DEEP-SOLVE V2 / CODE-VERIFIED

Loop Engineering: Your Plan

You already have the Pam loop, and it is real. The audit read your actual code, found it is partial in two specific ways, and surfaced one thing that needs fixing tonight.

Prepared for Anthony Updated June 14, 2026 8-agent audit, 535 file reads of the real code

TL;DR

- **You were right: Pam is already a maker/checker loop.** Verified in code: Pam makes (Sonnet), a separate Haiku grader scores her against the real human/Yuma answer, a second blind grader runs on Ross's own machine, and a zero-AI referee reconciles them. It is firing, and it has caught real grading errors.
- **It is a PARTIAL loop for two reasons.** The learn step has a manual hop (fixes reach Pam only when a human clicks approve), and the grade is mostly AI-vs-AI (about 80% of the answer keys are Yuma, and both graders are the same model). High agreement can hide a shared blind spot.
- **One thing needs action now.** Ross's second grader has been silent 2 days and grader agreement is 65.2% against your 90% bar. Pam's loop is running one-grader right now. Your own tripwire caught it. It needs a fix.
- **The rule I gave you was too strict, and you were right to push.** The real gate is not "does it repeat," it is "can the checker objectively verify it." One-off projects loop fine. Repeating only decides permanent vs disposable.
- **Jarvis is the opposite of Pam: zero checker today.** The plan splits him into a live deterministic check on irreversible actions plus an offline Pam-style judgment loop. That is how he safely becomes more like you. Cut from the plan per your call: the observability build.

1. The rule, corrected (you were right)

I told you loops are for repeating work. You pushed back: a big one-off project should still loop, because you give the objective and it self-drives with a checker. You are right, and I owe you the precise version.

> THE REAL GATE
>
> Not "does it repeat." It is **"can the checker objectively verify progress toward the objective?"** If you can write down the pass condition, you can loop it, one-off or not. "Repeating" only decides one thing: whether you build a **permanent, scheduled** loop (worth the setup only if you reuse it) or a **disposable** one you run once and throw away. Either way, you loop.

Two things stay true even on a one-off, because this is where the wording matters:

- The checker still has to be **separate** from the maker. "It fact-checks itself" is the one trap. The agent that did the work grades its own homework too kindly.
- The checker needs **something objective to check against**: tests/lint for code, or a written rubric / definition-of-done / source-of-truth for everything else. If "done" is pure taste, turn "good" into a checkable rubric first, then loop.

2. /goal vs building your own checker

Your question: how is a hand-built maker/checker different from the `/goal` command?

	/GOAL	HAND-BUILT MAKER/CHECKER
The checker	Built in, hidden, generic. Judges one thing: "is the exit condition met."	You write it. A separate agent, fresh context, skeptic instructions, grading against your rubric.
You design	Just the goal + exit condition.	The objective, the rubric, and the checker's lens.

Best for	Verification that is already objective and automatable (tests pass, lint clean).	Verification that is a judgment call (a doc, a model, a customer reply, research).
Setup	One line.	A few minutes: objective + checker prompt + round cap.

Same shape underneath. `/goal` hides the checker; hand-built lets you see it, swap it, and make it adversarial. **The punchline:** you do not have to build infrastructure. Hand me a one-off project plus a one-line pass condition, and I run maker then separate skeptical checker then fix, capped at 2 rounds. That is your one-off loop. (Note: `/loop` is confirmed live in your setup; `/goal` is the newer command from the articles and I have not yet confirmed it is enabled in your build.)

3. Pam audit: you already have the loop

I read the real files, not memory. Verdict: **yes, partial.** Your mental model is almost entirely correct.

WHERE YOU ARE RIGHT (ALL VERIFIED)

- ✓ Pam IS the maker (live listener + backtest replay, Sonnet).
- ✓ The nightly backtest DOES compare her to the human-or-Yuma answer that actually resolved the ticket.
- ✓ She IS graded and scored, rolled into the 90% gate.
- ✓ Ross-on-her-own-Claude IS a real, separate, blind checker. Proven: 8 of 23 tickets disagreed, and Ross ruled the primary grader wrong on 4.

WHERE THE MODEL IS SLIGHTLY OFF

- → Ross's Claude checks the GRADE, not Pam directly. It grades the same tickets to catch a bad grader. The thing checking Pam is the Haiku grader; Ross checks the

grader.

- → The loop is OPEN, not closed: fixes reach Pam only when a human clicks approve. There is a manual hop in "she gets graded and improves."
- → The grade is mostly AI-vs-AI: about 80% of answer keys are Yuma, and both graders are the same model on the same playbook.
- → The 90% gate is NOT wired into live Pam. Testing mode is enforced by construction (no customer-send path exists), not by the gate.

The maker / checker / stop / state checklist, as it exists today

PART	STATUS	WHAT EXISTS IN YOUR CODE
Maker	EXISTS	Live Pam (15s Slack listener, Sonnet, read-only connectors); backtest replays the same config.
Separate checker	PARTIAL	Primary Haiku grader (separate subprocess, cannot grade its own context) + Ross's blind Haiku audit + deterministic referee. Weakness: same model family; audit grader currently down.
Stop condition	PARTIAL	Fixed-count per run, dedup so each ticket grades once. No iterate-until-90% and no token ceiling that halts.
State on disk	EXISTS	proposals.jsonl, cst-learned.md (approval-only), the shared Grading Audit sheet. Append-only, resumes across days.

The fix list (ranked, all small to medium effort)

#	FIX	WHY IT MATTERS
1	Close the learn loop: auto-route confirmed grader mistakes + approved fixes into the proposal queue (keep the human approve click).	The manual hop is the single biggest reason this is partial, and what stalls the climb to 90%.
2	Report human-only score and require a minimum of human-keyed tickets before a	Today 90% could mean 90%-against-Yuma. The bar must mean 90%-against-

	topic clears 90%. Build a small hand-graded gold set.	humans.
3	Run the audit grader on a different model + a fallback copy on your always-on Mac.	Two same-model graders can be confidently wrong together, and the off-machine one is silent now.
4	Wire the 90% gate into the live listener as a real interlock (before any customer-send path exists).	Testing mode is a convention, not a coded lock. Build the lock first.
5	Add an LLM checker to the LIVE path + hard-gate the $100 rule in code (force-flag if no successful lookup or amount over/empty).	Live Pam's only check is a regex; a plausible fabricated tracking number passes it, and the $100 self-approve has no reviewer.
6	Add a halting token ceiling (using the telemetry you already collect) + dedupe cst-learned.md.	An ~18M-token backfill was caught after the fact; a near-duplicate rule appears about 6 times and bloats her context.

PATH TO 90% THEN LIVE, AT 75-80% AUTONOMY

Fix the bar (human-keyed + gold set), make the graders trustworthy (different model + always-on fallback), close the loop (auto-route confirmed errors to the queue, human approve stays). Let the nightly rule-staging lift the score. Then go live **per category, not all at once**: add the customer-send path only for categories that cleared 90% on human-keyed tickets, behind the interlock. For 75-80% autonomy: keep the $100 self-approve but hard-gated in code, let Pam send on cleared categories, reserve the human for over-$100, repeat courtesy, cash refund outside policy, legal, and any sub-90 category. The token ceiling protects your 5-hour window instead of a dollar cap.

4. Act tonight: the loop is half-blind right now

LIVE FINDING FROM THE REFEREE LOG

Ross's audit grader has been silent 2 days (June 13 and 14). Grader agreement is **65.2% over the last 14 days, against your 90% bar.** That means Pam's primary

grades are accruing unaudited, and the maker is currently being checked by a single same-family Haiku with no independent confirmation. The system flagged it itself, which is the loop working. It now needs a human (you) to act: either kick Ross's grader back on, or stand up the always-on fallback copy from fix #3. Want me to investigate why it went silent and stand up the fallback tonight?

---

5. The Jarvis loop, designed

You asked how a loop would work on Jarvis. First the honest finding: **Jarvis has no checker at all today.** His reply streams straight to Slack with no second model reviewing it, and in owner mode he runs fully ungated (bypass permissions, no per-action confirmation). The only guards are deterministic safety rails (redaction, a tool firewall for teammates, an after-the-fact audit sheet). They check don't-leak and can't-touch, never "is the answer right."

The key insight the audit surfaced: Jarvis's output splits into two halves that need two different loops.

TRACK A. THE PART THAT IS CHECKABLE (BUILD FIRST)

Tool side-effects are objective: he filed a card, wrote a Sheet cell, moved a column. Before flipping to the success reaction, a **deterministic post-condition check** re-reads the system and confirms the write matches intent. Irreversible or low-confidence actions route to your existing "Ready to Review" Basecamp column (enforced in code, not by convention). Plus a cheap-model pass for format/policy/hallucination before posting. This is real, today, and low-risk.

TRACK B. THE JUDGMENT PART (THE PAM PATTERN)

His advice and decisions are not live-checkable, so you do it offline, exactly like Pam: build a replay set of real Jarvis threads with your known-good answer, run a separate grader against it, distill the systematic misses into prompt rules, you approve. That is how Jarvis "learns to decide like you" over weeks without ever getting full autonomy. He stays an ambient agent with checkpoints, which is what every source says these agents require.

At 75-80% autonomy for Jarvis: keep bypass for reversible and read actions, gate the irreversible ones (Done moves, Sheet writes, any send to a non-Anthony recipient) behind the Track A post-condition check. And add a token + max-turns budget to the daemon, which is currently uncapped.

---

6. The plan (updated)

MO-1 Decided

First loop: Pam. Confirmed.

Autonomy target 75-80%. No hard dollar budget (token-window based). Observability build cut per your call (cron-health + scheduling tab already cover it).

MO-2 Act tonight

Restore the second grader

Investigate why Ross's audit grader is silent, and stand up an always-on fallback copy (fix #3) so the loop stops running one-grader.

MO-3 Next

Close the loop + fix the bar

Auto-route confirmed grader errors and approved fixes into the proposal queue (fix #1). Split the score by human-vs-Yuma and require human-keyed tickets for the 90% gate; build a small gold set (fix #2).

MO-4 Next

Harden the live path

Add an LLM checker before posting, hard-gate the $100 rule in code, add a halting token ceiling, dedupe cst-learned.md (fixes #5, #6).

MO-5 Then

Go live per-category behind the interlock

Wire the 90% gate into the listener (fix #4). Add the customer-send path only for categories that cleared 90% on human-keyed tickets, at 75-80% autonomy.

MO-6 Parallel or after

Jarvis two-track loop

Track A (deterministic post-condition check on irreversible actions) first, Track B (offline judgment eval) modeled on Pam once Pam's pattern is proven.

MO-7 After Pam proves it

Templatize into a team /loop-starter skill

Guided skill via your eg-skills pipeline: interviews a teammate, checks the verify-able gate, scaffolds maker/checker/stop/state, dry-runs it. Battle-tested template, not theory.

7. What you don't know that you'll wish you did

1. **A loop catching its own failure IS the loop working.**
   The 65.2% alert is not the loop being broken, it is the loop doing its job. The failure mode to fear is the opposite: green everywhere while nobody is actually checking.

2. **Open loop vs closed loop is the real maturity line.**
   Pam grades and you approve, but the path from "confirmed error" to "fix queued" is manual. Closing that hop is worth more than any new model.

3. **Two checkers of the same model is a same-model echo.**

They can be confidently wrong together. Real independence means a different model or a human anchor, not a second copy.

4 **The maker must never grade itself.**
Pam's backtest gets this right (separate subprocess, different model, only her text passed in). Her LIVE path does not (regex only). Mind the gap between the diagram and the running system.

5 **Verify against truth, not just against agreement.**
Two graders agreeing 90% of the time is not 90% correct. Ground truth only enters when a human rules. Keep a human gold set.

6 **"Done" needs an objective definition or it cannot loop.**
Code has tests. Judgment work needs a rubric you wrote. No rubric, no loop.

7 **Green does not mean understood.**
The loop ships faster than you read. You still read what it merges and what it sends.

8 **A no-budget loop on a cron is the billing-surprise risk.**
On your plan a runaway eats your 5-hour window, not your wallet. A halting token ceiling is cheap insurance; the ~18M-token backfill is the near-miss that proves it.

9 **Ambient agents need more gates as they get more capable.**
Jarvis in bypass mode is the highest-leverage and highest-blast-radius surface you own. Autonomy and checkpoints scale together.

10 **Skills compound, prompts don't.**
Your playbooks and learned-rules already are this. Keep feeding the loop named, tested recipes.

8. The team-shareable loop skill

/LOOP-STARTER, CONCEPT

A teammate runs it. It interviews them about a task they do, tells them honestly whether it is loop-able (can you write the pass condition?), identifies their maker / checker / stop / state, scaffolds the anchor files and the loop command, and runs a supervised dry-run. Because it reads their actual work, it customizes to them. Ships through your eg-skills install-once + daily-auto-pull pipeline. **Timing:** build it at MO-7, after Pam proves the pattern. Shipping an untested loop template company-wide is the over-application mistake at team scale.

---

9. What I need from you

Logged already: first loop = Pam, autonomy 75-80%, no hard dollar budget. Observability build removed. Remaining decisions, answer by number:

1. **Fix the silent grader tonight?** Want me to investigate why Ross's audit grader stopped and stand up an always-on fallback copy on your Mac.

My rec: yes. The loop is half-blind right now and you cleared heavy spend tonight.

2. **Start the top 3 Pam fixes?** Close the loop (auto-route), fix the bar (human-keyed + gold set), break correlated grader error.

My rec: yes, in that order. They are what move Pam from partial to a real closed loop and unblock 90%.

3. **Add a halting token ceiling to the Pam crons?** You said no dollar budget, and I agree on the dollar cap. This is different: a token ceiling that protects your 5-hour window from a runaway.

My rec: yes. It is the cheap insurance the 18M-token near-miss argues for.

4. **Jarvis: parallel or after Pam?** Build Track A (deterministic post-condition check) now alongside Pam, or wait until Pam's loop is closed.

My rec: Track A in parallel (it is low-risk and independent), Track B after Pam proves the eval pattern.

5. **Team /loop-starter skill timing?** Now, or after Pam is a proven closed loop.

My rec: after. Ship a proven template.

Generated by Claude. Pam + Jarvis findings verified against the live code (cst-bot, jarvis-agent, launchd plists, the Grading Audit sheet).

Level 1 / Static Doc