FROM IC TO MANAGER

How to become “scalable” beyond code ?

DEVSUMMIT
<MOVE_
FORWARD/>

Marcel MARQUEZ

Head of Engineering SCDP/PDP @Adeo

Previously Engineering Manager @Leroy Merlin France

And previously again: a staff and tech leader

Who is this talk for ?

DEVSUMMIT <MOVE_ FORWARD/>

You might be

A senior engineer/architect considering management

A newly promoted Engineering Manager (EM)

A tech lead feeling the pull of people leadership

What you'll walk away with

A new Definition of Done for your own work

Practical delegation frameworks

Tips to stay technically relevant without being a bottleneck

The psychology behind the "new manager bugs"

DEVSUMMIT <MOVE_ FORWARD/>

78%

of engineers report impostor syndrome at the leadership transition

DEVSUMMIT <MOVE_ FORWARD/>

The core misconception

Becoming manager is not a promotion

Management is often seen as a vertical promotion that rewards technical excellence

Reality: Become a manager is a career change, not a step. It requires new skills, rebuild his professional identity and way of working.

OUR JOURNEY TODAY

DEVSUMMIT <MOVE_ FORWARD/>

The IC World

How your typical day look like when you're an individual contributor

The Transition

When to move from IC to management

The Trial

First pain points and frustration as a manager

Scale as a manager

Ideas to move from scaling systems to scaling through people

Take away

Some left tips and tools to go deeper

DEVSUMMIT <MOVE_ FORWARD/>

The IC World

Individual contributor journey

Code written & reviewed

Test passing

Feature deployed in production

Ticket closed ✅

Definition of Done is clear

Dopamine feedback loop
instant

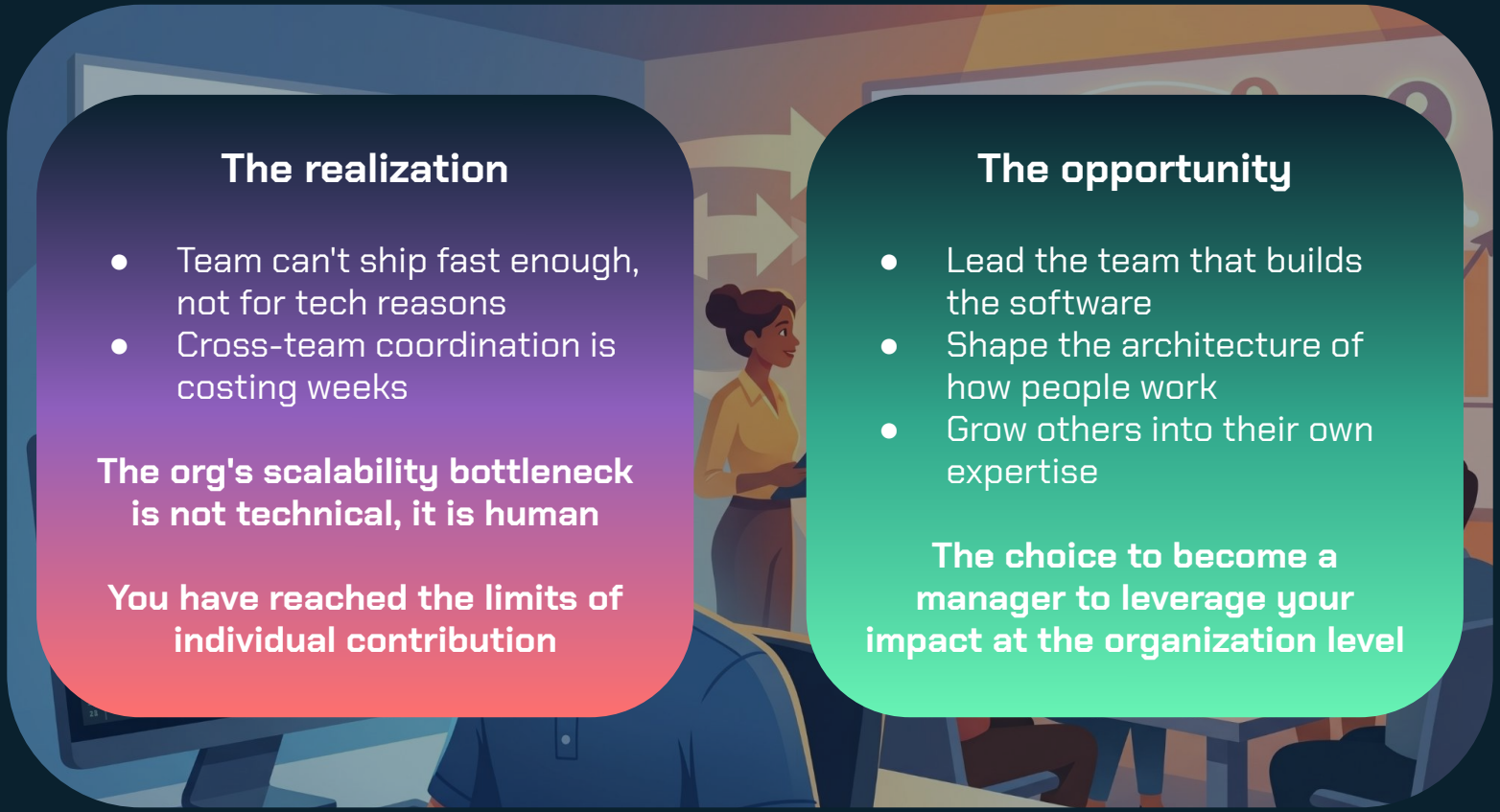
Success criteria
binary & measurable

Impact attribution
direct & personal

DEVSUMMIT <MOVE_ FORWARD/>

The Call to Adventure

Impact outside the team



The realization

- Team can't ship fast enough, not for tech reasons
- Cross-team coordination is costing weeks

The org's scalability bottleneck is not technical, it is human

You have reached the limits of individual contribution

The opportunity

- Lead the team that builds the software
- Shape the architecture of how people work
- Grow others into their own expertise

The choice to become a manager to leverage your impact at the organization level

DEVSUMMIT <MOVE_ FORWARD/>

Refuse the call

Reasons to not make the switch

Fears that are real

- “I'll become technically obsolete”
- “I'll be bad at the soft skills stuff”
- “I'll miss shipping real code”
- “What if I'm not respected without being the best coder?”

But fears are part of the process

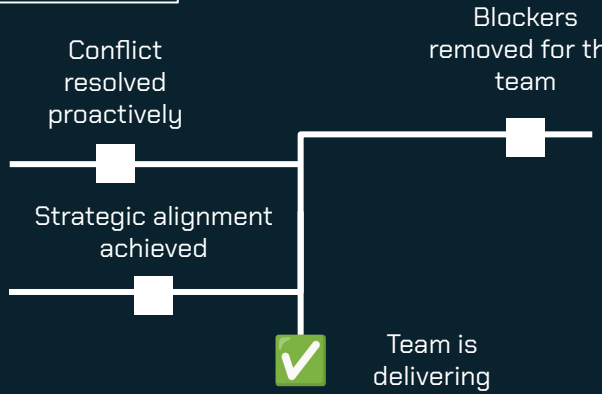
- 78% of engineers report impostor syndrome at the leadership position
- These fears are universal, not personal weaknesses

It’s a signal you're growing, not failing

DEVSUMMIT <MOVE_ FORWARD/>

You make the switch

Congratulation: your DoD just died



Definition of Done is unclear

Dopamine feedback loop
lost

Success criteria
uncertain

Impact attribution
indirect & team related

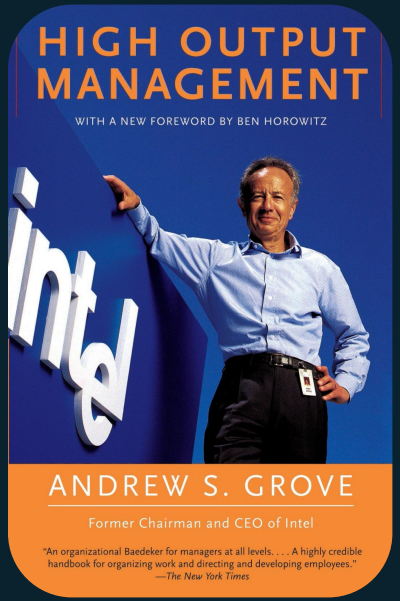
As a manager, your first weeks look all the same, with a difficult question

what did I actually produce ?

DEVSUMMIT <MOVE_ FORWARD/>

Andrew Grove

Former chairman and CEO of Intel



A manager's output =

the output of their organization

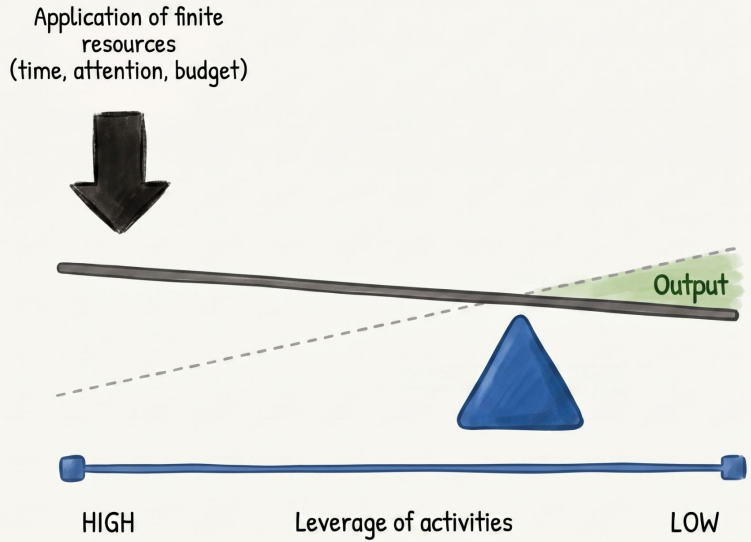
+

the output of the neighboring organizations under their influence

DEVSUMMIT <MOVE_ FORWARD/>

Scale your impact

Focus on high leverage activities



Additive model

Focus on *symptom solving, micromanagement, high volume of meetings, routine tasks*

Impact = your output

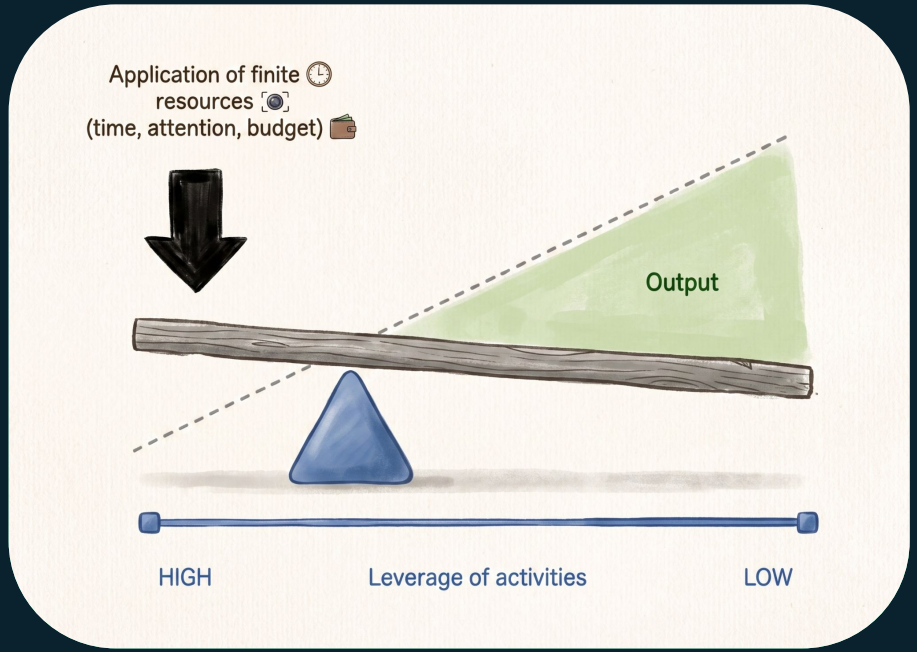
Scales linearly with hours worked.

Hard ceiling: ~1 person's bandwidth.

DEVSUMMIT <MOVE_ FORWARD/>

Scale your impact

Focus on high leverage activities



Multiplicative model

Focus on *training and onboarding, process definition, delegation, recruitment*

Impact = team output × your influence

Scales with the size of the system you build.

No ceiling: except your architecture skills.

DEVSUMMIT <MOVE_ FORWARD/>

Withdrawal Keyboard Syndrome

Worst of both world

BUG#1

Often, new managers try to be "Manager + Senior Dev" simultaneously. Research shows this is highly unstable and predictably fails.

Why it fails

- Engineering requires uninterrupted deep focus (flow state)
- Management is inherently interrupt-driven
- You can't optimise both simultaneously

You fragment your own cognitive bandwidth

The consequences

- Manager keeps the hardest, most interesting tickets
- Manager becomes SPOF, meetings fragment delivery
- Team is starved of growth challenges

Both management and engineering quality collapse, teammates leaves

DEVSUMMIT <MOVE_ FORWARD/>

How to become scalable

Stop being the doer: start being the multiplier

The Time Fallacy

"Nobody does it as well or as fast as me. It's more rational to just do it myself."

Short-term arithmetic is correct !
30 min yourself < 2 hours to train someone

➢ **But this thinking creates a Bus Factor of 1 - you stay on an additive model**

The Compounding View

Invest hours now → **save weeks of recurring work**

Refuse to delegate → **starve your team of growth opportunities**

The **70% heuristic**
if someone can do it 70% as well as you, **delegate it immediately**

Task-Relevant Maturity

source: Andrew Grove - High Output Management

Delegation as an algorithm

Delegate based on the person's maturity for the specific task.
Not their seniority level overall.

LOW	**Tell & structure**	**Provide specific "how"** instructions and follow up **Goal:** teach and build foundations. **Be directive**

Well adapted in 1:1 situation and operational tasks

DEVSUMMIT <MOVE_FORWARD/>

7 levels of delegation

source: Jurgen Appelo - Management 3.0

Go beyond TRM to deal with your team

1	**Tell**	Manager decides unilaterally, informs team. *E.g. P1 incident, legal mandate.*	**I decide**
2	**Sell**	Manager decides, explains reasoning, seeks buy-in. *E.g. strategic arch change.*	
3	**Consult**	Gathers input, retains final authority. *E.g. new system design with business risk.*	**WE decide**
4	**Agree**	Consensus decision as a peer. *E.g. team code standards, library choices.*	
5	**Advise**	Team decides; manager offers perspective only. *E.g. senior choosing test tooling.*	**YOU decide**
6	**Inquire**	Team decides and acts; informs manager after. *E.g. daily implementation choices.*	
7	**Delegate**	Full ownership; no oversight needed. *E.g. specialist domain, internal team rituals.*	

DEVSUMMIT <MOVE_ FORWARD/>

Define clear responsibilities, avoid frustrations

RACI matrix for the rescue

IDEA#1

New managers often overstep their new role to handle tasks that are no longer their responsibility, creating frustration within the team

Letter	Role	In Engineering Terms
Responsible	**Does the work**	The engineer(s) writing the code / implementing the feature
Accountable	**Owns the outcome**	The tech lead or EM who signs off on quality & delivery. One person only.
Consult	**Provides input**	Senior engineers, security, platform teams whose advice is needed
Inform	**Kept updated**	Product, stakeholders, other teams that need visibility but not decisions

DEVSUMMIT <MOVE_ FORWARD/>

The 1:1

Your Primary Management Tool

IDEA# 2

What you should aim

- Their meeting, not yours: **they set the agenda**
- Weekly, 30–45 min, **never cancelled (or at least move)**
- Career, blockers, wellbeing: not status updates

The place where trust is built or destroyed

Common failure modes

- Turning it into a project status meeting
- **Skipping** it when you're "too busy"
- Doing all the talking
- Never asking about career goals

A place where you’re micromanaging your teammate !

The signal: If your engineers only surface problems in all-hands or on Slack/GChat, your 1:1s aren't working.
Psychological safety is created one conversation at a time.

DEVSUMMIT <MOVE_ FORWARD/>

Micromanagement

The paradox of under-management

BUG#2

By trying to over-manage the how, you critically under-manage the what matters: vision, conflict resolution, alignment, and team growth.

The damage

- Destroys trust and autonomy
- Engineers stop thinking, just execute
- Innovation evaporates

Your best engineers leave first

The antidote

- Define constraints, not implementation.
- Never be the first to speak about solutions.
- Specify: architectural boundaries, quality criteria, business outcomes.
- Accept that the team will take a different path than you would have.

That's not failure: that's growth.

DEVSUMMIT <MOVE_ FORWARD/>

The manager's KPI

DORA to the rescue

🚀 Deployment Frequency
How often code reaches production

Low frequency → process bottlenecks, fear of deployment, large batches

💥 Change Failure Rate
% of deployments causing a production incident

Balances speed vs. quality

⏱️ Lead Time for Changes
First commit → production.

Reveals friction in the pipeline and review delays.

🔄 MTTR
Mean Time to Recovery

Measures resilience, observability quality & runbook completeness.

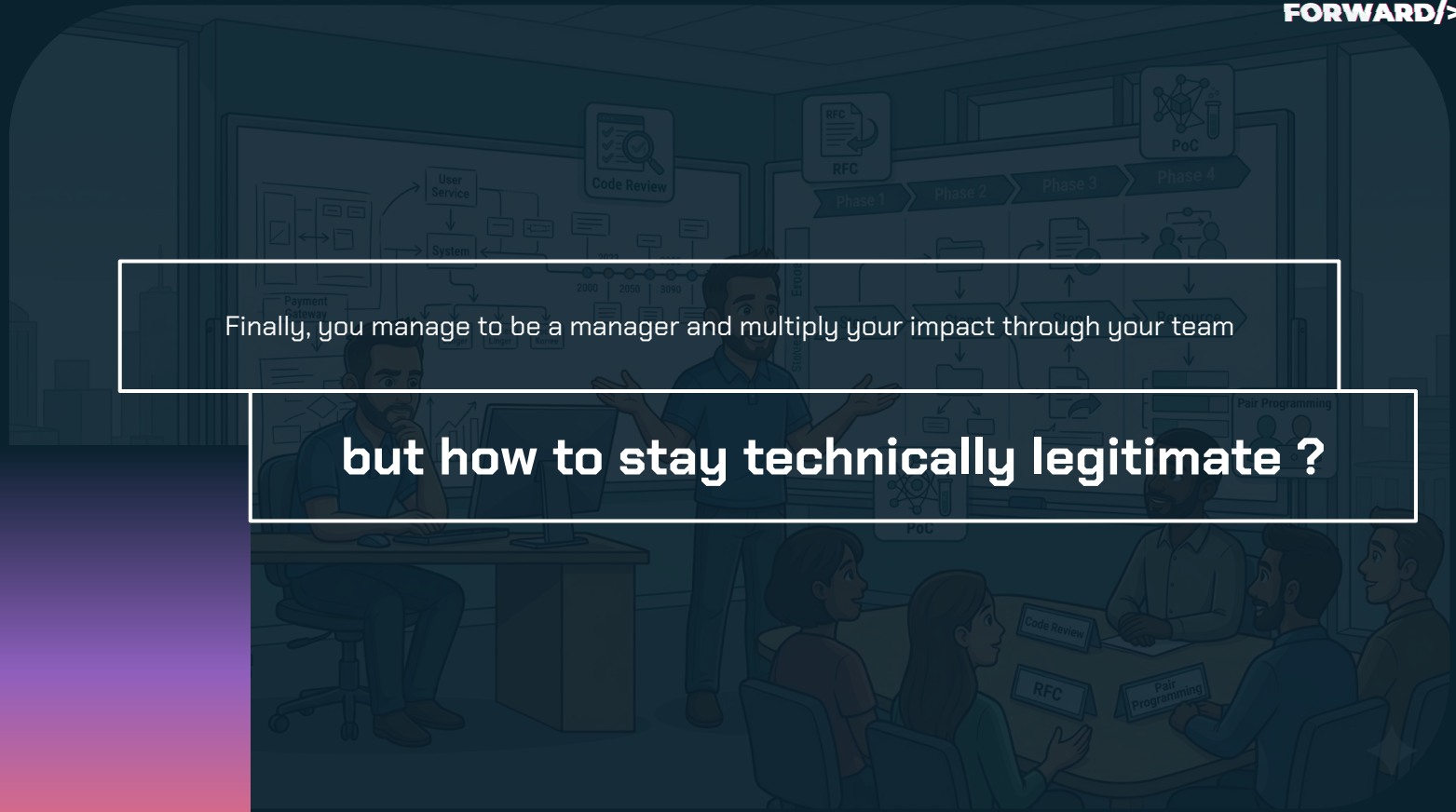
DEVSUMMIT <MOVE_ FORWARD/>

The manager's KPI

DORA as systemic diagnosis

Metric	Elite Teams	When it degrades => EM action
Deployment frequency	Multiple/days	Automated tests, pipelines, address team deployment anxiety
Lead time for changes	< 1 hour	Review PR size, time-to-first review, review silos
Change failure rate	0-5%	Invest in test coverage, enforce DoD, feature flags
MTTR	< 1 hour	Improve observability, write runbooks, enable rollbacks automation

⚠️ **Critical rule:** Use DORA for systemic diagnosis, never to rank or punish individuals. A bad metric is a process signal, not a performance signal.



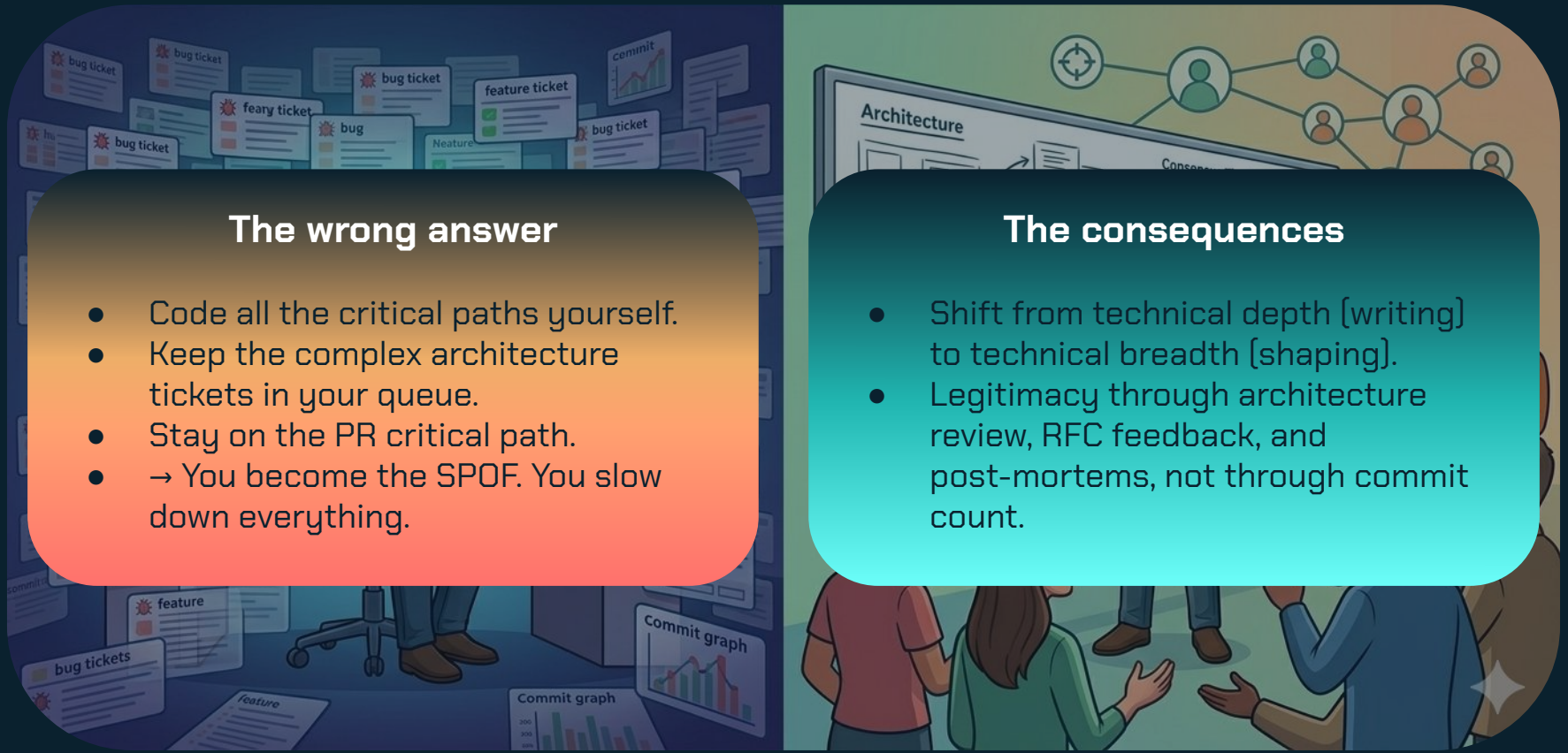
Finally, you manage to be a manager and multiply your impact through your team

but how to stay technically legitimate ?

DEVSUMMIT <MOVE_ FORWARD/>

The legitimate question

How to stay technically credible without being a bottleneck



The wrong answer

- Code all the critical paths yourself.
- Keep the complex architecture tickets in your queue.
- Stay on the PR critical path.
- → You become the SPOF. You slow down everything.

The consequences

- Shift from technical depth (writing) to technical breadth (shaping).
- Legitimacy through architecture review, RFC feedback, and post-mortems, not through commit count.

DEVSUMMIT <MOVE_ FORWARD/>

Staying Technical by Proxy

Lean on the team to stay in the loop

Target: ~20–30% of your time on technical engagement

1	**Structural Code Reviews**	Review for design and architecture, not syntax. Ask "does this fit the system's long-term direction?" never block a PR deployment with your approval.
2	**Architecture & Post-Mortems**	Lead RFC reviews, drive post-mortems. Ask the hard security, cost, and scalability questions. Translate to business language for leadership.
3	**Internal Tooling & PoCs**	If you need to code, write automation scripts or throw-away PoCs. Never on the critical path for customer delivery.
4	**Targeted Pair Programming**	Sit with a junior or new hire occasionally. Keeps you grounded in real workflows. Builds trust. Costs zero delivery time.

Remember ?

DEVSUMMIT <MOVE_ FORWARD/>

78%

of engineers report impostor syndrome at the leadership transition

DEVSUMMIT <MOVE_ FORWARD/>

Fight back Impostor Syndrome

ASPIRE method

Acknowledge & be aware	Admit the discomfort without judgment. "I feel out of my depth. That's okay."
Separate feelings from fact	Move your success metric: from code volume → team autonomy and health.
Progress over progression	Identify the specific area triggering doubt. Is it people skills? Strategic thinking? Name it precisely.
Identify achievements	Find evidence that contradicts the impostor narrative. You've solved hard human problems before.
Reframe from innerscript	You were promoted for strategy, people development, and systemic thinking. Not to keep coding.
Execute	Act despite the doubt. Build your peer network. Show vulnerability with your team. It builds trust, not weakness.

DEVSUMMIT <MOVE_FORWARD/>

Vulnerability as a Leadership Superpower

Admitting you don't know everything

Admitting you don't know the answer doesn't destroy your authority.
It destroys the toxic hierarchy that prevents your team from being honest with you.

What vulnerability enables

- Team members surface problems earlier
- Engineers admit mistakes without fear
- Blameless post-mortem culture becomes real
- Psychological safety → innovation

How to ?

- Say "I don't know, let's find out together" in team meetings
- Admit a management mistake publicly in a retro
- Ask your team what you should do differently
- Share your own impostor moments with new managers you mentor

Main takeaways

4 things to take home

DEVSUMMIT <MOVE_ FORWARD/>

Rebuild your Definition of Done

Your output is your team's output. Measure the system, not your commits.

Learn to delegate (TRM + 7 levels)

Calibrate to the task, not the title. Be explicit about the level you're granting.

Stay technical by proxy

Code reviews for architecture, RFCs, post-mortems, occasional pair programming. Stay off the critical path.

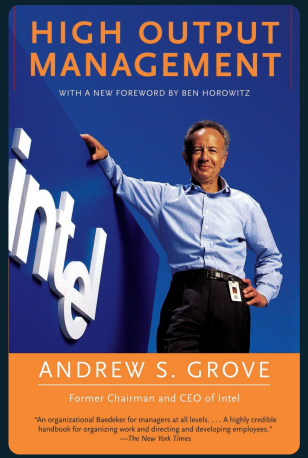
Name your bugs

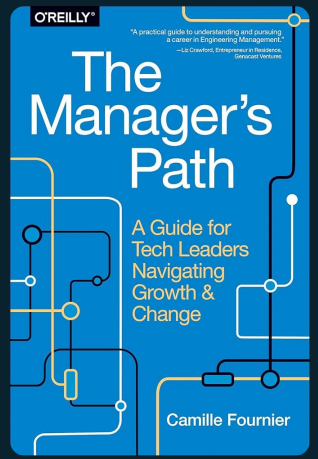
Keyboard withdrawal, micromanagement, impostor syndrome. Naming them is the first step to not being controlled by them.

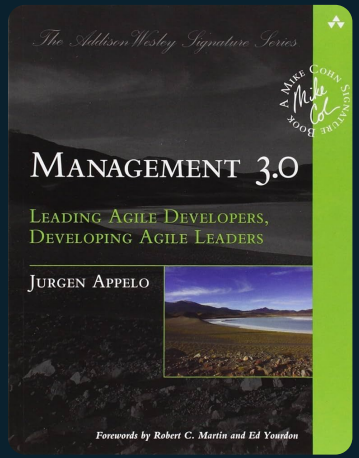
DEVSUMMIT <MOVE_ FORWARD/>

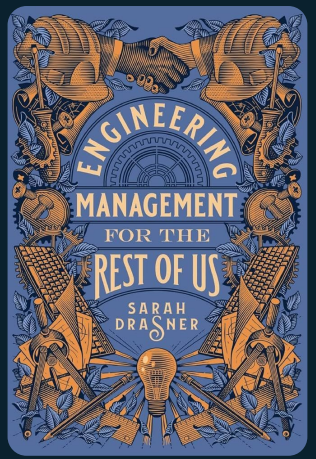
Want to go deeper ?

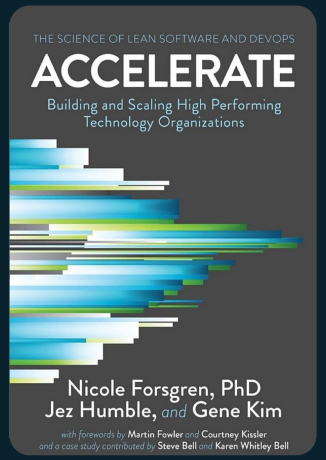
Some recommendations











The Final Word

DEVSUMMIT <MOVE_ FORWARD/>

It is only by accepting that you are no longer the best engineer in the room that you can hope to become the architect of those who will build the future.

Your action this week

- Identify one task you should delegate but haven't
- Determine its TRM level and/or the matching delegation level
- Delegate it. **For real**.

Your action this month

- Pull your team's DORA metrics
- Identify the one metric that would unlock the most value
- Make it the focus of your next team retrospective

Thanks.