KIT · MAINTAINED REFERENCE · SINGLE-DOCUMENT EDITION

The New Engineering Leader's Kit for the AI-Native Era

Ed Schaefer · Version 1.0 · September 2026 · edward-schaefer.com/kit/first-90-days · Free to share

navigating your first 90 days

This is the single-document edition: the router, the question bank, and the cadence calendar in one printable flow. Each piece also stands on its own: edward-schaefer.com/kit/first-90-days, /kit/first-90-days/questions, /kit/first-90-days/cadence.

Built for a VP or Director stepping into a new engineering org, whether you're new to the role or newly handed a team you didn't build. It holds up just as well if you're not new at all. A sitting leader taking their existing org into a new way of working runs into most of the same open questions a first-time VP does. Earlier-stage leads will find most of this useful too.

Michael Watkins named this transition years before AI entered the picture, and *The First 90 Days* is still the book most people reach for first. This kit borrows the same instinct (diagnose before you act) for a narrower moment: an engineering org, in an era where verification debt and comprehension debt are failure modes his book never had to cover.

What's different at day 90

This kit doesn't promise fluency across every practice area in the field guide. It promises five things, each pointing at something that already exists today, not a stub waiting on future writing:

1. **A running operating cadence.** 1:1s, feedback, review, and retro are already sequenced into one calendar, not still being invented.
2. **Verification defined before you need it.** A Definition of Verified in hand, not written after the first bad merge.
3. **An incident plan that predates the incident.** The Agent Incident Runbook filled in before something breaks, not after.
4. **A real read on your team's AI maturity.** Usage and governance, measured separately, not guessed at from one impression.
5. **Comprehension debt actively managed.** Checked against named audit signals, not discovered the hard way.

Part 1 — The Router: What to Do, in Order

Three blocks, roughly a month each, each building on the last: get the baseline running, then define what's still undefined, then close the loop before you need it. Most new leaders inherit rollouts already in motion, not a blank slate. The flow below is built around auditing what's already running as often as it's built around starting something new.

Days 1–30 — get the baseline running

- **Two diagnostics, run separately.** The AI-Native Engineering Maturity Model (org-level, free download) and the Leader's Self-Assessment (team-level, sixteen questions, about five minutes). Run both. A team's AI usage and its AI governance move at different speeds, and asking only "are we using AI" answers the wrong question.
- **The 1:1 Playbook.** Weekly, every direct report, thirty minutes, starting week one. The single most load-bearing move in this kit: establishing 1:1 rhythm with an inherited team is almost always a new leader's first real action.
- **The feedback playbooks: Giving and Receiving.** Not scheduled. Continuous from day one, both directions: giving it well, and getting something usable out of what arrives unplanned. When you're ready to go ask instead of waiting, Soliciting is the third of the set.
- **The AI Operating Model for Teams.** Fill in the "as of" date and hand it to your team as the week-1-to-30 document: how we work, the tier scale, what breaks a tie.
- **The Agent Rollout Pre-Flight Checklist, read as an audit, not a future-rollout form.** Walk it against the agent work already running, not something you're about to launch. For each rollout you inherited, ask which of these boxes the team would have been able to check when they set it up. It's the fastest concrete read on governance maturity available in the first month, and it turns a checklist meant for what's next into a read on what's already here.

Days 30–60 — define what's still undefined

- **The Definition of Verified.** Stood up before the first disputed merge, not written the week after one ships that shouldn't have.
- **Verification debt and comprehension debt, audited together.** Detail below.
- **The Review Playbook.** A fixed cadence, established rather than left to happen ad hoc. Every two weeks is the default absent a reason to run differently.
- **The Retrospective Playbook.** Slotted between the review and the next planning session, team-only.

Days 60–90 — close the loop before you need it

- **The Agent Incident Runbook.** Filled in before the first agent-caused incident, not improvised in the moment.
- **Inheriting an agent fleet.** Principle-level only, on purpose. Detail below.
- **The cadence, now running as one system.** 1:1s weekly, review on its fixed cycle, retro between review and planning, feedback continuous underneath all three. Part 3 below has the full compiled sequence.

Verification debt

Verification debt is what builds up when code gets produced faster than any human review process was built to check it. Chris Stokel-Walker's reporting for LeadDev put a number on the gap: 96% of developers don't fully trust that AI-generated code is functionally correct, yet only 48% consistently verify it before committing, with AI-authored code already at 42% of all committed code as of that reporting, on a trajectory toward 65% by 2027. A separate 1,100-plus-developer survey from Sonar found the same shape from a different angle: 53% report AI generating code that "looks correct but isn't reliable," and 88% report at least one negative technical-debt impact from AI use, even though most of the same respondents also report real benefits.

Two questions worth asking your team directly in this window: does AI-authored code get reviewed differently than human-authored code, or does everything get the same pass regardless of origin. And if five engineers on this team each defined "verified" for a merge, would you get five different answers, or one. The second question is exactly what the Definition of Verified template exists to close.

Comprehension debt

Comprehension debt is the gap between how much code exists in a system and how much of it anyone can still account for. Comprehension Debt Is a Debt covers the full argument. Kevin Cushnie's treatment for Forbes Technology Council gives a new leader something more useful than the definition alone: four concrete signals to check in the first weeks.

1. **Review-time-to-complexity ratio.** If review time per pull request has narrowed by more than 30% while complexity hasn't, that's compression, not efficiency.
2. **Architectural decision-record accuracy.** Pick a handful of recent architectural decisions. Below 60% match between what's on record and what shipped is a warning; below 40%, intent isn't being captured anywhere anymore.
3. **Post-mortem pattern-matching.** More than two recent incidents citing unclear intent as a contributing factor is a real signal, not noise.
4. **The explanation hedge.** Listen for your most senior engineers hedging more than they used to when asked to explain how something works in a design review.

Cushnie's own line is the sharpest summary available: the codebase nobody understands is also the codebase nobody can defend.

Inheriting an agent fleet

This section stays short on purpose. Verification debt and comprehension debt above are both well-sourced, with hard numbers and named signals behind them. Inheriting someone else's agent fleet, with its tools and spend nobody fully mapped, is the weakest-sourced piece of this kit, and there's no real client moment here yet to anchor it with. Nothing below is invented to fill that gap.

What does exist: AIMultiple's agent-sprawl checklist recommends one concrete first move for anyone inheriting an estate they didn't build. Run a systematic inventory of every agent already running — its purpose and what it can reach — because retrofitting governance onto a fleet that's already operating is harder than building it in from the start. On spend specifically, Redress Compliance's 2026 audit of enterprise software spend found off-books AI spend running 4 to 9% of total software budget at the audited organizations, a 6% median, typically two to three times what finance had actually budgeted for it. Treat this as a directional finding from a small sample, not a number to expect exactly. Separate practitioner survey data puts common AI tooling budgets around 1 to 3% of total engineering spend, often near $1,000 per developer per year, with most surveyed leaders (86% in that survey) uncertain which of their own tools are actually earning their spend.

The honest version of this section, for now: inventory before you add anything new, and expect the spend picture to be less mapped than the org chart suggests. A deeper, story-grounded version of this section is on the roadmap once real client moments surface to build it from.

Also relevant

Outside the five outcomes above, several existing essays speak directly to a new VP or Director, without being part of the ruled flow: Before You Schedule a Skip-Level, Ask This First, Emotional Acuity: The Leadership Capability Most VPs Never Learned, Surround Yourself with Courageous Truth-Tellers, Not Sycophantic "Yes Men", The Steadiness Test, and Tone in Digital Communication. The full argument this kit draws its vocabulary from lives in the AI-Native Ways of Working field guide.

Part 2 — The New Engineering Leader's Question Bank: what to ask, across your first 90 days

Every question below traces to a named source: a practitioner who's run this transition before, or one of the AI-era research lanes' verified findings. None of it is generic. Where a source gives a script or an exact phrasing, it's quoted; where it gives a pattern, this bank applies the pattern to the AI-native specifics the source itself didn't cover.

Grouped by when to ask, not by topic, because the sources that specify a shape (Hogan, Button, Larson, Uplevel) converge on the same one: listen before you narrow, and narrow before you act. Don't skip ahead to Month 2's questions in week one. The order is part of what makes the questions work.

Week 1 — before you ask anyone else anything

These go to your own manager, or to yourself, before the listening tour starts.

- **"What are you optimizing for?"** Ask your own manager this, not "why prioritize this." Button's framing: the "optimizing for" version surfaces the real constraint, revenue targets, shipping deadlines, a tech-debt tradeoff someone already made, where the "why" version just gets you a directive back. (Button, "The first 30 days as a director of engineering")
- **"What would make the first 90 days a clear win, in your words?"** Ask it explicitly rather than assuming you already know. Practitioner consensus (Larson, Uplevel) treats month one as diagnosis, not delivery. If your own manager's version of a win is delivery, that gap is worth surfacing now, not in month three. (Larson, "Your first 90 days as CTO or VP Engineering"; Uplevel, "Your First 90 Days as a New Engineering Leader")
- **"Is this team AI-assisted, or something closer to AI-native, and how would you tell the difference?"** Ask it of your own manager or whoever's read on the org is broadest. It sets expectations for the dual-axis assessment below, and it's a fair question to ask before you've formed your own view.

Weeks 1–2 — the listening tour

Run these as 1:1 conversations, roughly thirty minutes, with every direct report, and with cross-functional peers as you meet them. Lara Hogan's own structure for this window: listen only, don't act, and keep a brief prep note in the calendar invite so nobody's caught flat-footed.

With each direct report or peer: - "When you think about this team as a whole and how it works, what change do you want to see?" - "What's working so well that we shouldn't change?" - Follow-ups, only if the first two open something worth chasing: "What's the risk if we don't make that change?" "If we made that change, how would it land on you and your team specifically?" "Who else should I be talking to about this?"

(Hogan, "How to spend your first 30 days in a new senior-level role")

With your own skip-level, used carefully. If you've inherited managers, their reports are a skip-level relationship, not a direct one, and the discipline matters before the questions do: tell each manager first, framed as support ("I want a broader view so I can support you better"), not surveillance. Share back themes afterward, never individual comments. Solve what surfaces through the manager, not around them. If you're a sitting leader rather than newly arrived, these are skip-levels you already run — the discipline still holds, just reframe the ask around the shift to AI-native ways of working rather than around getting acquainted.

- "What's one thing you wish leadership understood better about the work your team is doing?"
- "Where do you spend most of your time day to day?"
- "If you had the power to change one thing about how this organization works, what would it be?"

(makemeacto.cc, "Skip Level Meetings Are About Insight, Not Distrust"; the discipline rules also draw on jayvilalta.com, "Managing Managers")

Weeks 2–4 — the AI-native diagnostic, two axes

Every source that measures both finds orgs mismatched on usage and governance, not uniformly high or low on either. Ask about them separately. Run the Maturity Model and Self-Assessment alongside these conversations, not instead of them.

Usage axis: - "Walk me through where AI actually touches your build-to-ship path today. Which phase is furthest along, and which one hasn't been touched at all?" Coder's own recommendation: find the *lowest*-maturity phase, not the average. Their assessment of 100 organizations found nearly 80% still sitting at the first two of five maturity stages, and 70% of agents running in infrastructure that was never built to support them. (Coder, "What 100 Engineering Teams Revealed About AI Maturity") - "If we could only prove one thing got better before we touched anything else, what would that one metric be?" (Coder, same source)

Governance axis: - "Where would you put us, roughly, on tooling maturity versus on governance maturity, and are those two numbers even close to each other?" Microsoft's own maturity model is explicit that the two can sit at very different levels inside the same org, a team at Level 400 on tooling and Level 100 on governance simultaneously. (Microsoft Learn, "Introduction to the Agentic AI adoption maturity model") - "Do we have a registry of every agent currently running: its purpose, its identity, and what it can reach? If not, who would even know how to start building that list?" (AIMultiple, "AI Agent Sprawl: Signs & Checklist to Manage Sprawl")

Spend, asked directly rather than inferred: - "What's our AI tooling spend today, and what share of the engineering budget does that represent?" A common benchmark is roughly 1 to 3% of engineering budget, often near $1,000 per developer per year, with a wide real range from $500 to $3,000-plus. (Laura Tacho, getdx.com, "How are engineering leaders approaching 2026 AI tooling budgets?") - "Which of these tools are we actually confident are earning that spend, and which has nobody

evaluated?" In the same survey, 86% of leaders weren't sure which of their own tools delivered the most benefit. Expect an honest answer here to be partial. (Tacho, same source)

Month 2 — the verification and comprehension-debt audit

These are closer to a structured audit than an open conversation. Run them against real, recent work, not hypotheticals.

Verification debt: - "Walk me through the last AI-authored change that shipped. Who checked it, and against what?" Sonar's 1,100-plus-developer survey found 96% of developers don't fully trust AI-generated code is functionally correct, yet only 48% consistently verify it before committing, and 53% report AI code that "looks correct but isn't reliable." (Sonar, "Sonar Data Reveals Critical 'Verification Gap' in AI Coding"; LeadDev, "You can't verify all the AI-generated code") - "Does AI-authored code get reviewed differently than human-authored code here, or does everything get the same pass regardless of where it came from?" - "If I asked five engineers on this team to define 'verified' for a merge, would I get five different answers?" This is exactly the gap the Definition of Verified template is built to close.

Comprehension debt, using Kevin Cushnie's four audit signals directly, each turned into a question you can ask this month: - "Has review time per pull request gone down noticeably even though the work itself hasn't gotten simpler?" A drop of more than 30% is a compression signal, not an efficiency win. - "Pick a handful of recent architectural decisions. Can anyone produce a record that actually matches what shipped?" Below 60% accurate is a warning; below 40%, nobody's capturing intent anywhere anymore. - "In our last handful of incidents, how many post-mortems named unclear intent as a contributing factor?" More than two is a real signal, not noise. - "In design reviews, are our most senior engineers hedging more than they used to when asked to explain how something works?" (Kevin Cushnie, Forbes Technology Council, "Comprehension Debt: How AI Is Re-Creating The Legacy Code Problem In Months")

Month 3 — close the loop

Self-facing, mostly, and worth writing the answers down rather than just thinking them.

- "What did I say I'd figure out by day 90, and did I?"
- "What's the one thing that was true in week one that I still haven't acted on?" Practitioner consensus warns against both extremes here: moving before you understand the shape of the problem, and never establishing traction at all. Neither is the goal. (Larson; Hogan)
- Team-facing, once the cadence has been running a full cycle: "Now that review, retro, and 1:1s are actually happening on a rhythm, what would you change about how we run them?" This feeds directly into the next retrospective, not a separate conversation.

Part 3 — The Cadence Calendar: your first 90 days as a rhythm

Two different things live in this part. The first table is the standing cadence: the rhythm your team runs on once it's set up, compiled from the four playbooks that already rule it. Nothing here is invented; each line is cited to the playbook that decided it. What follows the table is when, inside your first 90 days, each piece of this kit first gets used. That's a sequencing choice, not a new standing cadence: the kit doesn't add rituals the playbooks don't already have.

The ruled cadences, compiled

Ritual	Cadence	Who's in the room	Ruled in
1:1s	Weekly, every direct report, thirty minutes	Manager and report	The 1:1 Playbook
Feedback (giving)	Continuous. Said within days of the moment, not saved for a scheduled event	Whoever's involved	The Giving Feedback Playbook
Feedback (receiving and follow-through checkpoints)	Continuous as it arrives; recurring checkpoints get tied to a rhythm you already have, like the standing 1:1, rather than a separate meeting	Whoever's involved	The Receiving Feedback Playbook
Review	Fixed cadence: weekly, every other week, or monthly, depending on the team and how fast things change. Every two weeks is the default absent a reason to run differently	Team, plus stakeholders and sponsors for the demo	The Review Playbook
Retrospective	Between the review and the next planning session, after one and before the other	Team only	The Retrospective Playbook
AI Operating Model revisit	Quarterly, and after any agent incident	Team	The AI Operating Model for Teams

A note on the feedback rows: the three feedback playbooks (Giving, Receiving, and Soliciting) publish as separate pages. Until all three are live, `/playbooks/feedback` is the interim home. Both cadences above hold regardless of which page they live on.

When each kit asset first deploys

Days 1–7. 1:1s go on the calendar this week and start the weekly rhythm immediately. Feedback is live from day one; nothing to schedule. The Maturity Model and Self-Assessment run once, this week, as a starting read, not a recurring ritual.

Days 7–30. The AI Operating Model gets filled in and handed to the team as the week-1-to-30 reference document. The Pre-Flight Checklist gets walked once, this window, against the rollouts already running, the audit use rather than a pre-launch one.

First natural cycle (typically inside days 14–30, per the review default above). The first Review runs on whatever cadence you and the team settle on. The first Retrospective follows it, before the next planning session. From here forward, Review and Retro repeat on their standing cadence from the table above, not a kit-specific one.

Days 30–60. The Definition of Verified gets stood up and filled in with the team's actual work types. The verification-debt and comprehension-debt audit signals get checked once, together, against real recent work, then folded into ongoing review and retro conversations rather than run as a separate standing ritual.

Days 60–90. The Agent Incident Runbook gets filled in, before an incident, not after one. By the end of this window, the standing cadence above should be running as one system: 1:1s weekly, review on its own cycle, retro right behind it, feedback continuous underneath all three — not four separate things happening in isolation.

Ongoing, past day 90. The AI Operating Model gets revisited quarterly, and again after any agent incident, per its own ruled line. Nothing else on this calendar has a defined revisit cadence yet; that's a fair question to raise at your first quarterly Operating Model revisit, not something this calendar decides for you.

This is a fit question, not a recipe

The shape above is a strong default, not a finished map of your specific org. Where your team actually sits against it, and what that's costing, is what the Engineering Ways of Working Diagnostic is built to find.