line counts
```

```
Step C (iter 2): re-verify → likely PERFECT → write SUMMARY.md → SUCCESS
```

Actual closure path used for MC #99389: The task is in `review` status. Looking at the review queue (25+ tasks in review), there is no evidence of verifier invocation. The closure path was: pi-orchestrator marked ready → task sits in `review` queue → CEO/John is the implicit reviewer. This is the CEO-as-verifier pattern the CEO wants to eliminate.

---

5. Comparison with Existing Patterns

liveness-claim-validator.sh

- **Trigger:** PostToolUse hook, fires on every Write/Edit/MultiEdit tool call
- **Scope:** Memory files, spec files, agent definition files matching 4 path patterns
- **Mechanism:** Shell script reads tool input JSON from stdin, scans written content for bare liveness claims, blocks write if violations found (exit 2)
- **Auto-invoked:** YES, unconditionally, at the Claude Code hook level
- **Why verify-fix-loop is NOT similarly hooked:** The liveness validator is a passive scan that reads content already being written. The verify-fix-loop requires active agent dispatch (spawning sub-agents), which cannot be done from a shell hook. Shell hooks can block tool calls; they cannot spawn conversational agents.

This is the fundamental architectural gap: hooks can intercept tool calls synchronously, but spinning up a verify-fix-loop requires an async agent conversation that the hook system cannot initiate.

evidence-verifier agent

File: `~/.claude/agents/evidence-verifier.md` **Declared tools:** (not in scope of this read — but confirmed the agent exists) **Auto-invoked:** YES, but differently — it is called by `mc-ready-gate.sh` via the `evidence-contract-validator.sh` pathway. However, the evidence-contract-validator.sh is a pure shell script that validates JSON schema + file hashes — it does NOT dispatch the evidence-verifier agent. The agent definition exists for manual invocation. The shell script performs a deterministic (non-LLM) validation that is auto-invoked at `mc.js ready` time.

Pattern difference: The evidence-verifier pattern uses a shell script as the auto-invoke layer (deterministic, no LLM), with the agent definition as a fallback for edge cases. The verify-fix-loop requires LLM reasoning at every step, making shell-script auto-invocation insufficient.

---

6. Gap Analysis and Fix Proposal (Audit-Level Only)

Root cause of the gap

The verify-fix-loop was designed top-down as a skill (manual invocation). The liveness-claim-validator was designed bottom-up as a hook (automatic). There is no bridge layer that translates "mc.js ready event" → "spawn verify-fix-loop conversation".

The missing component is a **postflight agent dispatcher**: something that observes the `ready` event and spawns a verify-fix-loop session as a sub-agent task.

Minimum wiring needed

Option A: PostToolUse hook on mc.js ready (recommended)

Element	Detail
File to modify	`~/.claude/hooks/mc-ready-gate.sh` (already fires on mc.js ready)
Addition location	After line 196 (all gates passed — currently execs mc.js directly)
Trigger	After mc.js ready succeeds, spawn verify-fix-loop as a background Task
Mechanism	`mc-ready-gate.sh` would write a trigger file to `/tmp/vfl-trigger-<mc_id>.json` containing mc_id + spec_path + domain; a daemon polls this file

The problem: `mc-ready-gate.sh` is a synchronous shell script. It cannot spawn a conversational agent (Task dispatch requires a running Claude Code session). It can only write a file.

Option B: pi-orchestrator.js postflight hook (most natural wiring point)

Element	Detail
File to modify	`~/system/kernel/pi-orchestrator.js`
Addition location	Inside `reportCompletion()` function, after line ~3900 (after QA gate passes)
What to add	A call to write `/tmp/vfl-trigger-<task_id>.json` with task metadata
Trigger	The daemon below polls this and dispatches

Option C: /task-postflight skill modification (cleanest for H-tasks)

Element	Detail
File to modify	`~/.claude/skills/task-postflight/SKILL.md`
Addition location	After Section 2 (PROVEO VALIDATION DISPATCH), add Section 2b
What to add	Conditional: if Proveo returns PASS AND task domain is docs/system/refactor, dispatch /verify-fix-loop before writing the postflight marker
Trigger	Manual invocation of /task-postflight already exists for H/BLOCKER tasks
Advantage	Stays within the skill conversation context — Task dispatch works naturally here

Recommended wiring (Option C + Option B trigger file):

1. **Immediate (no new infrastructure):** Add a Section 2b to `/task-postflight` SKILL.md that dispatches `/verify-fix-loop` when Proveo passes and domain is non-high-stakes. This works today for all tasks that go through `/task-postflight`.
2. **Systematic (covers tasks that bypass /task-postflight):** Add a trigger file write to `pi-orchestrator.js` `reportCompletion()`. A lightweight daemon polls `/tmp/vfl-trigger-*.json` files and — when a pi-orchestrator session is active — dispatches the verify-fix-loop skill via the existing Claude Code session.

Loop budget recommendation

- Keep MAX_LOOPS = 3 (matches SKILL.md default)
- For postflight auto-invocation, restrict to `docs`, `system`, `refactor`, `polish` domains only
- Hard cap: $5 per invocation (already in SKILL.md)
- Add timeout: 5 minutes wall-clock before auto-escalation to CEO

Escalation path when budget exhausted

1. Write SUMMARY.md to EVIDENCE_DIR with full loop history
2. Call `node ~/system/tools/slack.js send alerts "[VFL-ESCALATED] MC #<id> — N/MAX loops used, last verdict: <CONFIDENCE>"` (Slack, not CEO direct)
3. Set task status to `blocked` via `mc.js block` with reason "verify-fix-loop budget exhausted — human review needed"
4. John receives Slack alert and decides: (a) override + mark done, (b) dispatch additional builder, (c) extend budget via [CEO_APPROVED] token

---

Open Questions

1. **Tool-level enforcement for verifier:** Should the verifier's `tools:` field be changed from `Read, Grep, Glob, Bash` to `Read, Grep, Glob` (removing Bash) to achieve structural isolation matching fix-builder? This would break the verifier's ability to run `curl -sI`, `git log`, `sqlite3 -readonly` probes — which are core to its value. The tradeoff is behavioral (current) vs structural enforcement.
2. **Conversation context for auto-dispatch:** Spawning a verify-fix-loop Task requires an active Claude Code conversation. If pi-orchestrator fires after a conversation closes, there is no context to spawn into. Does the system need a persistent "factory session" that stays open to receive postflight dispatches?
3. **High-stakes domain detection:** The SKILL.md defaults unknown domains to HIGH_STAKES (no autonomous correction). For auto-invocation, domain inference from spec path heuristics will frequently return unknown. Should the default be flipped to docs for auto-invoked postflight use cases?
4. **Proveo vs verifier: overlap management:** `/task-postflight` already dispatches Proveo for AC-checklist QA. If verify-fix-loop is added as Section 2b, tasks will run both Proveo (AC checklist) AND verifier (atomic claims) sequentially. Is this the intended double-verification model, or should one replace the other for certain task types?
5. **mc.js ready event vs pi-orchestrator ready:** Some tasks are marked ready by human John (`node ~/system/tools/mc.js ready <id>`), others by pi-orchestrator after build completion, and others by `/task-postflight`. The auto-invocation wiring point differs for each path. A comprehensive solution needs to intercept all three paths.

Evidence Metadata

Item	Value
Files read	8
Grep/Bash tool calls	12
Live agent invocations	0
Mutations	0
Wall-clock (estimated)	~18 min
Key source files	`~/.claude/skills/verify-fix-loop/SKILL.md`, `~/.claude/agents/verifier.md`, `~/.claude/agents/fix-builder.md`, `~/.claude/skills/task-postflight/SKILL.md`, `~/system/kernel/pi-orchestrator.js` (lines 3730–3930), `~/.claude/hooks/mc-ready-gate.sh`, `~/.claude/hooks/liveness-claim-validator.sh`

Revision #3
Created 2026-05-09 19:44:21 UTC by John
Updated 2026-07-19 20:01:13 UTC by John