A field guide for developers

Ticket taker, outcome owner

A developer's system for building trust, influence, and career momentum

Own the next state, not every task.

Mike Salari

Staff engineer, architect, author

Preface

For more than fifteen years I have watched capable developers stall for reasons that had nothing to do with code. They finished what was assigned and were told to be more proactive, which is feedback that is correct in direction and useless in detail. This book translates that judgment into behaviors you can perform during an ordinary workday: clarify the outcome, expose the assumption, offer a bounded move, communicate the changed state, close the loop.

It is not a personality conversion program. You do not need to become louder. You need a system that makes your judgment easier to trust.

Contents

About the author

Mike Salari is a staff engineer, architect, and author with more than fifteen years of experience building production software.

His work spans consumer products, fintech, enterprise systems, architecture, technical leadership, and regulated environments, with experience across Apple, Deloitte, Cisco, Mastercard, and Visa.

His writing focuses on the gap between technical ability and trusted professional judgment: how engineers explain decisions, handle ambiguity, communicate risk, design reliable systems, and grow into broader scope without turning work into performance theatre.

He is the author of engineering interview, architecture, and career-development books including *The iOS Interview Blueprint*, *Share Your Screen*, *Altitude*, and *The Engineer Visibility Blueprint*.

> ***Written from work I have actually done.***

More books: mikesalari.dev

Copyright and edition note

Ticket taker, outcome owner *A developer's system for building trust, influence, and career momentum*

1st Edition, July 2026. Published independently.

This book is provided for educational purposes. It does not constitute legal, medical, mental-health, employment, or financial advice. Workplace conditions, authority, policies, and safety requirements vary. Readers remain responsible for professional judgment and for following the rules of their employer and jurisdiction.

The stories of Daniel and Maya are composite teaching devices. They do not describe specific individuals or employers. Examples have been simplified to teach the underlying behavior.

Product names, company names, and trademarks belong to their respective owners. Their appearance does not imply endorsement.

Permissions, bulk orders, and professional inquiries: salari.dev/contact

Opening: The review nobody explains

Daniel entered his performance review expecting a promotion discussion. He left with two sentences:

> *Be more proactive. Take more ownership.*

He had finished every ticket. His code was careful. He was dependable. Yet the promotion went nowhere because the feedback named a judgment without naming a behavior.

I have sat on the hiring side of that verdict. Here is what the review rarely admits: by the time two people are discussing your promotion in a room, the decision they are weighing has mostly already formed, in dozens of small moments where you either made the work easier to trust or you did not. The review does not create the verdict so much as ratify one the daily behavior already wrote. Daniel's real problem was not the two sentences he was handed. It was that no one, himself included, could point to the moments where the verdict was being decided.

At the desk beside him, Maya was no louder and no more willing to live at work. The difference was visible in ordinary moments. Before coding, she clarified the outcome and the assumptions most likely to break it. When risk changed, she said so while the team still had choices. After merge, she verified rollout and closed the temporary work.

Daniel completed assigned tasks. Maya made the next state easier to understand, coordinate, and trust.

That is the transformation in this book.

> ***Own the next state, not every task.***

Ownership is not absorbing unlimited work. It is making the next responsible state visible and helping the work reach it.

How to use this book

Read it once. Then return to the chapter that matches the state you are in: unclear, stalled, invisible, unfinished, or overloaded. Every chapter ends with a minimum move you can perform during normal work.

You do not need to become louder. You need to become easier to trust.

Where this book fits

This book is about the daily behaviors that make a developer easier to trust at any level: what you do before a task is clear, while work is moving, when risk changes, and after code is merged.

Altitude is for senior engineers moving toward Staff scope. It focuses on broader technical judgment, cross-team influence, and the evidence promotion committees expect. Read this book for the operating habits. Read *Altitude* for the Staff-level scope those habits may eventually support.

None of this asks you to become extroverted, endlessly available, or the person who absorbs every unowned problem. It asks you to make better moves earlier, say them clearly, and finish them.

The proactive developer diagnostic

This is not an opinion survey. Do not rate how proactive you feel. Count what you actually did.

For each behavior below, count the separate times it happened in the last three weeks. Write the real number, then score it: zero times is 0, once is 1, twice is 2, three or more is 3. Fifteen behaviors, forty-five points on the table.

If you cannot name a specific instance, it scores 0. Intending to do it, or being the kind of person who would, is not a count. That is the whole point of counting. It cannot be flattered.

In the last three weeks, how many times did I...	Count
write down the outcome of a task in plain language, before naming any technology?	0-3
name an assumption that could invalidate my work, before building on it?	0-3
check whether a decision was mine to make, mine to inform, or one I needed approved?	0-3
flag a risk that had changed while the team still had time to act on it?	0-3
ask for help with what I expected, what I observed, and what I had already tried?	0-3
send an update that reported a changed state instead of logging my activity?	0-3
bring a recommendation or a bounded next step instead of only a problem?	0-3
open a pull request that stated its outcome, risk, verification, and scope?	0-3
close a handoff with a named owner and a date?	0-3
verify rollout, adoption, or operational readiness after a merge?	0-3
remove a source of repeated friction instead of absorbing it again?	0-3
say no to work by putting the trade-off on the table?	0-3
protect a block of focused work without going dark on the team?	0-3
record evidence of an outcome, a prevented risk, or a system I improved?	0-3
stop or hand off an initiative when the evidence, priority, or permission changed?	0-3

Add the scores. The band you land in is not a personality type. It is a description of what you currently cost the people around you.

Your result

- **0-9. Someone is quietly covering for you.** The people nearby absorb the clarifying, chasing, and closing your work needs. You are cheap to assign and expensive to manage, and that cost is obvious to them and invisible to you.
- **10-19. Your manager is your project manager.** Work moves when you are pushed. Someone else is holding your risks, your dates, and the memory of what you were doing.
- **20-29. You lose the last mile.** You open well and fade near the close, so people re-check whether your work actually shipped. Sometimes it did not.

- **30-38. Trusted in your lane, chased outside it.** Inside your area you rarely need follow-up. The moment the work crosses a boundary, you go quiet and someone comes looking.
- **39-45. Low-maintenance, which is rare.** People stop asking where your work stands. Notice how high the number had to climb to get here, and how little of it you could fake for three weeks.

The score is not a verdict on you. It is a list of the behaviors you are not yet doing. Take the lowest row and start there. Retake this after day 21 and count again, because the only honest measure of change is a second count.

The ownership signal

People do not experience your intentions. They experience the signal created by your behavior.

A useful ownership signal has three parts:

> ***Judgment x Clarity x Follow-through***

- **Judgment** means you notice the relevant outcome, constraint, risk, and trade-off.
- **Clarity** means other people can understand the state of the work and what is needed from them.
- **Follow-through** means the work reaches a real conclusion rather than becoming another abandoned thread.

The multiplication matters. If one factor is close to zero, the total signal collapses.

If judgment is strong but clarity is weak, people may experience you as smart but mysterious. If clarity is strong but judgment is weak, your updates may sound polished without improving a decision. Judgment and clarity without follow-through look insightful but unreliable. Follow-through without judgment looks like dependable execution that rarely changes the outcome.

Trust grows when all three travel together: you notice what matters, make the state understandable, and carry the work to a responsible end.

This book strengthens all three. SCOPE is the operating loop. The chapters that follow show what the loop looks like in the actual places where developers are judged: tickets, code, meetings, written artifacts, incidents, one-to-ones, and handoffs.

1. From ticket taker to outcome owner

On Monday morning, Daniel opened a new ticket:

> *Add biometric authentication to the account flow.*

The acceptance criteria described the happy path. Nothing mentioned devices without biometric enrollment, failed attempts, account recovery, analytics, or whether the existing PIN flow stayed supported.

Daniel assumed product would add the details if they mattered. He began coding.

Maya opened the same ticket and wrote five lines:

- Outcome: reduce login friction without weakening account recovery.
- Open question: is biometric login optional or the new default?
- Compatibility: what is the minimum supported OS and device policy?
- Risk: lockout behavior after repeated failures.
- Decision needed: who owns recovery and security approval?

She sent two questions, then built a small state diagram while she waited.

By Friday, Daniel's implementation looked polished and failed product review, because the recovery path was missing. Maya's was smaller, matched the intended policy, and was ready for QA.

The difference did not begin in code. It began in the first hour, in what each of them did with an ambiguous ticket.

What "be proactive" actually means

"Be proactive" and "take more ownership" are not really feedback. They are the compressed version of something the reviewer could not, or would not, say in full. I have sat in the rooms where that compression happens, where a handful of people decide who is ready for more and who is not, and the phrase almost always stands in for one of six specific things. None of them is about personality. Every one is a behavior you can perform on a Tuesday.

"Take more ownership" usually means the last time something you built broke after merge, someone else noticed it, chased it, and closed it. Ownership is tracking the result past your own code change, so the person who would otherwise chase it does not have to.

"Speak up earlier" usually means you had the concern in week one and raised it in week three, after the choice that would have been cheap to change had already set. It is not a request to talk more. It is a request to put the assumption, the risk, or the disagreement on the table while the team still has options.

"Be more independent" usually means you asked a question you could have answered yourself, or you waited for an answer instead of forming a view. Independence is investigating first and then asking a narrow question that moves a decision, not guessing and not stalling.

"Think more strategically" usually means you optimised the thing in front of you without checking whether it was the thing that mattered. It is connecting a technical change to a customer, a cost, a reliability number, or another team's work, out loud, before you start.

"Increase your visibility" usually means your manager could not reconstruct what you did last quarter without asking you. It is not self-promotion. It is leaving a trail, decisions and results in a form someone else can read after the fact.

"Operate at the next level" usually means they want evidence you can already do the job before they hand you the title. It is broader scope, more ambiguity, and more coordination, handled with judgment, inside the role you hold now.

Notice what those six have in common. Each is a sentence someone said about you in a meeting you were not in, and each one names a behavior, not a trait. So do not let the vague version stand. When you hear it, convert it in the moment:

> *When you say "more proactive," what is one recent situation where you expected a different action from me?*
>
> *What would the stronger behavior have looked like, and when should it have happened?*

You are dragging a personality judgment back into the world of behavior, where it can actually be practiced.

Reactive does not mean bad

Reactive work is necessary. Incidents need response, tickets need implementation, customer questions need answers. The problem is not reacting. It is when every piece of your work is reactive, because someone who only ever responds has to be aimed by somebody else, and being aimed by somebody else is the definition of the level you are trying to leave.

A ticket taker waits for a complete task, reports activity, stays blocked until asked, fixes the same symptom twice, and treats merge as the finish line. An outcome owner clarifies the result, reports the changed state, escalates with evidence, asks why the symptom keeps coming back, checks that the change was adopted, names the trade-off when priorities collide, and leaves a trail someone can follow.

Reliability is not the thing you graduate from. Outcome ownership is built on top of it. A brilliant clarifier who does not deliver is just a more interesting kind of unreliable.

The responsibility ladder

What counts as proactive changes with your level.

Level	Healthy proactive behavior
Junior	Clarifies tasks, shows work early, asks for help with evidence, communicates blockers, and closes assigned work.
Mid-level	Anticipates edge cases, coordinates dependencies, improves repeated friction, and owns delivery within a feature area.
Senior	Frames ambiguous problems, manages risk, influences decisions, improves systems, and owns outcomes across functions.
Staff or Lead	Identifies systemic constraints, creates mechanisms others adopt, aligns teams, and manages long-term trade-offs.

The trap at every rung is acting one level away from the authority you actually hold. A junior who secretly redesigns the architecture is not acting senior; they are acting unaccountable. A Staff engineer who waits for perfectly written tickets is not being careful; they are being a very expensive junior. The behavior has to match the decision boundary you hold, not the one you wish you held.

Proactivity is not only about you

Initiative is never purely a personal trait, because nobody takes it in a vacuum. People raise risks, ask questions, and start things when the last person who did so was thanked rather than punished, and when acting on a decision does not require three approvals. Where every small choice needs sign-off, where questions are read as incompetence, and where initiative reliably converts into more unpaid work, even strong individual behavior has a low ceiling.

This book teaches the part you control. It is also honest about the part you do not, and later it will help you tell the two apart, because burning yourself trying to out-behave a broken system is not ownership. It is a slower way to leave.

What ownership looks like on an ordinary Tuesday

None of this requires a flagship project. It shows up in small decisions. When a requirement is unclear, you investigate and ask about the one assumption that could invalidate the work. When an estimate slips, you update your confidence and offer a trade-off instead of hoping. When another team is late, you find the decision owner and a parallel path. When review is slow, you make the change cheaper to review and name what the delay costs the release. When a feature ships, you check rollout, monitoring, adoption, and cleanup. When a new request lands, you name the current priority and ask which outcome should move.

The through-line is simple: an outcome owner does not make the rest of the team chase reality.

The promotion trap

Most developers believe promotion runs in one order: you get the title, then the authority, then you start behaving at the new level. It is a reasonable belief, and it is backwards. Almost every organization I have seen runs it the other way. You demonstrate the next level's behavior inside safe boundaries first. You repeat it until it is a pattern and not an anecdote. You make that pattern visible to the people who decide. And only then, when a role and a headcount and a process line up, does the title arrive to confirm what everyone can already see.

That reversal is why so many capable people stall. They wait at step one for a permission that only arrives at step four, and they read the delay as unfairness when it is really a request for evidence they have not yet supplied.

None of which is a licence to quietly do a bigger job at your current pay forever. The move is to make the trade explicit:

> *I want to build evidence for the next level without creating an open-ended scope increase. Which outcome can I own over the next six weeks, what behavior will you evaluate, and when will we review the result?*

If your manager will not define the standard, the opportunity, or the review date, that refusal is itself the answer. It tells you the next level is not currently available to you here, whatever anyone says in the meeting.

Action plan: define your next-level behavior

1. Write the level you currently hold.
2. Write the level you want next.
3. Choose three behaviors from the responsibility ladder.
4. Ask your manager which one would create the most value in the next month.
5. Save one recent situation where you could have acted earlier.

Own the next state, not every task. For Daniel, the next state was the recovery path nobody had written down. Owning it meant naming that gap out loud before he wrote a line of code, not finishing the ticket faster.

Continue the full book

You just read the opening, the diagnostic, and the first chapter. The full edition continues with the five-move SCOPE loop, starting before motivation arrives, speaking and writing so work moves, blocker and closure protocols, boundaries, a 21-day plan, and the field cards.

Get the full book: mikesalari.dev/l/ticket-taker

You already have the diagnostic. Count today, and let the day 21 recount tell you what actually changed: day21.salari.dev

> ***Own the next state, not every task.***