THE ENGINEERING MANAGER'S Field Guide

Tactics for Building, Leading, and Scaling Tech Teams

Read it on a Sunday
Use something new by Monday

Stephane Moreau

Content

CHAPTER 0

This Isn't Just Another Management Book

WHO SHOULD READ THIS?

If you want to become a manager:
This will give you a strong base. It'll help you spot the challenges that no one warns you about. The stuff that doesn't show up in job descriptions, like how hard it is to stop coding, or how weird 1:1s can feel sometimes and what to do about that.

If you've been managing for a while:
Sometimes you need fresh eyes. Or a better way to explain something to your team. Or a reminder of what matters when everything feels urgent. You'll find that here.

If you lead other managers:
You want your managers to share a common way of thinking. A way that builds trust across teams. This can be your go-to manual. Use it with your leads, use it during training, use it when someone's stuck.

WHAT YOU'LL GET

Three simple things:

- Short, direct chapters that you can read in five minutes.
- Tools and tips that you can try right away.
- A shared language you can use with your team.

That's it.

WHY IT'S WORTH YOUR TIME

Being a good manager is hard. And it doesn't always feel rewarding. But it's one of the few jobs where how you show up can change someone's career - and life.

This book won't solve everything for you. But it will help you see clearer. And act faster.

CHAPTER 1

Mindsets That Separate Great Engineering Managers from Average Ones

Most people who step into Engineering Manager roles think it’s all about frameworks, tools, and healthy Jira boards.

That’s wrong.

Great management starts in the mind. And if your mindset sucks, no fancy process will save you.

Here are the six mental models that truly matter - and how to put them to work.

You’re there to serve

“Servant leadership” sounds like corporate crap. But it’s the opposite of that.

It means you stop trying to be the smartest person in the room and start clearing the path for others to win.

Here’s how that looks in real life:

- You don’t hand out solutions. You ask the right questions.
- You don’t micromanage. You unblock.
- You don’t focus on who’s late on their delivery. You focus on why they’re stuck.

Your team’s job is to build. Your job is to help them do it better.

Own everything

The feature is late? The customer is mad? The team is confused?

That’s on you.

Radical ownership means you stop blaming, start looking inward, and ask, “What could I have done earlier to change this?”

It doesn’t mean beating yourself up. It means asking better questions.

When you take full ownership:

- Your team sees it - and starts doing the same.
- Problems get fixed faster.
- Excuses disappear.

Blame kills teams. Ownership builds trust.

Never stop learning

You don’t need to be an expert in every new tool or trend.

But if you’ve stopped learning? You’ve already fallen behind.
Management is a skill that grows.

Here’s how to keep growing:

- Read 10 minutes a day on leadership.
- Join a Slack group with other managers.
- Ask other leads what’s working for them.

Don’t copy what they do exactly. Steal the good stuff, experiment, leave the rest.

Use empathy

This isn’t “being nice”.

It’s listening with real attention, asking questions before reacting, and seeing problems from your team’s side.

Example: If someone keeps missing standups, you don’t assume they’re lazy. You ask what’s going on. You might find out they’re caring for a sick parent or burning out quietly.

Empathy lets you manage and lead people, not just tasks.

And people are what matter.

Trust the numbers

You’re not a data scientist, but you still need to pay attention to a few simple numbers.
Things like:

- How often do we deploy to production
- How many bugs do we get
- What are your users saying

But don’t throw 50 charts at your team. Pick the 3-5 that matter, explain why they matter, and use them to have better discussions and focus.

Don’t code so much you forget to lead

This one’s tricky.

You used to write code. You were good at it. You miss it. So you jump into a pull request to “help”.

But too much coding means your team doesn’t get what they really need: feedback, clarity, and direction.

Ask yourself:

- Am I writing code to avoid hard people problems?
- Does my team really need me here, or am I just comfortable here?

Stay close to the work - but don’t let it replace the real job: leading.

CHAPTER 2

The Engineering Manager's Toolkit

5 simple frameworks you should know

If you're not using at least some kind of framework to make decisions, organize work, or get clarity, you're making life way harder than it needs to be.

These tools aren't here because they're trendy (in fact they're not trendy at all!).

They're here to save your sanity.

RACI: Who's doing what, exactly?

Ever watched a project drag because no one knew who was actually supposed to do the thing?

Yeah. That's what RACI fixes.

- Responsible – The person doing the work.
- Accountable – The person making sure it gets done right. This is usually you.
- Consulted – People you ask before doing something.
- Informed – People who just need to know what happened.

Use this for launches, team-wide projects, or whenever you've got too many cooks in the kitchen.

OKRs: Work toward something

Engineers are great at working on hard problems.
But that doesn't mean going in the same direction.
OKRs can give you a reason - a common goal.

- Objective - A clear goal. "Make mobile feel faster."
- Key Results - How we know it's working. "Crash rate below 0.5%," "+20% in session time", etc.

It's not magic. It's just direction.

North Star Metric

A North Star Metric is the one number that matters more than the others.

- For Airbnb: Nights booked.
- For YouTube: Hours watched.
- For your team? That's on you to decide with your team.

One number can't tell the whole story of course, but it can make certain decisions easier.

Lean + Agile (people's least favourite)

Think like this:

- Smaller releases = faster feedback.
- Regular retros = less repeat mistakes.

That's it. I am not asking for much more than that here.

Scrum, Kanban, sprints, whatever. The main idea is to keep moving, keep learning and improve as we go.

Decision models

Half the drama in engineering teams comes from bad decisions. Or also, no decision.

You need a model. Try these:

- DACI
    - Driver – Owns the decision.
    - Approver – Gives the green light.
    - Contributors – Share input.
    - Informed – Hear about it after.

- Know Your Mode
    - Consensus – Everyone agrees. Slow, but safe.
    - Consult – Get advice, then decide.
    - Command – You just call it. Use sparingly.

Clarity isn't optional. It's the only way people can trust the process, even if they disagree.

Use your time efficiently

One minute you're in a 1:1. The next, you're in a crisis meeting about an outage. After that, someone needs your opinion on when the project will be delivered.

You blink, and it's 6 PM. You barely had any time to prep for that other meeting you have tomorrow, and your brain feels like it just went 12 rounds.

Time management isn't just about being organized. It's about staying sane.

Use priority quadrants

Stephen Covey (author of 7 Habits Of Highly Effective People) gave us a simple way to stop drowning in tasks.

Draw a box. Cut it into four parts:

- Quadrant 1: Urgent + Important - Crisis bugs. Outages. Things that can't wait.
- Quadrant 2: Not Urgent + Important - This is where your long-term lives - roadmaps, hiring, team culture.
- Quadrant 3: Urgent + Not Important - Slack pings. Some status reports. Other people's "emergencies".
- Quadrant 4: Not Urgent + Not Important - Random tab-hopping. Endless email scroll. Useless meetings.

You have to balance tasks from each quadrant. Don't neglect any of them.

Block your time

Your day gets wrecked when meetings keep popping up.

- Pick a window for meetings. Stack them.
- Pick a window for deep work. Guard it like a dragon guards gold.
- Don't mix the two.

This one change will give you more focus.

The 2-minute rule

If it takes under two minutes? Just do it.

- Reply to the quick email
- Share the doc
- Make the one-line Jira update

Small tasks stack up fast. Kill them early.

Stop doing everything

I get it. You were a senior dev. You maybe still write better code than most.

But now your job is not to do that - it's to be a multiplier.

Here's how you can do this:

- Delegate on purpose: Ask yourself "Can someone else do this 80% as well as I can?" If yes, it's gone.
- Let people lead: Let your senior engineers own the tech choices. Stop jumping into every decision.

Make faster decisions

Decisions can take up way too much time. Speed them up with this 3-step trick:

- Write down pros, cons, and likely impact
- Get input fast (one thread, one meeting, no more)
- Set a deadline to decide

No more three-week Slack debates about renaming a microservice.

Tools are here to help you

Use your tools to your advantage

- Mute non-urgent Slack channels
- Turn off email alerts
- Only check messages at set times

For repetitive tasks use scripts, bots, or tool integrations.

Recovery time is non-negotiable

Being a manager is a lot for your brain drain. Build in recovery time or you'll crash.

Try this:

- End your day with 30 minutes of cleanup and prep
- Take a real lunch (no screens)
- Step outside. Move. Breathe.

Your output depends on how quickly you can reset.

CHAPTER 3

Building & Fostering High-Performing Teams

Culture

Culture isn’t the posters on the wall. It’s not the all-hands pep talks. It’s what actually happens when nobody’s watching.

It’s how your team handles stress. How they give feedback. How they react when something breaks on a Friday afternoon.

If you shape it, it can turn your team into a winning one.
Let’s talk about how to do that.

Why culture matters

A strong culture means:

- People help each other without being asked
- Hard conversations don’t get buried
- New hires feel like they belong on day one

A weak one is where you see blame, churn, and people being defensive as a pattern.

Culture is the team’s default setting. You want that setting to work for you, not against you.

Make the values clear

Don’t just say, “We have good communication”

Be more specific:

- “We raise red flags early.”
- “We learn from mistakes out loud.”

Pick values that actually matter for your team and your work. Don’t copy the latest tech unicorn’s list.

Psychological safety isn’t optional

Google studied top teams. The best ones all had one thing in common: people felt safe speaking up.

- Practice inclusivity
- Make sure everyone’s heard in meetings
- Admit your own mistakes first
- Praise people who question assumptions
- Don’t let loud voices dominate meetings

When folks aren’t scared to be honest, problems get solved faster. Ideas get better. Trust builds.

Recognition works (and it’s free)

People want to know their work matters. You don’t need a budget to make that happen.

- Use a #shoutouts or #kudos channel
- Call out small wins in standups
- Celebrate the teammate who merged a tough PR

It’s simple. You get more of what you reward. So reward what you want to see more of.

Freedom and accountability

Great teams feel trusted, and they also deliver.

Here’s how to balance both:

- Let people pick tools or approaches
- Be crystal clear on deadlines, quality, and expectations
- Follow up

You don’t want to micromanage. You want to support execution. And work with the people in your team.

Your behavior sets the standard

You are the culture thermostat. People will copy what you do.

If you skip retros, others will too.

If you show up prepared and respectful, that spreads.

Make sure your actions match your values - every single day.

Team rituals

Every team has habits. Some are helpful. Others waste time and slowly kill morale.

The goal isn’t to schedule more meetings. The goal is to set up lightweight rituals that give your team clarity, speed, and a little bit of rhythm.

Let’s get into the ones that actually move things forward.

Sprint planning

If your planning meetings feel like an endless slog, you’re doing it wrong.

Here’s how to keep it lean and useful:

- Come prepared. The team should already know the top priorities.
- Let the team choose. Don’t assign tasks like a project manager from 2006. Ask: “What do you feel good about picking up this week?”
- Define what “done” means. Don’t leave this open to guesswork. Finalize acceptance criteria together.

Sprint planning should be a pit stop, not a full-blown conference.

Daily standups (that stay under 15 minutes)

The classic three-question standup still works:

- What did I work on yesterday?
- What am I doing today?
- Is anything blocking me?

Keep it moving. If someone brings up a deep issue, park it. Solve it after the standup with just the people who need to be there.

Demos

Wrap up sprints with a quick demo. You can have some code walkthroughs every now and then but this should generally:

- Talk user impact. What does this mean for someone using the product?
- Keep it casual. Think show-and-tell, not TED Talk.

Retrospectives

Retrospectives don’t have to be heavy. But they do have to be honest.

Here’s a simple structure:

- What went well?
- What didn’t?
- What will we do differently next time?

Keep the tone: honest, forward-thinking, and blame-free. This is about getting better, not calling people out.

Collabs

Once a month (or even once a quarter) grab people from product, design, or data and work on something together.

Some of the best ideas come from unexpected pairings.

Try things like:

- Mini hackathons
- Shared problem brainstorms
- Quick build sessions

Keep it focused and fun.

Social stuff

Strong teams are built on trust. And trust doesn’t grow only through Jira tickets.

Try:

- Lunch & Learns. Let someone show off a tool, hobby, or idea.
- Remote coffee chats. 15-minute optional calls with zero agenda.
- Celebrate wins. Shout people out. Share a meme. Drop a GIF. Doesn’t matter - just mark the moment.
- Organise team activities that people want to participate in.

Don’t let rituals turn into red tape

Rituals should help you and the team.

Here’s how to keep them fresh:

- Ask the team what’s working. Use retros to audit your rituals.
- Cut or combine. If two meetings are doing the same thing, merge them.
- Make some optional. Let people skip the ones that don’t serve them.

Here’s a solid weekly flow you can start with:

- Monday: 30-minute sprint planning
- Tue–Fri: Daily 15-minute standup
- Friday: Demo session, and maybe a retro if you’re on a weekly cycle

That’s it.

Keep the rituals lightweight. Make them useful. Let them evolve.

If your team’s not getting value from a ritual, it’s not a ritual you want to have.

Conflict

You can have the nicest, most collaborative team in the world and still have conflict. That’s not a sign of dysfunction.

It’s a sign that people care.

Handled well, conflict makes your team stronger. Handled poorly, it will kill morale and definitely slow down your delivery.

Let’s talk about how to use conflict to make better decisions.

Teams fight and it’s ok

Most engineering drama doesn’t come from bad intent. It comes from normal stuff like:

- Different technical opinions. Everyone has their own “better” solution.
- Stress from tight deadlines. When the pressure is on, small things turn into shouting matches.
- Bad communication. People assume different things. Boom - misalignment.
- Personality mismatches. Some people just don’t vibe.

The fix isn’t pretending everything’s fine. The fix is knowing how to handle tension without letting it spiral.

Use SBI

SBI stands for Situation, Behavior, Impact. Use it when you need to give someone feedback without making it personal.

Here's the format:

- Situation: "In yesterday's design review..."
- Behavior: "You interrupted John several times."
- Impact: "It made it hard for him to explain his idea, and it came off as dismissive."

This isn't fluffy HR talk. It keeps things specific and anchored in facts. No assumptions. No attacks.

Focus on the problem, not the person

Let's say two engineers are arguing about which backend architecture is better.

You can step in and say, "Forget who's right, what do the numbers say? What solves the user problem best?"

That one shift can change everything. You move from ego to outcomes.

Make people actually listen to each other

If someone feels unheard, they'll keep arguing until they explode, or check out entirely.

One way to fix that: force a pause.

- Ask each side to repeat what they think the other person said.
- Then ask, "Did they get it right?"

Nine times out of ten, people realize they were arguing past each other.

Make disagreement safe

If speaking up means getting slammed or ignored, people go quiet. That silence isn't peace, it's disengagement.

As a manager, set the tone:

- Disagree all you want, but stay respectful.
- Step in fast if someone crosses the line.
- Praise people who raise concerns with grace, even if you don't act on it right away.

Safety means having a system for hard conversations that doesn't blow up the team.

Write things down

If a debate got intense, or a decision wasn't obvious, document it.

- Summarize what was discussed.
- Note the decision.
- Call out next steps and who's doing what.

Now no one can "forget" what was agreed. You avoid drama later.

Know when to escalate

Some conflicts are above your pay grade, or too big to settle.

If two leads can't align, the best thing might be to loop in the other people: your manager, CTO, whoever can make the call fast.

No shame in it. Don't let things stall for weeks.

What about personal friction?

Personal friction is a little bit harder to deal with.

What I recommend:

- Pull people aside. Use the SBI framework.
- Coach them. Remind them: being in the team means supporting others.
- Set clear lines. Respect is non-negotiable. If it keeps happening, it's a big issue that warrants performance conversations.

Stay ahead

Want to deal with less conflict? Be proactive.

- Weekly 1:1s. Small frustrations show up here first. Ask about them.
- Team norms. Write down how your team handles feedback.
- Watch for early signs. Eye rolls in meetings, passive-aggressive Slacks, weird silences - don't ignore them.

Conflict's not your enemy. Silence is.

CHAPTER 4

Strategic Roadmapping & Alignment

Connect engineering to business goals

If you think your job as an Engineering Manager is to ship code, you’re wrong.

Your job is to make sure that code means something. That it moves something forward - revenue, signups, user love, whatever your company and team actually care about.

Most teams don’t struggle with writing code. They struggle with writing the right code.

Engineers need to know the "why"

When devs don’t know why they’re building something, here’s what happens:

- They lose interest.
- Priorities get messy.
- The solutions are not quite right.

But when they get the full picture, you’ll see better ideas, smarter trade-offs, and faster delivery.

Just answer one question: How does this project help the business?

Make the strategy real

Let’s say the company wants to grow in a new region.

Instead of saying, “This is a top priority” say this:

“We need to integrate with Country X’s payment system. That unlocks a new market that could bring in 20% more users.”

You turned strategy into action. That’s what your team needs from you.

Use numbers people actually care about

Not all numbers matter.

Find a few that do. For a video app, that might be:

- Average watch time
- Monthly viewers
- Subscription renewals

Then show the link. “If we reduce buffering by 50%, people stay longer. That’s money.”

Tie every project to a real outcome. If your team uses OKRs, great.

Talk about the bigger picture… constantly

Don’t assume people get it. They don’t.

In sprint planning, in weekly syncs, even in Slack, remind people:

“This feature helps our conversion rate. That’s our top goal this quarter.”

“Tech Debt vs. Features”

This isn’t a battle. It’s a false choice.

Yes, the business wants shiny new stuff.

But old, messy code slows everything down - features, fixes, releases, you name it.

So frame it like this:

- “Cleaning this up now saves 2 weeks later.”
- “If we fix this, we can scale faster next quarter.”

Make the case in plain terms.

Be the translator between teams

Product wants X. Marketing wants Y. Sales wants Z.

You’re the one who figures out what’s possible, and what’s worth doing first.

So ask each team: “What outcome are you hoping for and why?”

Then bring that back to engineering and break it into plans the team can actually commit to.

Communicate relentlessly with other teams.

Plans change. That’s fine.

You might have a beautiful 12-month roadmap. Don’t get married to it.

Markets shift. Priorities change.

Check in every month or quarter and ask: Is this still the best use of our time?

If it’s not, adjust. And tell your team why. Nobody likes whiplash - but they can handle change if you’re clear.

Celebrate like it matters (because it does)

Don’t let wins fade into the background.
When your team ships something that moves a real number - say so.

- “Recommendation engine led to 12% longer sessions. Great work.”
- “Cloud spend dropped by 15%. That’s all you.”

This stuff fuels morale. It also shows the company what engineering is really worth.

Get good at prioritization

There’s always more to do than time to do it. Urgent bugs, flashy new features, tech debt, internal tools - all yelling for attention.

If you don’t have a system, you’ll make the wrong choices and usually, the loudest voice wins. That’s not strategy.

The 80/20 game

The 80/20 rule (Pareto Principle) is your best friend.

Ask yourself: Which 20% of work will get us 80% of the results?

- Fixing one slow page could boost conversions way more than building a whole new feature.
- Cleaning up a flaky test suite might unblock dozens of tasks later.

Start there. Always.

Say “not now” and mean it

Don’t treat everything as equally urgent. Label features like this:

- Must-Haves → We break something (or the law) if this isn’t done.
- Nice-to-Haves → Would be cool. Can wait.
- Not Now → Good idea, just not today.

Put “Not Now” stuff on a parking lot list so it doesn’t vanish, but it doesn’t distract.

Tech debt = Future pain

Ignore tech debt and you pay with time later.

- Refactors that speed up development? Worth it.
- Cleanup that prevents bugs? Absolutely.

Set a fixed chunk of each sprint (ge. 20%) to work on tech debt.

And don’t just say “this is tech debt”. Do your research so you can say things like: “If we don’t fix this, we’ll waste 3x more time in 2 months”. That’s how you get buy-in.

Use data

Decisions shouldn’t be made by whoever complains the loudest.

Ask:

- How many users are affected?
- How bad is the impact?
- Is this blocking revenue?

Example: If 80% of users hit a slow checkout at peak times? Fix it now.

If 2% have a minor UI glitch? Add it to the backlog.

Talk to other teams, but don’t get pushed around

Sales, marketing, product, they all have “urgent” work for you.

Your job: separate real urgency from noise.

What happens if we don’t do this right now?
If the answer is vague, it can probably wait.

You’re not being difficult. You’re protecting your team’s focus.

Force clarity

Forget marking everything P1. That’s useless.

Use a tool or board where tasks are stacked - top to bottom, no ties.

Even better? Do a short ranking workshop:

- Everyone votes.
- Debate.
- You lock in the order.

Now you’ve got buy-in and clarity.

Stop trying to do everything

If you’re working on 10 things at once, none of them are getting finished wel or on in a timely fashionl.

Limit active tasks.

Try focus sprints: pick 3 things, crush them, then move on.

Shipping 3 solid things beats juggling 10 half-done ones every time.

Handle stakeholders

If you ignore stakeholders, things break. Maybe not features. But trust does, timelines. And your sanity.

Here’s how to keep your stakeholders in the loop without drowning in meetings, emails, or Slack chaos.

Kill surprises before they start

Most blowups happen because someone expected X and got Y.

Get ahead of it.

- Scope → What exactly are we building?
- Constraints → Time, budget, tech limitations.
- Risks → External dependencies, tech risks, missing info.

Use a simple doc or quick meeting to walk through it all. End with: “Do we all agree on this?”

That line alone prevents so many future headaches.

Set a rhythm and stick to it

You don't need a fancy dashboard (though those help). What you do need is a habit.

Pick a cadence (weekly, bi-weekly, monthly) and share:

- What's done
- What's next
- What's blocked
- What you need from them

Send it via email, drop it in Slack, or run a 15-minute call. Just keep it short and consistent.

Be honest. Always.

Don't overpromise. People can smell it.

Say what's wrong. Say how you're fixing it.
"We hit a bug in the API integration. It'll push us back by a week. We're fixing it now and will confirm the new timeline by Friday."

You don't lose trust by missing a deadline. You lose it by hiding that you missed it.

Get them talking

Stakeholders aren't always clear on what they want.

So ask better questions:

- "What does success look like to you?"
- "Which group of users is most impacted?"
- "If we didn't do this, what happens?"

Then repeat it back. "Just to confirm, you want the dashboard to show real-time sales per region?"

Speak their language

Different people care about different things. Don't give everyone the same update.

- Executives → Big wins, deadlines, risks.
- Engineers → Dependencies, tech blockers, design choices.
- PMs & Designers → User outcomes, feature scope.

Tailor your message or risk losing their attention and/or their trust.

Draw the line

Feature creep is real.

You need to set boundaries without sounding like a blocker.
“If we add that now, we miss our current deadline. We can either build a lighter version or delay the release. What fits your goal better?”

They’ll respect you more when you offer trade-offs instead of just saying “no”.

Handle pushback

Conflict happens. Don’t panic.

- Listen → Maybe there’s pressure you didn’t know about.
- Explain → Walk them through the trade-offs or blockers.
- Negotiate → Can you ship a basic version first and iterate later?

People fight less when they feel heard. And when you’re calm, confident, and clear.

Celebrate the wins

Don’t just show up when things go wrong.

Show off progress. Run a short demo. Drop a note when a key feature ships.
“The new feature just went live - special shoutout to [stakeholder] for helping shape the design!”

These little moments build trust. And they buy you space when things do get hard.

CHAPTER 5

Hiring & Onboarding

Spot A+ talent

Hiring isn't about filling seats.

It's finding the people who raise the bar for everyone around them. Not the ones who just check boxes. The ones who build things you didn't think were possible. The ones who care. Who don't need hand-holding. Who challenge the work (and the team) to be better.

Know exactly what you need

Most hiring mistakes start here: you don't know what success looks like.

Not the title. I am talking about actual success.

You want someone who will:

- Improve backend speed by 10x?
- Rebuild your design system?
- Mentor your juniors and build up team morale?

Write these down in one page.

You'll also want to include a success profile. 3 things:

- What they'll own
- What skills they need
- What great looks like in 6 months

Clear expectations.

Build a process you can repeat

Interviews should not be improv theater.

They should be structured, consistent, and hard to game.

A basic structure:

- Screening Call: Are they clear, direct, and human? Can they talk through their background in a way that makes sense?
- Technical Round: However you define it based on your needs (leetcode, sys design, pair exercise, etc).
- Behavioral Round: Can they work with others? Handle pushback? Receive feedback? Lead a project?
- Team Meet: Let them talk to future teammates. You'll learn a lot just by watching how this goes with a light agenda.

Test the way they think, not just what they know

You’re not hiring an encyclopedia with legs.

You probably want someone who thinks deeply, solves problems, and explains their thinking like a teacher.

Ask questions like:

- “What would you optimize here, and why?”
- “What tradeoffs are you making with that approach?”
- “What else could go wrong?”

Don’t just check for the right answer. Check for how they talk about the wrong ones too.

Behavioral interviews are important

Everyone’s got a portfolio. Cool.

What you need to know: how they handle conflict, failure, and feedback.

Try questions like:

- “Tell me about a time you disagreed with your team.”
- “When was the last time a project you led failed? What happened and what did you learn?”
- “What’s something you were wrong about?”
- “What’s some feedback your manager or a peer has given you?”

These questions tell you more than any Leetcode question ever could.

Sell like a marketer

Top talent has options. If you’re not showing them why they should join you, someone else will.

What to talk about:

- Your team’s strengths - mentorship, flexibility, autonomy
- Real challenges - what’s broken, what’s improving
- Growth - they want to learn, not stagnate

Be real. Be honest. The right ones will lean in.

Don’t let bias mess this up

It’s way too easy to favor someone who looks, talks, or thinks like you.

That’s a trap.

Avoid it by:

- Using the same set of questions with everyone
- Bringing in interviewers from different backgrounds
- Writing feedback before you debrief with interviewers

Bias isn’t always obvious. But it’s always there unless you fight it.

Watch for these red flags

Doesn’t matter how good their code is, question a positive decision is you see:

- Poor communicator - Can’t explain things clearly? Huge problem.
- Blame game - If it’s always someone else’s fault, it’ll be yours next.
- Stuck in their ways - If they trash new ideas without real reasons, they’ll slow everyone down.

Hire for the mindset

Great engineers are learners first.

The ones who ask “what else can I learn?” instead of “what’s my title?”

Ask this:

- “What’s something you’ve learned recently?”
- “How do you keep your skills sharp?”

The people who light up when they answer? That’s your A+ talent.

Onboarding

Hiring someone is just the beginning.

You’ve got them in cool, but if you drop the ball now, that shiny new hire becomes a confused, underperforming stranger wondering if they made a mistake.
Start owning parts of the roadmap

Start before day one

Don’t wait for the first Zoom call to say “welcome”.

You can make the first impression before their laptop even ships.

What to send ahead:

- A quick welcome note
- A short guide to tools, culture, and what to expect
- A “starter pack” with hardware, accounts, and docs

Use a 30-60-90 Plan (Always)

Structure = clarity. Clarity = confidence.

Break their first 90 days into clear blocks:

- Day 1–30: Setup, intro calls, codebase tour, ship a small bug fix
- Day 31–60: Build a simple feature, do code reviews, learn the system
- Day 61–90: Own a small project, propose an idea, pair with different teammates

This kills the “what should I do?” loop. And shows you actually planned for them.

Assign a buddy

No one wants to be “that person” who keeps asking where stuff is.

Give them a go-to. Someone who knows the ropes and actually likes helping.

Pick someone who:

- Has time (don’t dump this on your busiest senior)
- Is patient, clear, and friendly
- Understands how the team works, not just the codebase

The buddy relationship matters a lot.

Teach the culture

Tools are easy. Culture is what trips people up.

They don’t know if decisions happen in meetings or Slack. They don’t know who actually calls the shots. Or if it’s okay to push back.

Fix that:

- Share unwritten rules (how we make decisions, ask for help, etc.)
- Set up casual coffee chats or “get to know you” threads
- Point them to the right Slack channels (yes, even the meme one)

You want them to feel like they belong.

Let them touch real work (early)

Don’t baby them. Let them get their hands dirty, fast.

How:

- A small bug or tweak in Week 1
- Pairing with someone on a real feature in Week 2
- Joining planning or retro calls early on

People learn by doing. Not by sitting through hours of docs.

Fix your docs (they’ll thank you)

Good documentation is a cheat code.

And your new hire is the best person to tell you what’s broken in it.

What to document (if you haven’t already):

- How to set up the environment
- How deployments work
- Coding conventions
- System architecture (keep it simple)

Then ask your new hire to spot gaps. You want to get their fresh perspective.

Check in - frequently

Don’t just wait for the 90-day review. That’s way too late.

Run weekly or biweekly check-ins focused on:

- What’s working
- What’s confusing
- What questions they have

These convos can be short but powerful. You’ll catch issues early and build trust without trying.

Celebrate small wins

First PR? First bug fix? First deploy? Hype them up.

Where to do it:

- “Kudos” Slack channels
- Team meetings
- Casual shoutouts in standup

This isn’t giving fake praise. It’s to create momentum. Confidence builds fast when you recognize effort.

Example timeline

Week 1:
- Meet the team
- Fix something tiny
- Join standups
- You check in midweek

Week 2:
- Pair on a feature
- Add/update docs
- Join a retro
- Start feeling part of the rhythm

Weeks 3-4:
- Own a task end-to-end
- Push to production (yes, really)
- Get feedback
- See the whole dev cycle in action

Weeks 4-6:
- Build confidence
- Give feedback in retros

Keep your engineers

Hiring gets the glory. Retention does the real work.

If your engineers are bouncing after 12 months, you’ve got a leak in the boat. Doesn’t matter how great your recruiting is. If people keep leaving, your team never levels up.

Retention isn’t just about money

Replacing a great engineer costs way more than their salary.

- You lose domain knowledge
- You lose trust between teammates
- You lose time rebuilding what was already working

And let’s be honest - losing someone strong can take the air out of the room.

Help them grow (their way)

Not everyone wants to be a manager. Some want to go deep into the tech. Others want to lead. You won’t know unless you ask.

Build a real plan:

- New skills to learn (Kubernetes, TypeScript, leadership - you name it)
- Stretch projects that challenge them without setting them up to fail
- Career path from where they are now to where they want to be next

One-size-fits-all plans don’t work. Build it with them.

Talk about careers before they ask

Don’t wait for the “I’m thinking of leaving” chat.

Use extended 1:1s to ask:

- “What do you want to get better at?”
- “Is your current role helping you grow?”
- “What’s something you’d like to try next?”

Just honest conversations.

Recognition still matters

Engineers aren't machines. They need to know their work means something.

Make it visible:

- Give shout-outs in team meetings or Slack
- Offer small spot bonuses or gift cards - just enough to say "we saw that"
- Be clear about the promo path - what it takes, what's next

You'd be surprised how far a little recognition goes.

Let them own stuff

Nothing kills motivation faster than feeling like a factory worker.

Flip the script:

- Let them propose tech changes or feature ideas
- Involve them in architecture decisions
- Encourage them to lead small projects or experiments

More ownership = more pride. And better work.

Make learning easy (and expected)

Top engineers want to stay sharp. If you don't help them grow, they'll find someone who will.

Support their learning:

- Budget for a course, conference, or certification
- Do internal "show and tell" sessions
- Block time for deep learning - not just silly tutorials

Make it normal. Make it expected. Growth should be part of the job.

Stop burnout before it breaks them

Burnout doesn’t announce itself. It comes in quietly.

What to watch for:

- Late nights, even when it’s not high-pressure time
- Short tempers or mood shifts
- Work quality slipping for no obvious reason

How to help:

- Reassign tasks if they’re overloaded
- Offer breaks, PTO, or schedule flexibility
- Have real talks - not performance reviews, just check-ins

You can’t build great things with people who are running on fumes.

Create actual camaraderie

People don’t just stay for the work. They stay for the people.

Build a team:

- Set up low-effort socials
- Encourage buddy systems or mentorship pairings
- Let juniors learn from seniors - and let seniors rediscover why they started

It doesn’t have to be forced fun. It just has to be real.

CHAPTER 6

Performance & Growth Management

Measure impact & success

Metrics are everywhere.

Most of them though don't actually tell you if your work matters.

Let’s talk about what does.

Output vs. Outcome

Output is what you ship.

Features, bug fixes, story points. Easy to measure. Easy to fake progress with.

Outcome is what actually changes for your company and customers.

More users stick around. Sales go up. Customers stop yelling in support tickets.

If your team rolls out 12 new features and nobody uses them that’s not success.

You need to be asking:

- Who’s using what we built?
- Does it solve their problem?
- Would they miss it if it disappeared?

Leading vs. Lagging indicators

Lagging indicators are the ones that show up after the fact like revenue, churn, market share change, etc.

Leading indicators show what’s happening right now.

For example:

- PR throughput
- Number of bugs found
- Sprint velocity (if you track that sort of thing)

You need to use both.

Quantitative vs. Qualitative

Quantitative = the numbers.

Page load time. Error rates. Ticket resolution time.

Qualitative = the stories.

“My day is 10x easier now.”
“This feature saves me an hour every shift.”

The best feedback is often a mix. You want metrics plus input from real users.

Tie it to business goals

If the company wants user growth, your work should help that.

Some ideas:

- Track how new features affect sign-ups
- Watch how downtime hits churn
- Follow how fixes for top pain points change customer reviews

Your team’s work should be moving a business needle.

Efficiency + Quality

Ship fast and clean. That’s the goal.

Some metrics to consider:

- Cycle Time - How long from code to prod?
- Bug Rate - How many things break?
- Uptime / Incidents – Can users trust your product?

Don’t just track how much you build, you need to also track how good it is.

Team health

You want to have some way to assess how your team feels.

How to do that:

- Health check surveys - Are people feeling challenged, supported, heard?
- Attrition - Are people leaving? Why?
- Participation - Do people speak up in retros? Who always stays silent?

Use dashboards that matter

It’s easy to get lost in a wall of charts. Most dashboards are just decoration.

Instead:

- Pick a few metrics that actually help you make better decisions
- Let the team help choose them
- Ditch the rest

If nobody looks at it? It’s not worth tracking.

Review. Adjust. Repeat.

What worked last quarter might not work this one.

Revisit your metrics often. Ask:

- Are we still measuring the right stuff?
- What’s noise vs. what’s useful?
- What’s missing?

The feedback playbook

Your team can write clean code, ship fast, but trust me that’s only temporary if you can’t give feedback to each other.

Most feedback sucks because it's either too vague, too late, or too mean. But done right, it makes people better, faster, and more aligned.

Why feedback exists

Feedback is not about who’s right and who’s wrong.

And it definitely is not your chance to show who’s boss.

It’s simple:

- Help someone improve
- Keep the team moving in the same direction
- Make sure nobody feels like they have no direction

Do it now. And be specific.

Timing matters.

Don’t wait six months to say, “Back in Q2, that didn’t land well”.

Feedback works best when it’s close to the moment. People remember what happened.

They know the context. And they can actually do something about it.

Specificity matters as well.
For example:
“In your last PR, there were a few comments about X that went unaddressed. Let’s make a plan for handling those moving forward.”

That’s feedback people can act on.

Positive isn’t optional

If all you give is critique, people tune out and they will stop caring.

Positive feedback builds trust. It shows people what good looks like. And it gives you balance for when you need to push harder.

You can say things like:

- “That test coverage tool you added was perfect.”
- “Thanks for stepping up during that outage, you stayed calm and focused while fixing the issue.”

Use a simple framework

SBI = Situation. Behavior. Impact.

It keeps things clean and focused.

- Situation: “In yesterday’s planning meeting...”
- Behavior: “You talked over Michael during his update.”
- Impact: “It derailed the discussion and we didn’t finish the agenda on time.”

Then pause. Ask for their take. Work together on how to handle it better next time.

Feedback goes both ways

If your team can’t give you feedback, you’re not a good leader.

Ask things like:

- “What is one thing you see I could do better?”
- “What advice would you give me to support you better?”

And when they tell you something uncomfortable, sit with it. Let it land. That’s how you show you actually want to grow too.

The hard stuff

Sometimes things get tricky. Performance issues, conflicts, drops in output.

Here’s how to handle it:

- Always in private - Never give feedback to someone in front of others.
- Lead with care - “Anything going on that’s making work tougher lately?”
- Plan, don’t punish - “Here’s where we are. Let’s build a step-by-step plan to fix it.”

The goal is to help.

Know who you’re talking to

People from different backgrounds receive feedback in different ways.

Some want it straight. Others need a little warmth first or they shut down.

Be smart about it:

- Pay attention to how each person responds
- Adjust your tone, but stay honest
- Ask open ended questions in the process to hear their views and opinions

Build a learning machine

Technolofy especially in the AI era is changing - fast.

You want your team to be learning and staying up to date with new tools/frameworks (even if you are not going to use them).

Learning never stops

The minute you feel like you've got it all figured out you might already be behind.

Here's what happens when your team stops learning:

- You solve the same problems the same way
- Hiring becomes difficult because candidates can see it
- Your best people will start looking elsewhere

Teams that keep learning move faster, try new things and stay sharp.

Every role needs a learning plan

During 1:1s, you need to be forming people's goals together.

Eg:

- "Take the lead on two projects this quarter"
- "Mentor a junior engineer to build leadership skills"
- "Facilitate 3 retrospectives in the next sprint quarter"

Then write them down, and make time to track them.

Share what you know (even if you're not an expert)

Lunch-and-learns. Demo Fridays. Ad hoc 20-minute Zooms.

Doesn't matter what you call them, just do them.

What matters is that people are sharing what they learn and what they're working on. Tools they like. Projects they just finished. Stuff they're curious about.

This builds a learning loop you need as a team.

Hack. Build. Tinker.

Give people time to work on weird ideas they have. You can have Hackathons and innovation sprints.

Some examples:

- Refactor a gnarly part of the codebase
- Try a new backend service
- Build a goofy-but-useful bot that will allow us to handle alerts better

Don't micromanage it. Give the team a day or two and let them show off what they built.

This kind of freedom often surfaces better ideas than weeks of planning.

Mentorship is a cheat code

Pair programming = fast learning.

It’s not just about code. It’s about how someone thinks through problems in real time.

You can set up mentor pairs in your team.

And the beauty of it is that it works well both ways: the mentor learns more by teaching, while the mentee benefits from what’s shared.

Offer formal training but don’t force it

Courses. Certs. Online platforms.

Some people love this stuff. Others don’t.

Make it available, not mandatory. Pay for it when you can.

Let people pick what works better for them.

Map the path to what’s next

People want to grow, but they need to see where they’re going.

Build a career framework if your company doesn’t have one:

- Clear titles (Associate, Mid, Senior, Staff, Principal)
- Specific skills and behaviors for each level
- Honest signals about when someone’s ready for the next step

CHAPTER 7

Communication

Storytelling

Facts don’t inspire action. Stories do.

You need to be aligning your team, selling ideas, and translating technical chaos into something non-technical people can actually understand.

That’s where storytelling comes in.

Storytelling wins

People don’t remember data. They remember stories. They remember the why.

If you want someone to care tell a story that has:

- Emotion - Even in code, motivation starts with feeling something
- Clarity - Stories break down messy topics into clean ideas

You’re not just explaining architecture here.

Know who you’re talking to

One story won’t work for everyone.

What lands well with engineers will not with your execs or your Product Manager.
Use the story arc

Every good story has four stages:

1. Setup - “Cart abandonment is at 25%”
2. Conflict - “That’s costing us $2M every quarter”
3. Resolution - “We’re proposing a new gateway to speed up checkout”
4. Impact - “This could boost conversions by 10% - and improve customer happiness”

Don’t overcomplicate it.

Mix data with emotion

Numbers build trust while stories make it stick.

Don’t just say: “Page load dropped by 1 second”
Say: “Now users can check out without waiting, and that one second means 5,000 more orders a month.”

- Use just enough data to prove your point
- Back it with a real human story

Rehearse. A lot.

You wouldn’t merge code without a second pair of eyes reviewing it. Same with storytelling.

Try it on someone outside your team if you can. If they don’t get it, maybe it’s not ready.

Teach your team to tell stories

It’s not just your job to communicate well. It should be part of your team’s DNA.

In sprint demos, retros, even PRs coach your engineers to explain:

- Who was affected?
- What changed?
- Why it mattered?

Lead in remote and hybrid teams

Working remotely or half-remotely isn’t new. But most teams are still winging it.

Slack pings 24/7. Zoom fatigue. Confused engineers wondering why the thing they built last week is already being scrapped.

Remote teams don’t run themselves. You can’t fake culture or communication. Truth is… you have to work harder for it.

Say it twice (sometimes three times or even more)

You don’t get random hallway chats anymore.

So you need to repeat yourself. A lot.

- Post written updates in Slack, Confluence, whatever tool your team uses.
- If you say something important in a meeting, write it down afterward.
- Context is everything. If you want decisions to stick, document them.

Don’t just expect people to remember what you said on a random Tuesday call.

Use meetings (or not)

Not everything needs to be a call.

Sync (real-time)

- Brainstorming
- Sensitive topics
- Fast back-and-forth

Async (not real-time)

- Project updates
- Weekly summaries

Make this your rule: if it doesn’t need a live reaction, it goes async.

Also: Respect time zones and calendars.

Create virtual “bump-into-you” moments

You have to manufacture social moments now. They don’t just happen.

- Random 1:1 coffee chats every week (rotate who gets paired).
- Slack channels just for pets, books, or fantasy football.
- Friday demos, remote games, or meme threads.

These small things matter a lot.

Goals

You can’t watch what your team is doing all day. Don’t even think of trying that.

Instead:

- Make every engineer’s goals crystal clear.
- Set timelines. Deadlines. Ownership.
- Do 1:1s with everyone.

Your job is to give clarity and trust while keeping people accountable.
So you need to find ways of being able to do that.

Don’t let time zones break your team

Working across 5 countries? That’s not an excuse for chaos.

Here’s how you can make it work:

- Set a 2–3 hour overlap window that everyone blocks.
- Rotate late or early meetings if needed (don’t burn out one region).
- Record everything: demos, decisions, status calls.

If someone misses something, they should have a way to catch up fast.

Write it down or it didn’t happen

You can’t run a remote team without documentation. Period.

- Specs: So anyone new can pick up a task and run with it.
- Decisions: Write them down, share them, link them.
- How-tos: From “how to set up the dev environment” to “how to book a day off.”

You need to have docs as your team’s second brain.

Starting a new hire remotely?

You want to have a smooth onboarding.

- Send a “starter pack” with links to everything.
- Assign a buddy.
- Set up short calls with each teammate in week one.
- Check in regularly for the first 30 days.

If you skip this, don’t be surprised when they ghost Slack messages after month two.

Burnout is harder to spot behind a screen

People working remotely often work longer without realizing it. Or they feel isolated.

So you have to look for it:

- Ask about their wellbeing in 1:1s
- Encourage full lunch breaks, afternoon walks, and no-message weekends.
- Watch out for people who stop speaking up, they might be mentally checked out.

Remote work is flexible, but you still need to protect your team from burnout.

CHAPTER 8

Further Exploration

Your journey as an Engineering Manager doesn’t end here of course. There’s always more to learn. Below is a curated list of books, blogs, newsletter, and podcasts to help you continue building your expertise.

Books

I keep a list with all the books I recommend on this page in categories:

- How to get started as a team lead
- How to create the right product
- How to create a great working culture
- How to give the team direction
- How to architect systems
- How to do modern QA
- How to build a larger organisation
- How to Communicate, Influence, and Build Your Brand

Podcasts

- Dare to Lead - provides insightful conversation from guests like organizational psychologist Adam Grant and thought leader Simon Sinek

- A Bit of Optimism - inspiring stories to motivate you to not only be a better leader but to approach issues and problems as a human first

- Take Command: A Dale Carnegie Podcast - conversation-based podcast, offering wisdom from a diverse range of leaders in various industries

- How I Built This - entrepreneurship and business sense from some of the most successful startups and corporations out there

- WorkLife with Adam Grant - find inspiration, learn, and make work not suck

- Coaching for Leaders - Host Dr. Dave Stachowiak talks with well-known authors, business experts, and various leaders to cover timely, relevant, and thought-provoking content, from how to manage up to how to build psychological safety

Blogs & Newsletters

- The Pragmatic Engineer: actionable leadership advice, insider interviews, and market trend analysis for tech leaders seeking insights on Big Tech and high-growth startups
- Lenny's Newsletter: this newsletter shares product, business, and career growth insights
- TLDR: daily free newsletter catering to CTOs, providing quick links to articles on startups, tech, and programming
- ByteByteGo: simplifies complex technical systems, focusing on large-scale system design trends
- Level Up: weekly newsletter that shares experiences in software engineering and management and provides insights on leadership, technology, and organisational processes
- Blog for Engineering Managers: the go-to resource for software engineers transitioning into leadership and for experienced managers looking to optimise their style

Mentoring & Coaching

Find a Mentor or Coach: Look within your network or professional platforms like LinkedIn. I also offer coaching so feel free to reach out to me here.

Offer to Mentor Others: Participate in mentorship programs like Women Who Code, CodeNewbie, or other platforms.

Want more insights like this?

I write every day on Linkedin

CONNECT WITH ME

Stephane Moreau