The Recursive Governance Loop

A GMP-Inspired Framework for Self-Correcting AI Orchestration

Author: Flow State Project Team **Date:** 2026-01-04 **Status:** Living Document

---

Abstract

This document describes an approach to AI agent orchestration that borrows from Good Manufacturing Practice (GMP) principles. The core insight is recursive: a Quality Management System (QMS) governs AI agents as they develop code, but also governs its own evolution—creating a feedback loop where process failures become inputs to process improvement. Rather than attempting to design the perfect orchestration framework upfront, we treat AI collaboration as an empirical problem and let the system learn from its own mistakes.

---

1. The Problem: Capable but Unconstrained

Modern AI agents are remarkably capable. They can read codebases, propose changes, execute complex multi-step tasks, and collaborate across specialized domains. But this capability comes with a risk: **AI agents tend to do too much**.

They are eager to please. They solve problems you didn't ask them to solve. They make changes before you've validated the direction. They compound small misunderstandings into large deviations. And they leave you with a *fait accompli* that's harder to undo than to have done correctly in the first place.

Traditional software development workflows assume human-paced collaboration—pull requests that sit for hours, code reviews that happen asynchronously, meetings to discuss architecture. AI agents operate at machine speed. Without deliberate constraints, they can get far ahead of human oversight, making course correction expensive or impossible.

The need: A framework that provides checkpoints for human oversight without stifling the capability that makes AI agents valuable in the first place.

---

2. Why GMP? The Unexpected Template

Good Manufacturing Practice (GMP) is a system of procedures, documentation, and oversight used in pharmaceutical manufacturing and medical device production. It exists because errors in those domains can kill people. Every change is documented. Every document is reviewed. Every review is recorded. Nothing happens without approval.

At first glance, this seems like bureaucratic overkill for software development. But the core principles map surprisingly well to AI orchestration:

GMP Principle	AI Orchestration Application
Stage gates	Human approval checkpoints before AI proceeds

GMP Principle	AI Orchestration Application
Multi-party review	Multiple specialized agents provide diverse perspectives
Audit trails	Full visibility into AI decision-making and rationale
Controlled documents	Prevent AI agents from unilaterally modifying critical files
Change control	Every modification requires explicit authorization
Investigation/CAPA	Process failures trigger systematic analysis and improvement

We're not importing bureaucracy. We're importing **deliberation before action**—the principle that consequential changes should be articulated, reviewed, and approved before execution.

---

3. The Two-Layer Architecture

The QMS operates on two layers simultaneously.

Layer 1: Code Governance

The primary function—governing how code gets written and changed:

- **Change Records (CRs)** authorize modifications to the codebase
- **Technical Unit (TU) agents** provide domain expertise (UI, simulation, geometry, physics)
- **QA agent** enforces procedural compliance
- **Stage gates** (pre-review, pre-approval, post-review, post-approval) provide explicit checkpoints where humans can intervene
- **Checkout/checkin** prevents AI agents from silently modifying controlled files

This layer ensures that code changes are deliberate, reviewed from multiple perspectives, and approved before execution.

Layer 2: Process Governance (Meta)

The recursive function—the QMS governs its own evolution:

- **Investigations (INV)** formally analyze process failures
- **Corrective actions** fix immediate issues identified by investigations
- **Preventive actions** modify procedures to prevent recurrence
- The QMS improves itself **using its own mechanisms**

When something goes wrong with the process, the same document control, review, and approval workflows that govern code changes are used to improve the process itself. The system learns from its own failures.

---

4. The Recursive Loop in Action

This isn't theoretical. The Flow State project demonstrated this loop in its first week of operation:

```
CR-005: Documentation fixes, QA permissions, and archive bug
    │
    │   [CR-005 closes with incomplete scope—SOP changes not executed]
    │
    ▼
INV-001: CR-005 Scope Deviation Investigation
    │
    │   [Investigation confirms procedural nonconformance]
    │   [Root cause: inadequate scope change control during execution]
    │
    ▼
CR-006: SOP-001 permission matrix update (corrective)
CR-007: SOP-002 ID reuse clarification (corrective)
    │
    │   [Immediate issues from CR-005 corrected]
    │
    ▼
CR-008: SOP-002 Section 7.3 post-review requirements (preventive)
    │
    │   [New requirement: documentation must be EFFECTIVE before CR post-approval]
    │   [QA must verify this during post-review]
    │
    ▼
Process is now more robust than before the failure
```

The process failed. The failure was investigated using the process. The investigation spawned changes to the process. The process improved. **The system learned.**

And critically: CR-008 was itself the first CR verified under the new Section 7.3 requirements it introduced. The recursive nature is literal—the improvement was validated by the improvement.

---

5. Why This Matters for AI Collaboration

AI orchestration is an open problem. No one has figured out the optimal way to coordinate multiple AI agents working on a shared codebase with human oversight. The approaches being explored—agent frameworks, tool-use patterns, memory systems, orchestration layers—are all experiments.

Rather than attempt to design the "perfect" framework upfront (which we don't know how to do), this approach treats AI orchestration as an **empirical problem**:

1. **Establish a baseline process** (the QMS)
2. **Use it to do real work** (develop the Flow State application)
3. **When it fails, investigate** (INV documents)
4. **Improve based on evidence** (corrective and preventive CRs)
5. **Repeat**

The QMS provides:

- **Safe experimentation** — Changes are controlled, reversible, and auditable
- **Empirical validation** — What works gets kept; what doesn't gets revised

- **Continuous improvement** — Every failure is an input to a better system

We're not claiming to have solved AI orchestration. We're claiming to have built a system that can **learn how to orchestrate AI agents** through structured trial and error.

---

6. The Constraints That Enable

A common objection: "Doesn't all this process slow things down?"

Yes. That's the point.

The constraints exist precisely because AI agents are fast. Without them, an agent can make dozens of changes before a human notices something is wrong. The stage gates create **forced reflection points** where:

- The agent must articulate what it intends to do (CR creation)
- Other agents must assess the proposal from their domains (TU review)
- A procedural authority must verify compliance (QA review)
- A human can approve, reject, or redirect (stage gate)

Mechanism	What It Prevents
Checkout/checkin	Silent modification of controlled files
Pre-review gate	Execution before articulation
Multi-agent review	Single-perspective blind spots
Stage gates	AI getting too far ahead of human oversight
Edit/Write deny rules	Over-zealous agents bypassing controls
Audit trail	Untraceable decisions

These constraints don't prevent AI agents from being useful. They prevent AI agents from being useful in ways you didn't want.

---

7. The Agents as a Simulated Organization

The QMS creates, in effect, a **simulated organization** of AI agents with defined roles:

Agent	Role	Perspective
Claude	Initiator	Primary implementer; creates and executes CRs
QA	Quality Assurance	Procedural compliance; mandatory reviewer
TU-UI	Technical Unit	User interface expertise
TU-SCENE	Technical Unit	Application lifecycle, orchestration
TU-SKETCH	Technical Unit	Geometry, constraints, CAD
TU-SIM	Technical Unit	Physics, particles, simulation

Agent	Role	Perspective
BU	Business Unit	User value, experience, "is this fun?"

Each agent has specific instructions focusing its attention on particular concerns. When a CR is reviewed, these agents examine it from their respective domains. A change that looks fine from an implementation perspective might raise concerns from a user experience perspective (BU) or reveal architectural issues (TU-SCENE).

This isn't just parallelization—it's **epistemic diversity**. Different prompts create different "viewpoints," and the stage gates ensure all viewpoints are heard before proceeding.

---

8. What We're Learning

Early observations from using this system:

On agent behavior:

- AI agents genuinely benefit from constraints; without them, they optimize for completion rather than correctness
- The checkout/checkin mechanism is essential—agents will modify files directly if allowed
- Stage gates catch issues that would otherwise compound across multiple changes

On process design:

- GMP patterns transfer surprisingly well, but need adaptation for machine-speed workflows
- The INV/CAPA pattern is powerful for process improvement; it makes failures generative
- Document proliferation is a real risk; the system must resist its own tendency toward bureaucracy

On human oversight:

- Stage gates provide natural intervention points without requiring constant monitoring
- The audit trail makes it possible to understand "what happened" after the fact
- Having explicit approval points reduces anxiety about AI agents "running away"

---

9. Conclusion

The QMS is simultaneously:

- **A governance framework** — Constraining AI agents to act within defined boundaries
- **A collaboration protocol** — Enabling multiple agents to contribute their perspectives
- **A laboratory** — Providing a controlled environment to experiment with orchestration patterns
- **A learning system** — Improving through its own investigation and correction mechanisms

The meta-insight: **The best way to figure out how to orchestrate AI agents is to build a system that can learn from its own failures.**

We don't know the optimal way to coordinate AI agents on a software project. But we've built a system that can discover it empirically—one investigation, one corrective action, one preventive measure at a time.

The recursive governance loop isn't just a feature of this approach. It's the point.

---

10. Final Thoughts: Three Deliverables from One Effort

At the end of this project, we will not just have code that works. We will have:

Layer	Deliverable	Value
Product	Code that works	The Flow State application
Method	A framework for producing code that works	A reusable AI orchestration pattern
Provenance	A historical record of how the framework was discovered	INVs, CAPAs, audit trails, chronicles

The third layer is the rarest. Most projects produce code. Some produce reusable methodologies. Almost none produce a traceable narrative of *how the methodology emerged*—the failures that triggered investigations, the investigations that spawned improvements, the improvements that prevented recurrence.

That provenance is:

- **Reproducible** — Others can follow the same discovery path
- **Auditable** — Every decision has a rationale in the record
- **Teachable** — The *why* is preserved alongside the *what*

The QMS isn't just governing the work. It's **documenting its own genesis**.

References

- SOP-001: Quality Management System - Document Control
- SOP-002: Change Control
- INV-001: CR-005 Scope Deviation Investigation
- CR-008: SOP-002 Post-Review Requirements
- Architectural Assessment of the Flow State QMS and a Proposal for Remediation

---

END OF DOCUMENT