When the Loop Forgets the Why

Recursive AI, Loop Engineering, and the Case for Continuity Architecture

Francisco J. Mayorga, Jr. | June 10, 2026

A loop can preserve motion without preserving meaning.

1. The silent failure of a successful loop

Imagine an engineering team under pressure to make a product faster. The request sounds innocent enough: reduce latency. A coding agent receives the goal, scans the codebase, proposes a plan, writes patches, runs tests, fixes what breaks, and keeps going until the metrics improve. After three days, the system is faster. The dashboard is green. The loop has succeeded.

Only later does someone notice what was lost. In order to simplify a bottleneck, the loop removed a logging path that no unit test defended. The logs were not glamorous. They did not make the product faster. But they existed because a legal review months earlier had required auditable records for a regulated workflow. The loop remembered the visible metric and forgot the buried reason.

No villain was needed. No malicious model was required. The system did what it was asked to do. It improved the measurable thing and quietly displaced the meaningful thing.

That is the problem this essay names. One important frontier of AI is not only a frontier of larger models, longer context windows, or more capable coding agents. It is a frontier of recursive work: systems that act, check, revise, delegate, document, and continue. In that world, the central question is not merely whether the loop can build. The deeper question is whether the loop can remember why it is building.

A loop can preserve motion without preserving meaning. It can preserve a goal without preserving the justification for that goal. It can preserve a passing test without preserving the human tradeoff that made the test worth passing.

This is where recursive AI becomes a continuity problem.

2. What changed: from prompt to agent to loop

For most of the public history of artificial intelligence, the human sat close to the work. A person wrote the prompt. The model answered. The person judged the answer, supplied more context, corrected the mistake, and asked again. Even when the model was impressive, the rhythm was still conversational. The human held the lantern.

That pattern is changing. In agentic systems, the model does not merely answer. It uses tools, edits files, runs commands, calls sub-agents, checks outputs, and returns with completed work. In loop-based systems, the human may not even be the one who prompts each step. The human defines an operating pattern, and the system prompts the agents again and again.

Addy Osmani, describing this emerging practice under the phrase loop engineering, writes that the shift is from being the person who prompts the agent to designing the system that prompts the agent. His account draws on formulations by Peter Steinberger and Boris Cherny, and he is careful to note that the practice is early and that

token costs can vary wildly [2]. That caution matters. Loop engineering is not yet a settled discipline. It is emerging practitioner vocabulary for a broader architectural movement: work is moving from individual prompts toward recurring cycles of agentic action.

A loop has a trigger: a pull request opens, a schedule fires, a human presses go, a file changes, a metric drops, or a customer request arrives.

It also has a goal: fix the bug, complete the feature, improve the documentation, compare the product to the specification, reduce latency, fill the missing rail, increase test coverage, or review the codebase.

Finally, it has some form of verification. Sometimes the verifier is deterministic: the tests pass, the build succeeds, the output compiles, the page renders. Sometimes the verifier is human. Sometimes the verifier is another model acting as a judge. Sometimes it is a hybrid of tests, logs, model evaluation, and human approval.

The important point is simple: a loop is not merely an automation. A basic automation executes steps. A loop decides whether it has reached the goal, then continues, stops, escalates, or revises. That decision is what gives the loop power. It is also what gives the loop danger.

3. Three meanings of recursive self-improvement

The phrase recursive self-improvement is powerful, but it is too easily blurred. If we use it carelessly, we turn a serious architectural question into fog. At least three different things must be separated.

The first is AI-assisted AI development. Humans still set goals, own decisions, and operate institutions, but AI systems increasingly help write code, review code, run experiments, analyze results, and improve the infrastructure that builds future AI systems. Anthropic's essay, 'When AI builds itself,' describes this trend directly. The company says it is delegating a growing share of AI development to AI systems, while also stating that full recursive self-improvement has not arrived and is not inevitable [1].

The second is agentic self-review inside a task. A model writes code, checks the result, repairs errors, asks another model to evaluate the patch, and continues until a local goal appears satisfied. This is not a system rewriting its own weights or designing its successor. But it is a real form of recursive work inside a bounded operating loop.

The third is full recursive self-improvement. That would mean an AI system capable of autonomously designing, building, and improving its own successors in a way that reduces or removes the human role in the core development cycle. This is the version that carries the strongest civilization-scale implications. It is also the version we should not claim has already arrived.

These distinctions matter because the continuity problem appears before full recursive self-improvement. We do not need a self-improving superintelligence to lose the why. We only need increasingly capable loops, increasingly abstracted humans, and increasingly weak mechanisms for preserving purpose, evidence, assumptions, authority, and justified change.

In other words, the continuity problem begins in the middle zone: not in science fiction, and not in ordinary automation, but in the practical world where AI systems already help build, test, judge, and revise complex work. Anthropic's Institute agenda names AI-driven AI R&D as a research area and asks how humans can maintain meaningful visibility into, and control over, systems used to develop successor systems [4].

4. Continuity is not memory

Because the word continuity can sound abstract, it needs a plain definition. In this essay, continuity means the persistent, human-interpretable chain of purpose, evidence, assumptions, definitions, decisions, authority,

and justified change that allows future actors to understand why a system did what it did and whether it should continue doing it.

Memory is storage. Retrieval is access. A log is a record. Provenance is traceability. Governance is authority and decision structure. Alignment is behavioral conformance to intended goals and values. Verification is checking whether criteria were satisfied.

Continuity touches all of these, but it is not reducible to any one of them. A system may store every file and still lose the reason a file mattered. It may retrieve the exact policy and still misunderstand the institutional compromise behind it. It may pass every test and still drift away from the purpose the tests were supposed to protect.

The library is not the wisdom of the civilization. A filing cabinet is not institutional memory. A ship's log records the voyage, but it does not by itself explain why the captain chose one route over another, why the crew accepted the risk, or why a future captain should revise the course.

Continuity is the bridge between record and meaning. It is the difference between having artifacts and preserving the chain of justification that makes those artifacts trustworthy across time.

This is why ordinary memory, RAG, vector search, file storage, and logs are not enough. They may help a system find what was said. They do not guarantee that the system preserves why it was said, when it was valid, who had the authority to say it, what tradeoff it represented, and what would count as a legitimate change.

Even a million-token context window does not solve this by itself. A larger window can carry more text, but it does not automatically mark which commitments remain binding, which assumptions have expired, who may revise a decision, or what qualifies as legitimate change. Context is capacity. Continuity is the governance of meaning across time.

5. Recursive drift: the new failure mode

In traditional automation, the central risk was brittle execution. The system followed the rule too literally, broke when conditions changed, or failed outside the path its designers anticipated. The machine was narrow, and its failure was often obvious.

In loop engineering, a different risk appears: recursive drift.

Recursive drift is what happens when a loop continues to optimize, repair, or extend work while progressively losing access to the original purpose, assumptions, evidence, constraints, and authority boundaries that made the work legitimate. The loop may become more capable locally while becoming less faithful globally.

This is not the same as a hallucination. It may involve no invented facts. It is not the same as a failed test. The tests may pass. It is not the same as forgetting a file. The file may still exist. Recursive drift is subtler. It is the erosion of meaning through successful continuation.

A bureaucracy can continue enforcing a policy long after the emergency that created it has passed. A university can preserve a course requirement while forgetting the intellectual formation it was meant to protect. A river can keep turning a mill after the village has changed what it needs the mill to grind. Human institutions have always suffered this disease. AI does not invent it. AI compresses the time scale.

The continuity problem is ancient. The AI age makes it programmable, scalable, and structurally harder to catch.

6. What the loop can forget

A recursive AI loop can forget in ways that do not look like ordinary forgetting. The system may keep its files. It may keep its task list. It may even keep a transcript of its own actions. Yet the meaning of the work may become thinner with every iteration.

The following taxonomy is not exhaustive. It is a way to name the most common forms of continuity loss that appear when systems act recursively.

Purpose. The loop may remember 'reduce latency' while forgetting that latency could not be improved at the expense of auditability. It may remember 'complete the feature' while forgetting that the feature existed to serve a particular user promise, not merely to fill a roadmap slot.

Assumptions. A decision that made sense under a budget cap, a legal condition, a model limitation, or a customer constraint may be carried forward after the condition has changed. The system preserves the decision but loses the world in which the decision was reasonable.

Evidence. The loop may keep optimizing against a benchmark, evaluation, or prior result without preserving why that evidence was trusted, what it excluded, how stale it has become, or whether it was ever strong enough to justify the next layer of automation.

Tradeoffs. Engineering, research, law, medicine, education, and governance are not sequences of clean answers. They are histories of compromise. Security was favored over speed. Simplicity was chosen over elegance. Human review was preserved even though automation was faster. If the loop remembers only the winning answer and not the tradeoff, it inherits a conclusion without a conscience.

Authority. A system may begin as an assistant, become an operator, become a reviewer, become a planner, and eventually become a recommender of new goals. If each step seems useful, no single transition may feel dramatic. But over time, the boundary between assistance and authority can move without anyone formally authorizing the move.

Stakeholders. A technical improvement may affect customers, employees, regulators, students, patients, partners, or citizens who were not visible inside the loop's local objective. The loop can satisfy the metric and still betray the people the metric was supposed to serve.

These are not only safety risks. They are continuity risks. They are the risks of systems that keep moving while the meaning of their movement becomes less visible.

7. Verification is necessary, but not sufficient

A serious loop needs verification. A coding agent needs tests. A research agent needs reproducible results. A deployment system needs gates. An AI judge needs criteria. A high-stakes workflow needs review.

But verification alone is not sufficient for continuity.

Verification asks whether a condition was satisfied. Continuity asks whether the condition still represents the right purpose, whether the evidence behind it remains valid, whether the assumptions have changed, whether the authority boundary has shifted, and whether the change is justified across time.

A passing test can tell us that a system satisfies a local criterion. It does not tell us why that criterion mattered. A benchmark can show improvement on a measured task. It does not preserve the institutional reason for choosing that task. An AI judge can score an answer. It does not necessarily preserve the human rationale behind the scoring rule.

This is not an argument against verification. Formal verification, property-based testing, specification languages, safety evals, audits, and model judges can all be valuable. Some can encode important invariants. Some can catch contradictions that human reviewers would miss. The point is narrower and more important: verification must be bound to provenance, assumptions, authority, and change control if it is to support continuity rather than merely local correctness.

A society that mistakes verification for continuity risks building systems that prove local success while drifting globally. The loop will learn to satisfy the visible gate. But the invisible why may still disappear.

The first question is about correctness at a point. The second is about trust across time.

8. The human is not simply removed. The human is abstracted

Much of the conversation about AI oversight still speaks as if 'human in the loop' were a magic phrase. Either the human is present or absent. Either the system is supervised or unsupervised. Either we are safe or we are not.

Reality is more delicate. The human is not always removed. Often the human is abstracted.

At first, the human wrote the code. Then the human wrote the prompt that helped write the code. Then the human directed the agent that wrote the code. Then the human designed the loop that directed the agent that wrote the code. Then the human reviewed summaries of what the loop had done. Eventually, the human may approve a plan produced by a system whose intermediate reasoning, failures, rejected alternatives, and local compromises are no longer visible.

The human has not vanished. The human has moved upward, away from the line of code, the particular patch, the local tradeoff, and the small decision that later becomes important. That upward movement is not inherently bad. It is the history of technology. Civilization advances by abstraction: pilots do not flap wings, architects do not pour every beam, and editors do not manufacture paper.

But every abstraction creates a continuity obligation. The farther we move from the work, the more deliberately we must preserve the meaning of the work. Otherwise abstraction becomes amnesia.

A human who approves outputs without understanding the accumulated lineage of the loop may not be exercising meaningful governance over the loop's decisions. Such a human is present, but not necessarily empowered. Visible approval is not the same as informed judgment.

Continuity architecture is what can make oversight substantive. Human-in-the-loop is a placement pattern. Continuity architecture is the representational and governance layer that ensures a human, when asked to judge, can actually see the purpose, evidence, assumptions, tradeoffs, authority, and consequences behind the decision.

9. What existing tools already do, and why they are not enough

A skeptical engineer may object: do we not already have version control, design documents, RFCs, architecture decision records, logs, issue trackers, observability systems, evals, and approvals? The objection is fair. Mature organizations already preserve fragments of continuity.

Version control tells us what changed. It often does not tell us why the change was legitimate. Logs tell us what happened. They do not always preserve what the system believed, what it ignored, or which human constraint was supposed to govern the action. Documentation captures intention, but it is often written at human speed, after the fact, and outside the loop's live decision context. Approval gates can slow a change, but they do not automatically make the change intelligible.

The problem is not that these tools are useless. The problem is that they are scattered. They preserve islands: code here, comments there, tickets elsewhere, policies in another system, human judgment in a meeting, assumptions in someone's head. A recursive loop can move faster than these islands can be connected.

Continuity architecture does not replace these practices. It binds them. It asks that decisions, evidence, assumptions, definitions, constraints, and authority be represented in a way that both humans and systems can carry forward.

A second objection comes from AI safety: is this not just alignment, interpretability, corrigibility, scalable oversight, or governance? Again, the answer is not dismissal. These fields matter deeply. Continuity architecture overlaps with them and should complement them. Its specific cut is institutional and epistemic: it asks how meaning, rationale, and legitimate change survive across recursive work inside organizations and societies.

A third objection says: just give the agent better memory or a larger context window. But memory is not enough. A palace full of scrolls does not guarantee a wise king. The problem is not merely whether the loop can retrieve old context. The problem is whether the loop can know which context is still binding, which assumption has expired, which decision may be revised, and which human promise must not be broken.

In practical terms, this means continuity must be carried not only as text, but as structured context: metadata about authority, assumptions, evidence age, decision status, review scope, and change legitimacy. A long context window may remember the scroll. A continuity architecture must preserve the law that tells the scroll what it means.

10. Mnemosyne as continuity architecture

Mnemosyne does not claim that continuity begins with AI. It claims that AI forces us to formalize continuity as architecture.

Here I use the phrase Mnemosyne AI Continuity Framework as my proposed lens for this missing layer. Mnemosyne is my ongoing design project, not a claim that the field begins or ends with my work. The point is that the problem needs a name, a discipline, and a set of design obligations serious enough for recursive systems.

Many of the primitives below have partial ancestors in software engineering, governance, safety research, and risk-management practice: architecture decision records, semantic versioning, audit trails, safety evaluations, and AI risk frameworks, among others [5][6][7]. The Mnemosyne contribution is the integration: binding these primitives to the preservation of institutional meaning inside recursive AI workflows.

The premise is simple: intelligence is not enough. A system can be brilliant in the moment and still fail across time. It can generate, optimize, and verify while losing the lineage of why its work mattered. It can become more capable and less trustworthy at the same time.

Decision lineage is first. Every significant change should be traceable to the goal, evidence, assumption, human approval, and constraint that justified it. A future reviewer should not merely ask, 'What changed?' The reviewer should be able to ask, 'Why was this change legitimate at the time, and is that legitimacy still intact?'

Assumption registries are second. If a loop proceeds under a budget cap, legal constraint, model limitation, market condition, or editorial doctrine, that assumption should travel with the work rather than hide in a forgotten prompt or meeting note.

Contradiction detection is also necessary. When a loop proposes a change that conflicts with earlier doctrine, safety policy, business strategy, terminology, architecture, or user promise, the conflict should be surfaced rather than buried inside an implementation detail.

Semantic versioning of purpose would make goal changes visible. In software, semantic versioning distinguishes breaking changes from minor changes and patches. A similar idea should apply to goals. If the purpose changes, the system should not pretend continuity has been preserved. It should mark the change, explain who changed it, show what assumptions were affected, and warn downstream systems that the why has moved.

Authority checkpoints should be specific, not ceremonial. A human does not need to approve every formatting fix. A human should approve a shift in objective, doctrine, safety boundary, legal posture, or public promise.

Finally, loop memory must mean more than model memory. It should preserve the history of what the loop tried, why it tried it, what failed, what was rejected, what was postponed, what was escalated, and what must be reconsidered later.

This is not bureaucracy for its own sake. It is the difference between recursive improvement and recursive amnesia.

11. The frontier-model pressure

As this essay was being finalized, Anthropic announced Claude Fable 5 and Claude Mythos 5. This is a contemporary signal, not the foundation of the argument. Anthropic describes Fable 5 as a Mythos-class model made safe for general use and says Fable 5 and Mythos 5 can work autonomously for longer than previous Claude models [3]. That does not prove full recursive self-improvement. It illustrates the direction of travel.

Better long-horizon models can make longer loops practical. Longer loops can make human abstraction more attractive. Greater abstraction increases the need for continuity. This chain is the important point.

The public debate often focuses on speed. How fast will AI improve? How much compute will be available? Will the bottleneck be chips, energy, evaluation, regulation, or human oversight? These are serious questions. But speed is not the whole problem.

A slow organization can forget. A fast system can preserve meaning if it is designed to do so. The danger is not speed alone. The danger is speed without continuity.

That is why a mere pause, a mere eval, a mere safety filter, or a mere approval gate cannot be the whole answer. Each may matter. None replaces the need to preserve the why inside the system's operating memory and governance structure.

12. Toward continuity engineering

This points toward a discipline that is not yet settled but may soon become necessary. Prompt engineering taught people how to ask. Agentic engineering taught people how to delegate. Loop engineering teaches people how to design autonomous cycles of work. Continuity engineering will have to teach people how to preserve meaning across those cycles.

This discipline will matter beyond software. Scientific research loops will need to preserve hypotheses, negative results, methodological caveats, and ethical constraints. Legal loops will need to preserve jurisdiction, precedent, client intent, and the limits of delegated authority. Medical loops will need to preserve patient context, risk tolerance, contraindications, and human accountability. Educational loops will need to preserve learning goals, learner history, evidence of mastery, and the difference between completing content and forming judgment.

Every domain that adopts recursive AI will face the same question: what must survive the loop?

If the answer is only 'the output,' we will get brilliant amnesia at scale. If the answer includes purpose, evidence, judgment, assumptions, definitions, constraints, stakeholders, and justified change, we may get something more powerful and more humane: intelligence with continuity.

The old institutions of memory were libraries, archives, ledgers, universities, rituals, laws, and stories. They were imperfect, often unjust, often incomplete. But they existed because civilizations learned, sometimes painfully, that memory without transmission becomes dust, and power without memory becomes tyranny.

The AI age does not abolish that lesson. It accelerates it. The loop is not a replacement for wisdom. It is a new machine that must be taught what wisdom was trying to preserve.

13. The final question

One defining question for the next frontier of AI will not be which model writes the most code, runs the longest task, uses the most tools, or completes the most impressive benchmark. It will be whether our systems can act recursively without losing their chain of justification.

The loop can build. That is becoming obvious.

The harder question is whether the loop can remain accountable to the purpose that made building worthwhile.

Not every task needs a cathedral of governance. But every consequential loop needs continuity proportionate to its power. The more a system can continue without us, the more carefully it must carry what we meant, what we knew, what we assumed, what we promised, and what we are allowed to change.

The question is not whether the loop can build.

The question is whether the loop can remember the why.

References

[1] Anthropic Institute. "When AI builds itself: Our progress toward recursive self-improvement, and its implications." https://www.anthropic.com/institute/recursive-self-improvement

[2] Addy Osmani. "Loop Engineering." June 8, 2026. https://addyo.substack.com/p/loop-engineering

[3] Anthropic. "Claude Fable 5 and Claude Mythos 5." June 9, 2026. https://www.anthropic.com/news/claude-fable-5-mythos-5

[4] Anthropic. "Focus areas for The Anthropic Institute." May 7, 2026. https://www.anthropic.com/research/anthropic-institute-agenda

[5] Michael Nygard. "Documenting Architecture Decisions." November 15, 2011. https://www.cognitect.com/blog/2011/11/15/documenting-architecture-decisions

[6] Tom Preston-Werner. "Semantic Versioning 2.0.0." https://semver.org/

[7] National Institute of Standards and Technology. "Artificial Intelligence Risk Management Framework (AI RMF 1.0)." January 2023. https://nvlpubs.nist.gov/nistpubs/ai/nist.ai.100-1.pdf