SINGULARICS

White Paper

The Fleet Lead and the Agent Fleet

Judgment stays human. Production does not. How tiny teams command AI fleets.

Alexander S. Petty
SINGULARICS | singularics.ai

First published August 2026.

singularics.ai

Contents

Abstract

When execution cost falls toward zero, the constraint moves to human judgment. The Decision Owner holds business judgment and accepts the work every day. The Fleet Lead holds the other half: technical judgment and the responsibility to direct the agent fleet that does the building. Together they form the human core of a team that can be as small as two people commanding dozens of agents. The paper defines the Fleet Lead role, the four skills that distinguish it from a senior engineering role with concrete examples of each, the three agent types and why their separation is load-bearing, the agent convergence problem, how fleet capacity is planned as a real constraint, and the precise boundary between the Fleet Lead and the Decision Owner. The Fleet Lead is the role that cannot be eliminated at any agent capability level, because it holds knowledge about intent rather than about code.

The Fleet Lead

The fastest path to understanding this role is to watch what happens without it. In one early engagement, an organization gave a capable agent fleet directly to business analysts and told them to build. The fleet produced 4,000 lines of working code in two days. None of it integrated with the existing system. The API contracts did not match. The data layer used a pattern the codebase had abandoned a year earlier. The harness tests all passed, because the agent wrote tests against its own implementation. Two days of output, zero days of value. The missing element was not capability. It was the technical judgment to direct that capability toward something that fits.

One to three in a crew, the small human complement wrapped around a fleet, the human component of the fleet itself. They are fleet drivers, knowledge-holding dispatchers of agentic coders. They specify the technical approach, drive the agent fleet, and hold technical judgment. Their scarce contribution is not code. It is knowing when the fleet has produced something that looks right and is not.

The shift is worth stating plainly, because it is easy to miss. In a traditional team the senior engineer's value was in what they could build. They were the best coder, the fastest debugger, the person you called when the system was down. In an AI-native team the Fleet Lead's value is in what they can judge. They decompose the spec into fleet-executable work, set the technical constraints the spec does not state, maintain the integrity of the harness, and decide when to stop directing agents and intervene directly.

> *The Fleet Lead's scarce contribution is not code. It is knowing when the fleet has produced something that looks right and is not. That judgment cannot be automated, because it rests on understanding why the system is shaped as it is.*

They own four things concretely: the decomposition of the spec into fleet-executable work, the integrity of the harness, the technical constraints the spec does not state, and the decision about when to stop directing agents and intervene directly. Each of these is a judgment call, not a mechanical step.

The Author and the Gatekeeper

The Fleet Lead replaces a cluster of roles and practices that exist in traditional delivery organizations. Naming them explicitly prevents the old roles from co-existing with the new one.

The tech lead as chief coder. In a traditional team the senior engineer's value was in what they could build. They were the best coder, the fastest debugger, the person you called when the system was down. The Fleet Lead's value is in what they can judge. A Fleet Lead who spends their day writing code has not transitioned. They have kept their old role and added agent management on top of it, which produces burnout and a review bottleneck.

The code reviewer as line reader. Traditional code review meant reading every line of every pull request. That practice assumed human production speed. At fleet production speed, line-by-line review is physically impossible and pretending it is possible is how defects get waved through. The Fleet Lead reviews at altitude (see below).

The scrum master as process facilitator. The Fleet Lead makes technical judgment calls. They do not facilitate ceremonies, track velocity, or manage a board. Daily planning handles coordination. The Flow Council handles capacity. If the Fleet Lead is spending their day in status meetings, the fleet is running unsupervised.

The QA handoff. Quality assurance as a separate phase performed by a separate team is replaced by the harness, whose integrity the Fleet Lead owns the integrity of. What disappears is the handoff, the queue, and the lag. What remains is the Fleet Lead's daily review that the harness checks correspond to the spec criteria rather than to the implementation.

Four Things That Make This Distinct

Four things separate the Fleet Lead from a senior engineer who happens to use AI tools. These are not features of the role. They are the role.

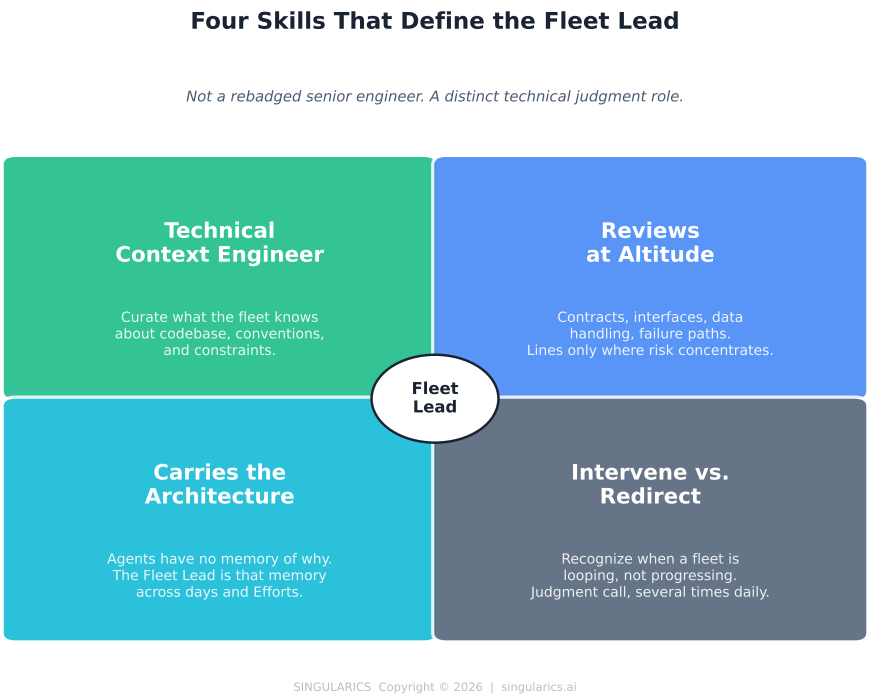

Figure 1: The four skills that define the Fleet Lead. Each addresses a gap that agents cannot close on their own, regardless of capability level.

Technical context engineer

This mirrors what the Decision Owner does for business context. The Fleet Lead curates what the fleet knows about the codebase, its conventions, its prior decisions, and its constraints. A fleet working from thin technical context produces plausible code that fits nothing around it.

This is context engineering applied to the technical domain. Which files matter for today's increment, which architectural decisions constrain the solution space, which conventions are unwritten but enforced by the team's history. The Fleet Lead assembles this context before the fleet starts, because an agent that discovers a constraint mid-build will either ignore it or halt.

Concrete example. The spec says "add a loan-level comment history." The Fleet Lead knows that the data model uses event sourcing for all audit-sensitive entities, that the comment service must integrate with the existing authorization

middleware, and that a previous team agreed to a specific API naming convention six months ago. None of this is in the spec. All of it determines whether the fleet's output fits the system. The Fleet Lead writes this into the technical context the fleet receives before the first agent starts. Without it, the fleet produces a working comment system that uses a different data pattern, a different authorization approach, and different API names. It passes its own tests and fails integration.

The parallel to the Decision Owner is precise. The Decision Owner engineers business context: what the outcome is worth, what constraints apply, what the policy says. The Fleet Lead engineers technical context: how the system is shaped, what conventions govern it, what prior decisions constrain the solution space. Both are context engineering. Both are learnable skills. Both determine whether the fleet produces something that fits or something that merely works.

Reviews at altitude

When the fleet produces several times more implementation per day than a person can read line by line, pretending that line-by-line review is still possible is how defects get waved through. The Fleet Lead reviews contracts, interfaces, data handling, failure paths, and anything touching policy. They read lines only where risk concentrates.

> *Line-by-line review of agent output is no longer possible at production volume. The Fleet Lead reviews at altitude: contracts, interfaces, failure paths, and risk concentrations. Pretending otherwise is how defects get waved through.*

This is a learned skill, not a lesser version of thorough review. It requires knowing where defects hide in this particular system, which interfaces carry the most risk, and which patterns signal that an agent has produced something structurally wrong. A reviewer who reads everything reviews nothing, because attention spread across a thousand lines catches less than attention focused on the twenty that matter.

Concrete example. The fleet produces 2,000 lines of implementation in a build day. The Fleet Lead does not read 2,000 lines. They check the API contracts against the spec. They verify the error handling returns structured error bodies as required. They confirm the data access layer uses the authorized query patterns. They look at the test agent's harness checks and verify they test the spec criteria, not the implementation. Total review time: 60 to 90 minutes on a well-harnessed system, focused on the 200 lines that carry the most risk.

Carries the architecture

Agents have no memory of why the system is shaped as it is. They reset between sessions. They carry no context from last week's decisions, last month's architectural choices, or last quarter's agreements between teams. Across days and across Efforts, the Fleet Lead is that memory. This is the part that cannot be delegated to the fleet at any capability level, because it is knowledge about intent rather than about code.

An agent can read every file in the codebase. It cannot know that a particular module was structured to accommodate a regulatory change that has not happened yet, or that two services were kept separate because they will need independent scaling next quarter, or that a naming convention exists because three teams agreed to it in a meeting six months ago. The Fleet Lead carries this context and injects it where the fleet would otherwise make locally correct decisions that are globally wrong.

Concrete example. The fleet is building a new reporting module. The agent finds that it could simplify the implementation by merging two data services into one. Locally this is correct: the merge reduces complexity, eliminates duplication, and produces cleaner code. Globally it is wrong: those services were separated because one will move to a different deployment boundary next quarter when the team splits the monolith. The Fleet Lead catches this because they were in the architecture session where the decision was made. The agent was not.

Decides when to intervene vs. redirect

Recognizing that a fleet is looping rather than progressing, and stepping in directly, is a judgment call made several times a day. Getting it wrong in one direction wastes hours of machine work. Getting it wrong in the other direction removes the leverage the model exists for.

The decision is not binary. Sometimes the right move is to redirect with better context. Sometimes it is to rewrite the prompt. Sometimes it is to write twenty lines of code directly because the agent will take an hour to converge on what the Fleet Lead can produce in minutes. The skill is reading the situation, not defaulting to one response.

Concrete example. A build agent has been working on an authentication flow for 45 minutes and has produced three attempts, each with a different approach, none matching the existing auth pattern. The Fleet Lead checks the technical context and realizes they omitted the reference to the auth middleware documentation. Two possible responses: redirect the agent with the missing context (takes five minutes, agent produces a correct implementation in 15), or write the 30 lines directly (takes ten minutes, guaranteed correct). The Fleet Lead chooses based on whether this pattern will recur in future increments. If it will, the redirect teaches the context for next time. If it will not, the direct intervention is faster.

The Three Agent Roles

Three agent roles are worth distinguishing operationally, because collapsing them into one produces a failure mode that is difficult to detect and expensive to correct.

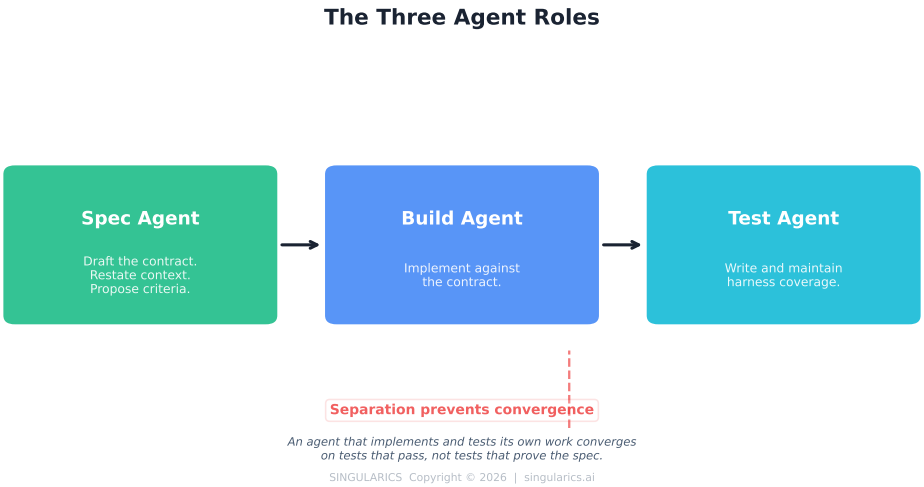

Figure 2: The three agent roles and the separation boundary. Spec agents draft the contract, build agents implement it, and test agents prove it. The critical separation is between build and test.

Spec agents

Spec agents draft the contract with the Decision Owner. They restate context, propose criteria, and surface ambiguities that need resolution before the build starts. The Decision Owner supplies the business knowledge; the spec agent structures it into something the build fleet can execute against.

This is not automation of the spec. The Decision Owner still makes every judgment call about what the increment should achieve. The spec agent handles the mechanical work of formatting, cross-referencing prior decisions, and checking that criteria are testable.

Build agents

Build agents implement against the contract. They take the signed spec, the technical context the Fleet Lead provides, and produce the increment. This is where the volume lives. A single Fleet Lead may direct several build agents working in parallel on different parts of the same increment.

The build agent's output is measured against the spec, not against the Fleet Lead's expectations. If the spec says one thing and the build agent does another, the

build agent is wrong. If the spec is ambiguous and the build agent guesses, the spec is wrong. This discipline is what keeps the contract meaningful.

Test agents

Test agents write and maintain the harness coverage. They work from the spec's acceptance criteria, not from the implementation. Their job is to prove that what the build agent produced satisfies what the Decision Owner asked for.

The agent convergence problem

> *A single agent that both implements and tests its own work will converge on tests that pass. That is not the same as tests that prove the spec. The separation between build and test is not overhead. It is the mechanism that keeps the harness honest.*

The failure mode is subtle and worth understanding precisely, because it is the most common way a green harness stops meaning anything. An agent that writes both the implementation and the tests has a natural tendency to write tests that exercise the code it produced rather than tests that verify the criteria it was given. The agent is not being deceptive. It is optimizing for coherence between the code and the tests, which is exactly what it should do if the goal is working software. The problem is that the goal is not working software. The goal is software that satisfies the spec.

Concrete example. The spec says: "When a loan application is submitted with a debt-to-income ratio above 43%, the system rejects it with a specific error code and logs the rejection reason." A single agent implementing and testing this might build a system that rejects at 45% and write tests that verify the 45% threshold. The tests pass. The harness is green. The implementation is wrong, because the spec said 43%. The agent converged on internal consistency, not on spec fidelity.

A separate test agent, working from the spec alone and never seeing the implementation code, writes a test that submits a loan with a 43.5% ratio and expects rejection. The build agent's implementation fails this test, because it used 45% instead of 43%. The defect surfaces immediately.

Keeping build and test agents separate forces the test agent to work from the spec rather than from the code. The test agent has never seen the implementation. It

knows only what the spec requires. This is the same independence that traditional quality assurance provides, applied at machine speed.

The Fleet Lead's most important review

The single most important review the Fleet Lead performs is checking whether the harness proves the spec or proves the implementation. This review is what makes the Fleet Lead role non-eliminable, because it requires understanding both the spec's intent and the harness's coverage.

The failure mode the playbook calls "harness proves the implementation" looks like this: coverage metrics are high, the harness is green on every push, and defects still reach the demo. The Decision Owner sees something wrong that the harness did not catch. The problem is not that the harness is broken. The problem is that the checks correspond to the code rather than to the criteria. The harness is proving that the system does what it does, not that it does what it should.

The Fleet Lead catches this by reading the harness checks against the spec criteria, not against the implementation. For every criterion in the spec, there should be a corresponding check in the harness. If a criterion has no check, the harness has a coverage gap. If a check has no corresponding criterion, the check is testing implementation detail rather than business intent. Both are defects in the harness, and both are the Fleet Lead's responsibility to catch.

How Many Agents Can One Lead Direct

Treat this as a real capacity question rather than an unlimited one. Organizations that treat it as unlimited will discover the answer through defects rather than through measurement.

The constraint is not agent availability. Agents are provisioned from a pool and can be scaled. The constraint is how much output one person can meaningfully review at altitude. A Fleet Lead directing more agents than they can review is not leading a fleet. They are rubber-stamping output, and the defect rate will prove it within a week.

Start conservative. Measure where review quality degrades, and let the observed number inform how the Flow Council sizes fleet capacity. The right ratio is discovered, not declared.

The number depends on three variables: the complexity of the domain, the maturity of the harness, and the Fleet Lead's experience with the codebase. In a well-harnessed system with stable conventions, one lead can typically direct 3 to 5 concurrent build agents while maintaining review quality. In a greenfield build with no harness, the sustainable number drops to 1 to 2, because every line of output depends on human judgment. These are observed ranges, not theoretical limits, and they will shift as tooling improves. The important thing is to discover your own number through measurement rather than to adopt someone else's.

How to find the ratio

Start with a number that feels conservative. Run it for two weeks. Measure two things: the defect rate at demo (defects the harness did not catch) and the Fleet Lead's subjective confidence in the output they reviewed. If both are healthy, increase the ratio. If either degrades, reduce it.

The tell that the ratio is too high: the Fleet Lead stops reviewing the harness checks against the spec criteria and starts trusting the green build. At that point they have become a rubber stamp, and the harness integrity review described above is not happening. Defects will reach the demo within days.

The tell that the ratio is too low: the Fleet Lead has idle time during the build day and is spending it writing code directly. That is not wrong in itself, but it means agent capacity is being wasted because a human could direct more agents than they currently are.

The Flow Council sizes fleet capacity at weekly calibration using this ratio. If the portfolio has six Fleet Leads and the observed sustainable ratio is one lead to N agents, total fleet capacity is approximately 6N. Exceeding that total means either hiring more Fleet Leads or accepting higher defect rates. There is no third option.

Capacity

Agent fleet capacity is provisioned to Efforts by the Flow Council in the same pool as human capacity. This is a deliberate design choice, not a limitation. Treating agent capacity as a planned resource rather than an unlimited utility produces three effects that matter.

First, it forces the Flow Council to make real trade-offs. Allocating more fleet capacity to one Effort means less is available for another. This is the same discipline applied to human capacity, and it prevents the portfolio from overcommitting on the assumption that machines are free.

Second, it makes cost visible. Agent capacity has real cost: compute, API calls, context window usage, and the Fleet Lead's review time that scales with volume. Treating it as unlimited hides these costs until they appear in a budget variance report, by which point the spending pattern is established and difficult to reverse.

Third, it keeps the ratio between Fleet Lead review capacity and agent output in balance. If agent capacity is unlimited but Fleet Lead capacity is not, the natural result is a review bottleneck that produces either rubber-stamped output or a growing queue of unreviewed work. Planning both together prevents this.

> *Agent fleet capacity is planned alongside human capacity for the same reason: both are real constraints with real cost, and pretending either is unlimited produces the same failure, which is overcommitment followed by quality collapse.*

The Flow Council sizes the fleet at weekly calibration, using the same evidence it uses for human capacity: what is in the Now horizon, what each Effort needs, and what review capacity is available from the Fleet Leads currently assigned.

The Relationship to the Decision Owner

The boundary between the Decision Owner and the Fleet Lead is clean and worth stating precisely, because blurring it produces two predictable failures.

The Decision Owner owns what is right. They hold the business context, write the spec, set the acceptance criteria, and judge whether the increment satisfies the outcome. Their authority is bounded by the envelope the Intent Council sets at the start of the outcome.

The Fleet Lead owns how it is built. They hold the technical context, decompose the spec into fleet-executable work, direct the agents, review the output, and maintain the harness. Their authority covers every technical decision that does not change what the increment delivers.

Neither decides the other's domain.

> *The owner owns what is right. The Fleet Lead owns how it is built. Neither decides the other's domain. When either crosses that line, the model produces work that is either technically sound and business-wrong, or business-right and structurally unsound.*

When the Decision Owner starts making technical choices, the implementation reflects business intuition rather than architectural judgment, and the system accumulates structural debt that the Fleet Lead did not authorize and cannot defend. When the Fleet Lead starts making business calls, the increment drifts from the outcome because technical convenience replaced business need, and the Decision Owner discovers the gap at acceptance rather than at spec time.

The two roles collaborate most visibly during spec work. The Decision Owner states what the increment must achieve. The Fleet Lead responds with what is technically feasible in one cycle, what constraints the codebase imposes, and what the decomposition will look like. This exchange takes 10 to 20 minutes when both are present. When the Fleet Lead is absent from spec work, the Decision Owner produces a spec that the Fleet Lead must rework at build time, which costs 1 to 3 hours and sometimes an entire build day. The decision rule: if the Fleet Lead misses spec time, delay the build day rather than start on a spec the Fleet Lead has not reviewed. One delayed start costs less than one wasted build day.

Together, the Decision Owner and the Fleet Lead form the human core of the team. In the smallest configuration, that is the entire human team: two people and

a fleet of agents. The Decision Owner supplies judgment about the business. The Fleet Lead supplies judgment about the system. The agents supply execution. Nothing else is required for the daily cycle to close.

The Role's Negative Space

The role is new enough to attract misidentification, and each misidentification produces a different failure.

Not a rebadged senior engineer

A senior engineer's value in a traditional team was in what they could build. The Fleet Lead's value is in what they can judge. The skills overlap, because judgment requires deep technical knowledge, but the daily work is different. A senior engineer who spends their day writing code has not transitioned to the Fleet Lead role. They have kept their old role and added agent management on top of it, which produces burnout and a review bottleneck.

Not a project manager

The Fleet Lead makes technical judgment calls. They do not coordinate handoffs, track status, or manage dependencies. The daily planning ceremony handles coordination. The Flow Council handles capacity. The Fleet Lead's time is spent on decomposition, context engineering, review, and intervention. If they are spending their day in status meetings, they are not leading the fleet, and the fleet is running unsupervised.

Not a coordinator

Coordination is necessary and it is not this role. A coordinator routes information between people. The Fleet Lead routes technical judgment to agents, reviews what comes back, and decides what to do about it. The difference is that a coordinator can be replaced by a better process. The Fleet Lead cannot, because their contribution is judgment rather than information flow.

Not eliminable

This is the central claim, and it is the one most likely to draw skepticism. At every increase in agent capability, someone will ask whether the Fleet Lead can be removed. The answer is no, for a reason that does not change with capability: agents do not carry the intent behind the architecture. They can read every file in the codebase and still not know why it is shaped as it is. That knowledge lives

in the Fleet Lead, and it is what prevents locally correct decisions from being globally wrong.

The strongest counterargument is that context windows are growing and agents are becoming better at reasoning across large codebases, so the architectural memory gap will close. Grant the premise for the sake of argument. Even an agent that can read and reason about every file still cannot know that the CTO agreed in a meeting last month to keep two services separate because of a planned org split next quarter. That knowledge is organizational intent, not code. It changes through conversations, not through commits. No repository contains it. No context window captures it. The Fleet Lead is the carrier, and the need for that carrier persists at every capability level we can currently project.

> *The Fleet Lead is a technical judgment role that cannot be eliminated at any agent capability level. Agents can read every file in the codebase and still not know why it is shaped as it is. That knowledge lives in the Fleet Lead.*

The role will evolve. The ratio of agents to leads will change. The tools will improve. The review techniques will mature. But the need for a human who carries architectural intent across time and across Efforts will persist for as long as systems are built to serve purposes that change, because purpose is not in the code.

Failure Modes

Harness proves the implementation. Coverage is high, defects still reach the demo. The tell: the Decision Owner sees problems the harness did not catch, repeatedly. The fix: the Fleet Lead reviews harness checks against criteria, not against code. Every criterion in the spec must have a corresponding check. Every check without a corresponding criterion is testing implementation detail.

Fleet Lead becomes chief coder. The Fleet Lead spends the day writing code instead of directing the fleet. The tell: agent output volume drops, the Fleet Lead is the bottleneck, and the team produces less than it did with the fleet running unsupervised. The fix: the Fleet Lead's value is in judgment, not in code. If they are writing code, they are not reviewing, not maintaining the harness, and not carrying the architecture. Redirect.

Fleet Lead rubber-stamps. The ratio is too high. The Fleet Lead approves output without meaningful review. The tell: defects reach the demo that a review at altitude would have caught, specifically contract mismatches, interface violations, and policy-touching code that was not flagged. The fix: reduce the ratio and re-establish the altitude review discipline.

Build and test agents are collapsed. To save time or simplify the workflow, someone collapses build and test into a single agent. The tell: the harness is always green, but the demo reveals functional gaps. The agent converged on tests that pass rather than tests that prove. The fix: re-separate. The test agent must work from the spec, not from the code. This separation is not optional.

Technical context is thin. The Fleet Lead does not invest in assembling the technical context the fleet needs before the build starts. The tell: the fleet produces output that works in isolation but does not integrate with the existing system. Interfaces do not match conventions. Data patterns diverge from the established model. The fix: the Fleet Lead spends the first hour of the build day on context assembly, the same way the Decision Owner spends 45 to 90 minutes on spec work. The two activities are parallel and equally important.

Fleet Lead not present at spec time. The Decision Owner writes the spec without the Fleet Lead's input on technical feasibility. The tell: the Fleet Lead discovers at build time that the spec is technically infeasible in one cycle, and the day is spent reworking the spec rather than building. The fix: spec work is a collaboration between the Decision Owner and the Fleet Lead. The Decision Owner states what. The Fleet Lead responds with what is feasible. This exchange takes minutes when both are present and days when one is missing.

The Coordination Arithmetic

Teams as delivery has known them were sized to production, and coordination cost climbed with every added member, the links between people multiplying far faster than the hands. A ten-person team is forty-five conversations. The crew around a fleet is three or four people, and machine parallelism adds no coordination channels between humans, because agents pass artifacts rather than interpretations. Most of their coordination is not pairwise at all. The fleet converges on the spec and the harness, a hub rather than a mesh. It scales linearly, it never tires, and it keeps no calendar.

A sharper law replaces the old one. A lossless network propagates everything losslessly, including mistakes. One wrong assumption injected at the hub reaches the whole formation with perfect fidelity. The communication bottleneck is gone, and the judgment bottleneck it was hiding is now the whole game. It sits where the model puts its people, at the spec, at the context pack, at the harness.

Where the Theory Meets the Text

If the fleet writes all the code, someone still has to be able to read it. An organization whose systems no human can audit becomes a passenger in them, polite, hopeful, and unable to check. Aviation settled this long ago. The autopilot flies most of every flight, and a certified pilot sits in the seat because the person responsible must be able to take the controls. The Fleet Lead is that requirement, fluent upward in intent and downward in code, able to descend into the text on the day something breaks. Production no longer needs human hands. Accountability still needs human reading, and no enterprise gets to tell a regulator, a customer, or a court that the machine was the one who understood.

Machines store, and storage is not holding. A stored decision cannot notice the moment it becomes relevant. A held one can. That is why carrying the architecture cannot be delegated to the fleet at any capability level.

The Way In

The entry-level path is not gone. It moved. The tasks juniors used to learn on were absorbed by the fleets first, and what replaces the old ticket ladder is the seat beside a senior Fleet Lead. New engineers arrive code-fluent, with orchestration instincts their elders learned late, and they spend their first years reading at altitude next to someone who carries a theory of the system, absorbing it the only way theory transfers, by working under one. The rungs change. The ladder does not go away.

The model makes the seat structural. Any crew may carry a learning seat, a named, funded position for a code-fluent junior. A crew that has run a year with one should be able to say what its junior can now accept, review, and drive alone. If nothing has moved, the seat was a chair, not an apprenticeship.

Building a Fleet Lead practice?

30 minutes to discuss the role, the ratio, and what changes in your context. No pitch.

Book a call | singularics.ai

SINGULARICS

We help large organizations close the intent gap between strategy and execution so that when they accelerate with AI, they accelerate in the right direction.

singularics.ai | Book a conversation