PROFORMA GLOBAL RESEARCH

The Five Orchestration Patterns

Matt Rollings, Founder and Principal, Proforma Global

Whitepaper · 2026-05-18

Executive Summary

Background. The choice of orchestration pattern is the foundational architectural decision in any agent system. Most teams never make it consciously. They inherit whatever pattern their framework defaults to (usually dynamic agent-driven routing) and discover the limits of that pattern in production. There are five distinct orchestration patterns. Each has a class of problem it handles well and a class it cannot handle at all. Selecting the wrong pattern is the most expensive mistake in agent design, and it is the most common.

Subject. The five patterns are flat, iterative, agent-driven (dynamic), defined deterministic workflow, and recursive. The first three are well-documented and routinely deployed. The fourth is rare. The fifth is rare and routinely counterfeited: most systems labeled "recursive" are iterative loops with a counter, wearing the label. Recursive orchestration gets the most attention in this paper. It is the most under-deployed pattern, the most undervalued, and the one that mechanically prevents the failure mode described in the first paper in this series.

Approach. Each pattern is mapped onto four axes that determine fit: discipline count (single vs. multiple), scope bounding (predictable vs. open-ended), action reversibility (cheap vs. expensive to undo), and context requirements (narrow vs. broad). The axes give the architect a framework for matching a pattern to a problem instead of defaulting to whichever pattern the framework ships with.

Findings. No pattern is universally superior. Flat is right for bounded single-pass work. Iterative is right for batch transformation over a known input set. Agent-driven is right for exploration and conversational assistance. Defined deterministic workflow is right for multi-discipline production work where reliability and reproducibility matter. Recursive is right for layered work where context can be bounded per layer and failure can be addressed by collapsing back to the level with the context to recover. The pattern most consistently misapplied in the market is agent-driven dynamic routing, which gets used as a default for problems whose properties demand recursive structure or deterministic workflows. The pattern most consistently mis-implemented is recursion itself, which is shipped as an iterative loop and called something it is not.

Conclusions. The orchestration pattern determines outcomes more than the model determines outcomes. Choosing the pattern is an architectural responsibility that does not transfer to the model. It cannot, because the model itself is biased toward producing the wrong pattern when asked. Architects who choose explicitly produce systems that hold up in production. Architects who inherit the default produce systems that pilot, demo, and fail when reality arrives. The five-pattern map in Section 2 and the selection framework in Section 3 are offered as the means of choosing well.

---

1. The Choice Teams Skip

Every agent system makes a foundational architectural decision before it makes any other: how the work is orchestrated. The choice determines which class of problem the system can solve, what its failure modes will be, what it will cost to run, and how predictably it will behave. Every downstream decision (model selection, prompt design, tool integration, evaluation strategy) operates within the constraints set by that one architectural choice.

Most teams never make the choice. They adopt a framework, the framework defaults to one of the five patterns, and the team inherits the default. The choice gets made by whoever wrote the framework's documentation, for reasons that had to do with what was easy to demonstrate, not what would survive production. The team's first encounter with the choice happens after deployment, when the pattern hits the class of problem it cannot handle and the system fails in the specific way that pattern fails.

The wrong orchestration pattern is not fixable by changing models, better prompts, or more compute. These are the levers teams reach for first, because the alternative (re-architecting the orchestration) is expensive enough to feel disproportionate to a problem that "should" be solvable with a smaller intervention. It is not solvable with a smaller intervention. Orchestration is a structural decision, and the fix has to be structural too.

The five patterns covered here are not interchangeable. Each was developed for a specific shape of problem. Each has produced reliable systems when matched to its fit, and each has produced expensive failures when used outside it. The market does not distinguish between them clearly. Most public writing on agent architecture treats orchestration as a detail. It is the architectural decision.

The industry's current ambition is to build a universal orchestrator: a single agent that handles every task correctly, all of the time. The ambition is structurally fragile. Tasks have different shapes, and an orchestrator using one pattern across all shapes will fail on every shape that pattern was not designed for. A universal orchestrator built on dynamic agent-driven routing fails on multi-discipline production work; one built on a deterministic workflow fails on open-ended exploration. No single pattern fits the full surface area of real work.

The five patterns in this paper are the vocabulary required to design any orchestrator, universal or otherwise. A team building a single-purpose system needs to know which pattern fits the problem. A team building a meta-orchestrator that selects per task needs to know the patterns in order to select among them. Either way the knowledge is foundational. Skipping it produces a system that handles one shape of task and fails on the rest.

The remainder of this paper covers what each pattern is, what it does well, what it fails at, and how to choose among them. The longest treatment goes to recursive orchestration, because it is the pattern most under-deployed in practice, most under-described in the literature, and most directly capable of preventing the failure mode discussed in the first paper of this series.

---

2. The Five Patterns

The five patterns have visibly different control-flow shapes. The shape is the pattern; everything else is implementation detail.

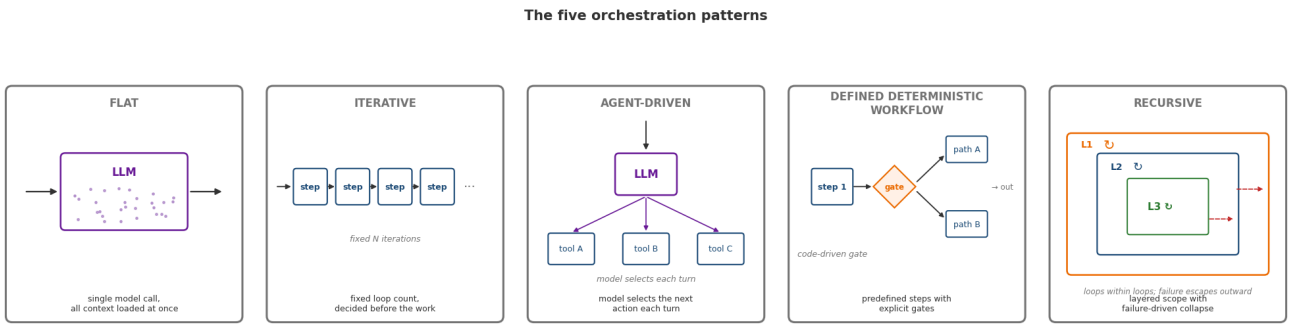

Figure 1: The five orchestration patterns for agent systems. Each pattern has a control-flow shape that is the pattern; everything else is implementation detail. Detailed treatment of each follows in §§2.1–2.5.

2.1 Flat

A single model call. One prompt, one output. No loop, no decomposition, no internal state. The model receives a question and returns an answer.

Flat orchestration is the right choice for bounded single-pass work where the answer fits in one shot. Classification, summarization, single-step extraction, simple translation, direct Q&A. Anything where the cost of getting it wrong is bounded and the cost of additional structure exceeds the benefit.

The failure mode appears when teams use flat orchestration for problems with steps. The model produces plausible output for cases it cannot actually handle, because it has no mechanism to flag the cases where its single pass was insufficient. The output looks correct. Downstream consumers treat it as correct. The cases that needed more structure surface as quiet errors at unpredictable points downstream.

2.2 Iterative

A fixed for-loop over a step or sequence of steps. The loop count is decided in advance, by the caller, not by the work itself. Each iteration runs the same logic against a different input or accumulates against a shared state.

Iterative orchestration is the right choice for batch transformation over a known input set. Process N records. Score N candidates. Run validation against N rules. Anything where the size of the work is determined by the input and each unit of work is independent.

The failure mode appears when teams use iterative orchestration for convergence problems: work where the right number of iterations depends on the work itself. The loop runs too few times and stops short of done. Or it runs too many times and produces a degraded result by overworking what was already finished. Iteration cannot detect completion, because the loop count was chosen before the loop knew anything.

2.3 Agent-Driven (Dynamic)

The model decides what step to take next. Tools are exposed; the model selects among them and assembles the next call based on its interpretation of the state. The pattern is sometimes called ReAct, tool-use, or autonomous agent orchestration. The control flow is determined turn-by-turn by the model.

Agent-driven orchestration is the right choice for exploration, research, conversational assistance, and any problem where the path through the work cannot be enumerated in advance. The pattern works because the model's reasoning ability is being applied to the planning of the work, not just the execution of it. For open-ended single-discipline work, this is the correct shape.

The failure mode is the one described at length in the first paper of this series. The model carrying the full state of an open-ended decision process accumulates context until guardrails get crowded out and hallucination begins. For multi-discipline complex work, this is the pattern most consistently misapplied as a default. It is the right pattern for the wrong problem set in most production deployments today.

2.4 Defined Deterministic Workflow

A predefined sequence of steps with explicit branches, gates, and state. Each step has a defined input and output contract. The workflow decides what happens next based on the data flowing through it, not based on a model's interpretation. Language models may be invoked inside steps but do not choose the next step.

Defined deterministic workflows are the right choice for multi-discipline production work where reliability, reproducibility, and auditability matter. They hold the cross-discipline state that an orchestrator would otherwise have to carry. They produce the same output from the same input. They survive changes in model versions because the structure does not depend on model behavior.

The failure mode appears when teams use workflows for problems whose shape was not knowable in advance. Novel-problem investigation. Open-ended research. Exploration where the next step depends on conditions the workflow author could not anticipate. Workflows do not extend to handle cases their author did not design for; they fail at the boundary of their definition.

2.5 Recursive

This is the pattern most undervalued in agent design, most counterfeited in implementation, and most directly capable of producing the kind of system the first paper in this series argues for. It is treated at length because most public writing on agent architecture either skips it entirely or misdescribes it as a control-flow trick.

Recursive orchestration is a structural pattern with three properties working together.

The first is **layered scope.** The orchestration is organized in layers, each with a defined responsibility and a bounded context appropriate to that responsibility. The top layer carries the goal. Middle layers carry decomposition. Leaf layers carry the actual work. No single layer holds the entire workflow's context: the layering is what bounds it.

The second is **dynamic context assembly.** At each layer the prompt is assembled from only what that layer needs. Going one layer deeper trims away context the deeper layer does not require. Going one layer up adds context the higher layer holds. The prompt at any given step is bounded by the scope of that step, not by the total state of the workflow. The instruction-loss failure mode that breaks dynamic agent-driven orchestration cannot occur at the same scale in a recursive system, because the structure mechanically prevents any single call from carrying everything.

The third is **failure-driven collapse.** When a step at depth N fails, the failure does not get retried at the same level. It propagates up the stack to the layer that holds the context required to address that specific failure type. That layer enriches the context with what the failure revealed, then re-invokes downward. The system has explicit knowledge of which layer recovers which failure type. This is the property most "recursive" agent designs miss. They retry at the same level with the same context and call it recursion. Real recursion has the collapse behavior. *Fail here, recover there, re-invoke with enriched context.*

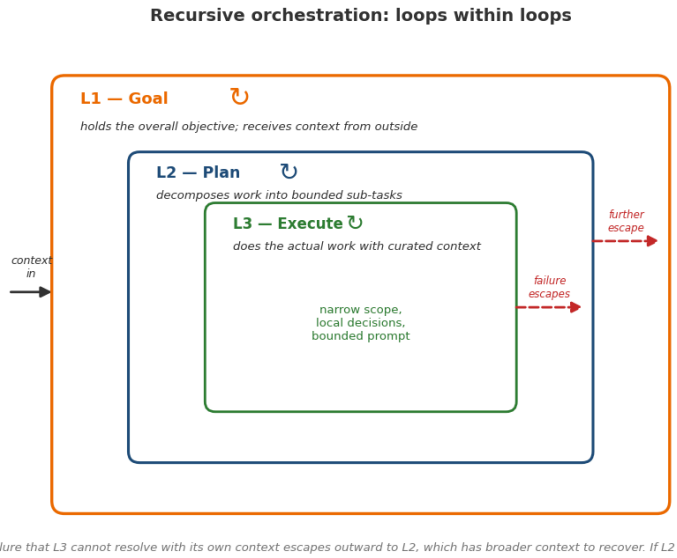

Figure 2: Recursive orchestration is loops within loops. Each layer carries its own internal flow over a bounded context appropriate to its responsibility. A failure that L3 cannot resolve within its own context escapes outward to L2, which has broader context. If L2 cannot resolve it either, the failure escapes again to L1. The prompt at any single layer never carries the entire workflow state. The structure mechanically prevents it.

Recursive orchestration is the right choice for any multi-step problem with potential for partial failure where the steps can be cleanly layered. Refinement work. Decomposition where each piece can be attempted and recovered independently. Investigation and diagnosis. Reasoning that benefits from being able to back up and try again with more context. Most multi-discipline production agent work falls in this region.

The failure mode appears in two places, both of them implementation failures rather than pattern failures. The first is the for-loop counterfeit: a loop with a counter, dressed in the language of recursion, missing all three structural properties. The second is the missing collapse pattern: a real layered structure that retries at the failing level instead of collapsing to the layer with the context to recover, producing oscillation or quality plateau across iterations. Both are common. Neither is recursion.

The pattern is under-deployed because the tools teams use to design agent systems do not produce it. Language models asked to design orchestration default to iterative loops. Most agent frameworks default to dynamic routing as their control flow. The discipline to design real recursive orchestration has to be imposed by the architect, against the grain of every default. The systems that get it right are not common. The ones that do are noticeably more reliable, more cost-efficient, and more capable of handling the multi-discipline production work that defeats the other patterns.

---

3. A Selection Framework

The five patterns are not interchangeable. Each fits a specific shape of problem, and the shape can be described by four axes. An architect choosing an orchestration pattern works through the axes, locates the problem on each, and reads the pattern that fits the combination.

Axis 1: Discipline count

How many independent bodies of knowledge does the task touch.

- **Single discipline.** Flat, iterative, agent-driven, and recursive all work. The pattern choice is driven by the other axes.
- **Multiple disciplines.** Flat and iterative fail by construction. Agent-driven fails for the reasons in the first paper of this series. The viable patterns are defined deterministic workflow and recursive.

Axis 2: Scope bounding

Is the path through the work knowable in advance.

- **Bounded path.** The work has a defined shape and the steps can be enumerated. Flat, iterative, or defined deterministic workflow apply. Recursion also applies when the layering is knowable.
- **Open-ended path.** The work requires deciding the path as it goes. Agent-driven dynamic orchestration is the only pattern that handles this honestly; recursion can apply when the layering itself is stable and only the leaf work is unbounded.

Axis 3: Action reversibility

What does it cost to undo a wrong step.

- **Cheap to undo.** Reruns are tolerable. The worst case is wasted compute. Any pattern works; the choice is driven by the other axes.
- **Expensive to undo.** The action modifies shared state, deploys to production, or feeds systems that cannot be told to ignore what they just received. Flat is dangerous because it has no completion check. Iterative is dangerous because it has no convergence detection. Agent-driven is dangerous because it has no structural prevention of cascading repairs. The safe choices are defined deterministic workflow and recursion with proper failure-driven collapse.

Axis 4: Context requirements

How much context does the work require to coordinate.

- **Narrow context.** A single call can hold everything that matters. Flat fits. So does any other pattern; the architect has more freedom here.
- **Broad context.** The work requires more state than a single prompt can carry without crowding out the instructions. This is the threshold described in the first paper of this series. The only patterns that survive at scale are defined deterministic workflow (state lives outside the prompt) and recursive (state is partitioned across layers).

Combining the axes produces the selection. Multi-discipline, bounded path, expensive to undo, broad context is the region where defined deterministic workflows belong. Multi-discipline, open-ended path, expensive to undo, broad context is the region where recursive orchestration belongs. Single-discipline, open-ended path, cheap to undo is where agent-driven dynamic shines. Single-discipline, bounded path, narrow context is where flat or iterative are correct.

Figure 3: Pattern fit by problem property

Problem property	Flat	Iterative	Agent-driven	Workflow	Recursive
Single discipline	✓	✓	✓	✓	✓
Multiple disciplines	✗	✗	✗	✓	✓
Bounded path	✓	✓	partial	✓	✓
Open-ended path	✗	✗	✓	✗	partial
Cheap to undo	✓	✓	✓	✓	✓
Expensive to undo	✗	✗	✗	✓	✓
Narrow context	✓	✓	✓	✓	✓
Broad context	✗	✗	✗	✓	✓

A ✓ means the pattern handles the property reliably at production scale. A ✗ means the pattern fails on that property in a way that surfaces downstream. "Partial" means the pattern handles a subset of the property's range (recursion fits open-ended work when the layering itself is stable and only the leaf work is unbounded).

The architect's job is to locate the problem on the axes before choosing the pattern. Most teams choose the pattern first and try to bend the problem to fit. The bending fails because the pattern was the wrong fit before the work began.

4. The Default Pattern Problem

The agent industry runs on two defaults. The first is agent-driven dynamic orchestration for production work. The second is iterative loops dressed as recursion for anything that needs to refine or repair its output. Both defaults are wrong for most of the problems they are applied to.

Agent-driven dynamic orchestration is correct for exploration and research. It is being deployed by default for multi-discipline production work, where it produces the failure mode described in the first paper of this series. The pattern itself is fine; the default applies it to

problems it was never designed for. Most production enterprise agent work is multi-discipline, expensive to undo, and broad in context. None of those properties favor dynamic routing.

The iterative-as-recursive default is the more insidious of the two. A team intending to build recursive structure produces a for-loop with a counter. The structure looks right from the outside. The collapse behavior is missing. The dynamic context assembly is missing. The layered scope is missing. What gets built is iteration with the wrong label, and it fails in the ways iteration fails (quality plateau, oscillation, no convergence) while the team believes they are running recursion. The diagnostic step that would catch this, checking whether the system actually has failure-driven collapse, is rarely performed because the assumption that the code is recursive prevents the question from being asked.

Both defaults persist because the tools used to design agent systems reinforce them. Frameworks default to dynamic routing. Models asked to design orchestration produce iterative loops. The discipline to choose the right pattern, and to implement it correctly when the right pattern is recursion, has to be imposed by an architect who is paying attention to a category of decision the rest of the industry treats as a detail.

The pursuit of a universal orchestrator that handles every task correctly is the same mistake at scale. A universal orchestrator has to pick one pattern. That pattern is wrong for most of the work. The universal system is fragile in exactly the way the single-pattern default is fragile, because it is the single-pattern default applied uniformly across a surface area it was never designed to cover. The discipline that produces reliable agent systems is the same whether the system handles one task or many: choose the orchestration shape per problem, compose where necessary, and resist the temptation to build a single pattern that does everything.

Figure 4: Composed orchestration. A workflow runs a defined sequence of steps; each step holds its own recursive substructure with bounded context and internal failure recovery; the workflow itself permits stepping back when a later step surfaces a problem the earlier step needs to address. Real systems compose patterns this way. They rarely pick one.

There is no such pattern. The five shapes are the alphabet. Reliable systems are sentences.