The High-Performance Team Playbook

A real-world guide to building performance, embedding AI and reducing risk in software development teams.

Authored by

Simon Azzopardi
Partner

Eman Zerafa
CTO

CLEVERBIT SOFTWARE

Introduction:

The **Hard Truth** About Building Teams

Simon Azzopardi
Partner

If you've been in engineering leadership long enough, you carry scars. I know I do.

I'm Simon and I've led teams in startups where the pressure was relentless. We had to move fast, deal with technologically complex problems, and ship products nobody thought were even possible. The bar for performance wasn't just high. It was brutal: Deliver late, and you lose funding. Deliver poorly, and you lose customers. Deliver without vision, and you lose your team.

That experience forced me to learn quickly. Not just how to deliver value at speed, but how to test, break, and rebuild what makes both small and large teams perform. I've seen brilliant groups implode because they lacked alignment. I've seen average teams suddenly click when ownership and clarity were in place. And I've seen how culture, transparency, and retention can make or break engineering performance more than technology ever will.

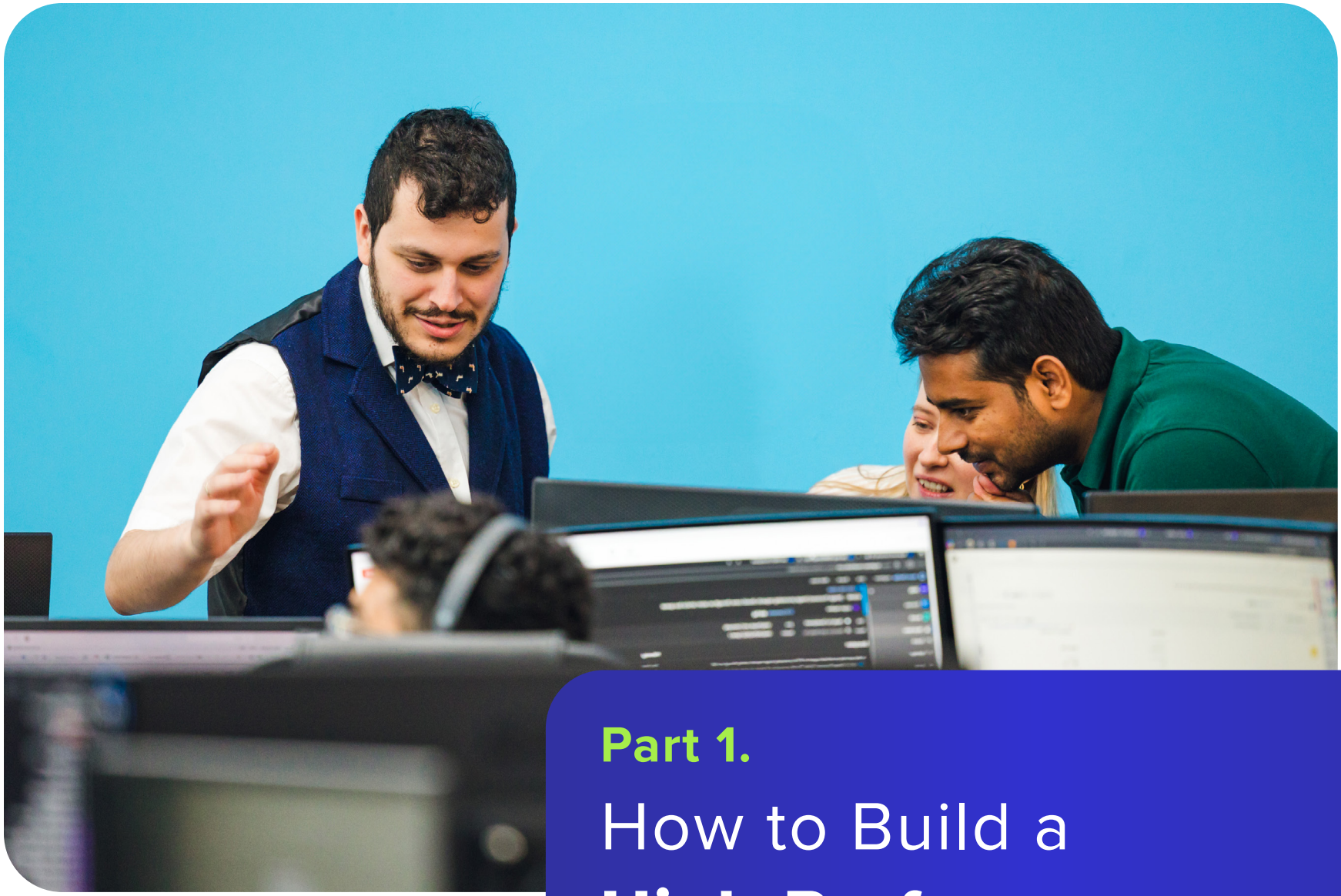
The truth is this: most engineering teams are average at best, fragile at worst. And it's not because leaders don't care. It's because building high-performance teams is harder than most people admit. It's not just about hiring smart engineers and giving them Jira boards. It's about building a system: ownership, retention, trust, continuity, and now AI maturity. Miss even one, and performance starts to crumble.

This paper is the result of years of trial, error, and hard lessons. It's written for CTOs and engineering managers under the same pressures I've lived through: where the stakes are high, the timelines short, and the demands unrelenting.

Along the way, I'll also bring in our CTO, Eman Zerafa. Eman has deep tech leadership experience and is today breaking new ground on the topic of how AI is reshaping software delivery. His perspective comes straight from experiments with real teams and systems. With the way software development is changing, his views and research are critical. Where my scars come from leading under pressure, his come from testing how new tools and approaches actually work in practice.

My aim is simple: if you're going at it alone, here's the playbook I wish I'd had when I was in your shoes.

Table of **Contents**



Part 1.

How to Build a **High-Performance** Software Engineering Team

1.1 Foundations of High-Performance Teams

When I look back at the teams I've led, the difference between those that thrived and those that broke down came down to five foundations. None of these are theoretical. They're scars, things I've learned by getting them wrong before I got them right. And the research backs it up.

Bridging from the Classics

When people hear about "five foundations" of teams, many think of Patrick Lencioni's *Five Dysfunctions of a Team*. It remains one of the best frameworks for understanding the relational side of teamwork: trust, conflict, commitment, accountability, and results.

What we share here doesn't replace that, it builds on it. If Lencioni described the **interpersonal dynamics**, these are the **operational dynamics** we've seen in software engineering: the structures, cultural signals, and leadership choices that consistently separate fragile teams from high-performance ones.

The research echoes these lessons:

- **Google's Project Aristotle** found psychological safety was the strongest predictor of team success, but also showed that cultural alignment and humility mattered more than pure technical skill.
- **DORA's State of DevOps** reports showed elite teams deliver more frequently, recover faster, and avoid burnout, thanks to lightweight governance, clear ownership, and a focus on outcomes over activity.

Our five foundations come from both scars and evidence. We've seen teams falter when one was missing, and thrive when all were present.

Foundation 1: Ownership Over Delivery

Performance collapses without ownership. Teams that only deliver tasks may look efficient in the short term, but when priorities shift or pressure rises, they stall.

High-performance teams behave differently. They act like partners, not order-takers. They connect their work to business outcomes, raise issues early, and balance pragmatism with quality.

Ownership is the thread that ties it all together, without it, the other foundations don't stick. That's why ownership has to run through everything: how people are recruited, how tasks are written, how teams are structured. It's why work is framed with the "why" as much as the "what." It's why mentorship and context are part of onboarding. And it's why simple principles, think an **ownership manifesto**, exist to guide decisions when pressure is high.

As we often remind teams:
"If you were the one paying for this, would you still make this decision?"

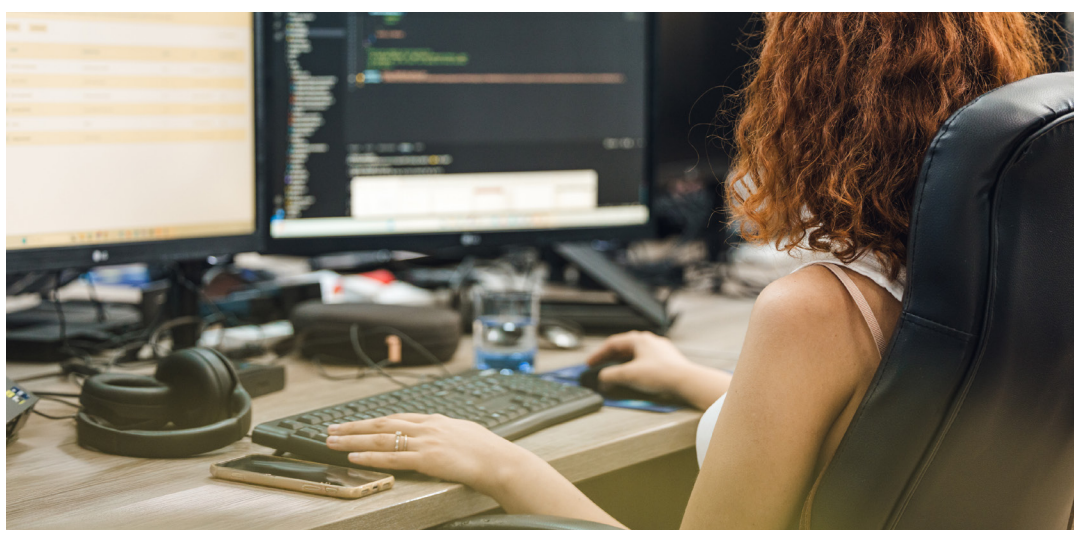
That question shifts people from executors to partners, from order-takers to owners.

Average Teams:
Close tickets, but remain disconnected from outcomes.

High-Performance Teams:
Take accountability, connect work to value, and adapt as if success or failure were their own.

*According to McKinsey, teams with strong alignment to business purpose are **2.4x more likely** to hit performance targets than those without.*

Lesson Learned:
Don't starve developers of context. If you treat them like code monkeys, they'll act like it. If you give them purpose, they'll act like owners.



Foundation 2: Cultural Alignment (Anti-Diva, Pro-Business)

The strongest developers I've worked with weren't just great coders. They could talk to the business. They could say "no" when something didn't make sense, and they had the humility to admit when they didn't know the answer.

The weak link is the "diva developer": technically brilliant, but allergic to feedback, dismissive of the business, or convinced they always know best. I've seen one diva destroy a team faster than a bad manager.

Average Teams:
Stay in their technical bubble, doing what they're told, or letting strong personalities dominate.

High-Performance Teams:
Integrate into the wider business. They challenge constructively, they listen, and they know they don't have all the answers.

Research by Google's Project Aristotle found that the strongest predictor of team success wasn't raw IQ or technical skills, but ***conversational turn-taking and social sensitivity****: in other words, cultural alignment and humility.*

Lesson Learned:
Culture is not about "being nice" or "feeling like a family." It's about building a team of adults who can talk across the business and check their egos at the door.

Foundation 3: Psychological Safety with Accountability (Meritocracy, Not Family Spirit)

I've worked in environments where everyone said "we're a family."

It sounds nice, but it usually means avoiding hard conversations and tolerating mediocrity.

That kills performance.

Average Teams:
Either stay silent out of fear or hide behind "family spirit" where nothing is challenged.

High-Performance Teams:
Speak up, take ownership, and know performance is what earns respect.

In Google's study of 180+ teams, ***psychological safety*** *was the #1 factor for performance. But without accountability, it slips into complacency. Gallup data shows only* ***21% of employees strongly agree their performance is managed in a way that motivates them to do outstanding work.***

Lesson Learned:
Safety without ownership is chaos. Ownership without safety is silence. You need both.

Foundation 4: Clarity of Roles and Autonomy (With Room to Fail)

I once watched a release stall for three weeks because nobody knew who was supposed to make a decision. Leadership kept getting dragged in, morale dipped, and progress stopped.

High-performance teams thrive on clarity. Roles are clear, decisions are delegated, and autonomy is real. People don't need a manager in every meeting to move forward.

And yes, autonomy means mistakes will happen. That's where learning comes from.

Average Teams:
Depend on managers for every call, paralysed by fear of screwing up.

High-Performance Teams:
Know who owns what, act with autonomy, and learn from failure.

*Harvard Business Review found that teams with clear roles and decision-making structures are **53% more effective** at execution than teams without them.*

Lesson Learned:
Autonomy means people will sometimes mess up. That's the price of building leaders, not followers.

Foundation 5: Retention as a Performance Strategy (But for the Right Reasons)

I used to think retention was always good. If people were staying, I assumed we had stability. I've since learned that retention can also mean stagnation. I've seen teams where nobody left, not because they were thriving but because they were comfortable, coasting or stuck.

Performance was flat, innovation was dead, but on paper retention looked great. The high-performance teams I've worked with had high quality retention. The right people stayed because they were challenged and growing. The wrong people moved on.

Average Teams:
Retain by default, even as performance stagnates.

High-Performance Teams:
Retain because the environment is challenging, fair, and rewarding.

*A recent LinkedIn survey showed that **94% of employees would stay longer** if companies invested in their growth. Retention without growth, however, leads to stagnation.*

Lesson Learned:
Retention matters, but only if it's driven by growth and performance, not comfort and complacency.

Bringing It Together

When you see these five foundations in place, the difference is obvious. Developers don't just code; they argue priorities, speak to the business, own mistakes, and stay because they're growing.

Miss any of these, and you feel the cracks fast. I've seen teams fall apart because they had smart people but no context, or because they clung to "family spirit" while mediocrity spread. High performance isn't about hype. It's about design, discipline, and outcomes.

Foundations of
High-Performance Teams

1.2 Talent: Hiring, Retention & Development

You can't build a high-performance team without the right people.

But I've learned the hard way: it's not just about hiring "the best."

It's about hiring for fit, keeping them for the right reasons, and constantly pushing for growth.

Strategy 1. Hiring for Fit, Not Just Skills

I once hired a technically brilliant engineer to lead a team of very senior people. He truly was a genius. He was also almost always right. The problem was he loved being right. He shoved it down everyone's throat. Nobody else had a voice. Instead of lifting the team, he drained its energy.

When we finally let him go, the difference was immediate. The team, still highly competent, found its voice. They debated, they disagreed, they owned outcomes together. And they moved faster.

Average Teams:
Hire "rockstars" who dominate the room but suffocate collaboration.

High-Performance Teams:
Hire problem-solvers who enable others, not just themselves.

*Google found that technical skills accounted for less than **20% of long-term success** in engineering hires; adaptability and collaboration mattered far more.*

Lesson Learned:
Don't just hire brilliance. Hire brilliance that leaves room for others to shine.

Strategy 2. Retention Through Growth (Not Comfort)

Let's face it: you build relationships with people. I once had someone on a team responsible for solving time-sensitive issues. She was struggling. There were very real personal challenges, and I wanted to support her. We all picked up the slack, made excuses, gave more time.

But things got worse, not better. I dragged my feet for months. In hindsight, I should have acted in days or weeks. The truth is that the whole team is more important than any individual, no matter how much empathy you feel.

Average Teams:
Keep people out of loyalty or comfort, even when performance is gone.

High-Performance Teams:
Retain because people are growing, not because it feels easier to avoid tough calls.

*LinkedIn data shows **94% of employees would stay longer** if companies invested in their growth and development.*

Lesson Learned:
Retention should mean progress. Keeping someone too long out of loyalty isn't kindness; it risks the whole team.

Strategy 3. Knowledge Transfer & Continuity

One thing I've always been obsessive about is knowledge transfer. I never wanted a single engineer, no matter how good, to be indispensable.

I made knowledge socialisation a core practice. Not just documentation, but deliberate opportunities for people to share what they knew, through demos, shadowing, rotating ownership. Because the truth is, people do leave. Pretending otherwise is reckless.

Average Teams:
Depend on "hero engineers" and hope they never leave.

High-Performance Teams:
Build systems where knowledge is shared, and no resignation can derail the roadmap.

*Forrester reports that **53% of IT leaders see knowledge loss due to turnover as a top threat to performance.***

Lesson Learned:
Leaders aren't indispensable. The moment you believe they are, you've built fragility into your team.

Strategy 4. Development & Mentorship

I am the best example of why mentorship matters. My career accelerated because I had great mentors. The quality of my mentors directly influenced the speed of my growth.

That lesson stuck with me.

I want people I lead to feel that same lift: that someone cares about their potential, pushes them, and shows them the next step. And I want them to pass that same care on to others.

Average Teams:
Leave growth to chance.

High-Performance Teams:
Build mentorship into the culture, so learning compounds across the team.

*Deloitte found that organisations with formal mentorship programs see **higher engagement (67%) and retention** compared to those without.*

Lesson Learned:
A team that isn't learning is a team that's falling behind. Mentorship isn't a "nice to have"; it's a performance engine.

Bringing It Together

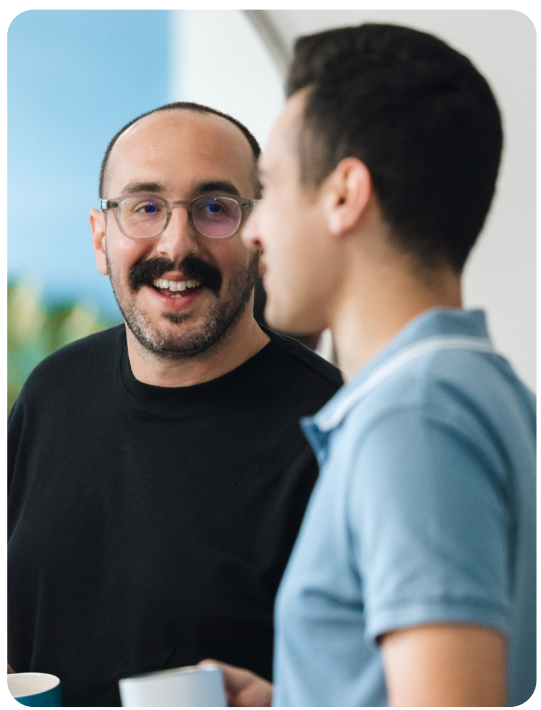
Hiring great people isn't enough. If they don't fit, if they don't grow, or if knowledge isn't shared, performance will eventually break.

But if you get hiring, retention, continuity, and mentorship working together, you create compounding performance.

Talent: Strategies for Hiring, Retention & Development

1.3 Structures That Enable Performance

Talent alone won't make a team high-performing. You need structures that channel that talent into consistent, reliable output while preserving autonomy.



I've seen brilliant people underperform in the wrong structure, and average teams thrive in the right one.

Consideration 1: Team Composition Matters

I was once part of an early-stage business that had plenty of "C-level power," senior strategists, architects, and executives, but very little execution strength. Most of the team in the engine room were juniors, and the disconnect was comical. Strategy after strategy came out of the boardroom, but very little actually moved forward.

The problem? Leaders often want to hire people like themselves. Senior people hire seniors, because it feels comfortable. But a team full of strategists without enough doers is a recipe for slow progress.

Average Teams:
Over-index on one level of seniority, with either too many juniors (fast but brittle) or too many seniors (smart but slow).

High-Performance Teams:
Strike a balance. Seniors set direction and mentor, mids carry the load, and juniors inject energy and learn.

*Microsoft research found that teams with a balanced seniority mix were **30% more productive** than homogeneous teams.*

Lesson Learned:
Don't build teams for comfort. Build them for execution.

Consideration 2: Leadership as an Enabler (Not a Commander)

I was once thrown into a company built entirely around a command-style leader. When that leader left, the organisation fell into utter confusion. Nobody knew how to act, because they were used to being told what to do. The dependency was the risk.

When we shifted leadership style, empowering managers to set context and teams to make decisions, the effect was dramatic. We gained speed, resilience, and during the pandemic (the hardest operating environment I've ever seen), we hit record growth.

Average Teams:
Rely on a commander to make every call.

High-Performance Teams:
Have leaders who enable, not dictate, removing blockers, setting vision, and creating space for people to act.

*Gallup shows that teams with "coaching-style" leaders are **12% more productive and 30% more engaged** than those with directive managers.*

Lesson Learned:
A "commander" isn't just risky; it's a single point of failure. Leadership should enable resilience, not dependency.

Consideration 3: Governance Without Bureaucracy

I've seen governance spin out of control. We create rules to manage exceptions, to prevent the mistakes of weak teams. But those same rules end up suffocating good teams.

Processes multiply until high performers are slowed down by bureaucracy designed for mediocrity.

Average Teams:
Drown in governance built for the lowest common denominator.

High-Performance Teams:
Use lightweight governance, including automation, dashboards, and clear reviews, as a safety net, not a straitjacket.

*The DORA report shows that elite-performing teams spend **50% less time** on unplanned work thanks to lightweight, automated governance.*

Lesson Learned:
Build governance for performance, not mediocrity.

Consideration 4: Exit & Continuity Planning

I once managed 60 engineers outsourced from a supplier. They were excellent; talented, motivated, delivering real value. Then the supplier was sold. New management came in, prices spiked, and culture shifted overnight. What had been a high-performing setup turned into a nightmare. Exiting was painful, expensive, and filled with uncertainty, right in the middle of the busiest season of the business.

That experience left a scar. It taught me that continuity and exit planning aren't optional extras. They're survival.

Average Teams:
Hope key people and suppliers never change.

High-Performance Teams:
Bake continuity in, with knowledge redundancy, succession planning, and contracts designed to preserve ownership and flexibility.

*PwC found that **39% of CEOs are "extremely concerned" about talent continuity risk**, yet most companies don't build structures to mitigate it.*

Lesson Learned:
Continuity is the invisible foundation of performance. Ignore it, and you're gambling with your future.

Structures that **Enable Performance**

Bringing It **Together**

The right structure doesn't feel like bureaucracy. It feels like freedom. Teams have clarity, balance, and resilience. They move fast without being reckless, and they can survive change without falling apart.

And here's the ultimate truth I've learned: **bad performance is never just about the people. It's about the context too.** The wrong structure can make good people look bad. The right structure can unlock hidden potential. If you want high performance, you need both: quality people and an environment designed for them to thrive.

1.4 Building Trust & Transparency

The strongest teams I've worked with weren't just technically capable; they were radically open with each other and with the business.

The weakest ones were opaque, defensive, or siloed.

If there's one thing I've learned, it's that trust doesn't appear by accident. It has to be engineered.

Step 1: Hyper-Transparency: Beyond Dashboards

In the early days, our teams used to give periodic updates to the wider company. They were dry, technical presentations; informative, but forgettable.

I started coaching lower-level engineers to present instead, and to inject some of their personality into it. Suddenly, these updates weren't just about code, they were about people.

The impact was surprising: one Chief Strategy Officer found common ground with a UX designer over their shared love of Japanese culture. That connection wouldn't have happened without engineers being visible as humans, not just coders.

And once those human bonds formed, conversations about priorities and trade-offs flowed far more naturally.

That's the essence of hyper-transparency: making information visible, and equally, making the people behind it visible.

Transparency should build trust not just in the work, but in the humans delivering it.

Step 2: Adapting Communication for Stakeholders

However transparency isn't just about volume. I've seen engineers get frustrated when a client asks a question and they respond with, "Yes, but I already told you this."

It's a classic trap. Information may have been shared, but not in a way that stuck with the audience. Transparency isn't about dumping data; it's about making sure stakeholders can actually understand and use it.

Average Teams:
Share information passively and expect the business to adapt.

High-Performance Teams:
Adapt their communication so stakeholders can absorb it, act on it, and trust it.

*PMI found that **56% of project failures** cite poor communication as the main cause, usually not because information wasn't shared, but because it wasn't shared in the right way.*

Lesson Learned:
Transparency without empathy is noise. True transparency means making information visible and consumable

Building **Trust & Transparency**

Bringing It Together

Trust is built on visibility, but also on accessibility and humanity. Dashboards, metrics, and repos are essential. But so is giving engineers a voice, letting them connect as people, and adapting communication so stakeholders actually trust what they hear.

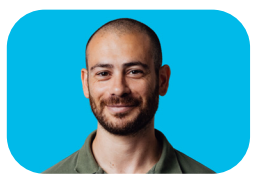
When transparency is engineered this way, trust becomes natural, and with trust comes speed.

Part 2.

Leveraging AI in High-Performance Teams

2.1 Leveraging AI in High-Performance Teams

*Up to this point, I've shared what I've learned about building high-performance teams: ownership, culture, safety, clarity, retention, structure, trust, transparency. But there's one area where I'm not the expert, and where the landscape is shifting faster than ever: AI. At Cleverbit, our **CTO - Eman Zerafa,** has been leading our research into how AI is changing software engineering, not in theory, but in daily practice across live projects. I've asked him to share his perspective here, because the reality is that high-performance teams today can't ignore AI. It's already transforming the way we build, and it brings both opportunities and risks.*



Eman Zerafa

CTO, Engineering & AI Leadership

AI is no longer a question of "if." It's already in the room. Most developers today have GitHub Copilot, ChatGPT, or some other assistant running in the corner of their IDE. The real question is: **are you using it in a way that compounds performance, or in a way that creates fragility?**

Over the last two years, my focus has been on testing AI-first development models.

What we've found is simple: AI can be a force multiplier, but only when used with maturity and guardrails. Without that, it leads to what we call vibe code drift: plausible-looking code that quietly rots your codebase.

Lever 1: AI as a Force Multiplier

Most teams use AI haphazardly. AI isn't a silver bullet; it's a system component. Yet, most teams use it like a shiny new gadget. Developers try whatever tool lands on their feed, with no defined workflow, no quality standards, and no shared understanding of when to trust or verify outputs. The result? Inconsistent code, duplicated effort, and a false sense of speed.

High-performance teams don't just use AI. They design for it. They define where it fits in their development cycle: scaffolding, testing, review, documentation. They set guardrails for what's acceptable and measurable outcomes for what "AI-assisted" really means.

Average Teams:
Treat AI as a fancy autocomplete.

High-Performance Teams:
Use AI as part of structured workflows - scaffolding, testing, edge case discovery, documentation.

McKinsey estimates AI in software development can ***increase productivity by 20–45%,*** *depending on maturity of adoption.*

Lesson Learned:
Treat AI as part of your delivery system, not a toy in your IDE.

Lever 2: The Risk of Vibe Code Drift

The biggest risk with AI in development isn't that it writes bad code; it's that it writes convincing code. Clean, idiomatic, and seemingly bulletproof, yet still wrong. That's vibe code drift: when developers start to trust the output because it feels right. Over time, reviews become cursory, assumptions harden into "truths," and subtle flaws accumulate beneath a surface of polished syntax.

We've seen pull requests that looked flawless. Until someone traced the logic and realized the code was doing something no one actually intended. That's the danger: trust built on vibe, not verification.

Average Teams:
Trust AI output at face value, fix it only when it fails.

High-Performance Teams:
Treat AI-generated code as disposable. If it's wrong, regenerate. Don't patch.

A GitHub study found that ***36% of developers using AI-assisted coding admitted to shipping code they didn't fully understand.***

Lesson Learned:
AI isn't dangerous because it's bad. It's dangerous because it's convincing.

Lever 3: Regenerate, Don't Patch

When AI-generated code fails, the reflex is to fix it: adjust a few lines, rerun the test, and move on. But that mindset turns AI into a productivity sink. You end up chasing coherence across outputs that were never meant to fit together.

The better approach is to reset. Start from a clean slate with a clearer objective and simpler context. AI performs best when the problem space is well-defined, not when it's buried under layers of patched prompts and partial fixes.

Average Teams:
Keep tweaking AI output until it works.

High-Performance Teams:
Know when to stop fixing and start over. Keep the context clean, and the workflow simple.

Lesson Learned:
AI is probabilistic. Don't fix symptoms - fix the process.

Lever 4: Humans Where It Counts

AI is designed to please and will confidently build whatever you ask for. It can analyse regulations, propose trade-offs, and generate solutions at scale. What it can't do is decide which problem is worth solving, or take responsibility when choices have real impact.

The strongest results come from teams that use AI for what it's best at: speed, breadth, and synthesis, while keeping humans in charge of intent, ethics, and accountability.

AI explores possibilities, generates options, and tests assumptions. **Humans** define direction, decide what matters, and sign off on what ships.

Guardrails ensure the system reflects human intent, not just statistical confidence.

*Harvard Business Review reports that **78% of executives believe human judgment will remain critical in AI-augmented work.***

Lesson Learned:
AI can accelerate delivery. Only humans can ensure it's the right delivery.

Lever 5: Guardrails, not handcuffs

The goal isn't to replace human judgment; it's to scale it. Most teams swing between two extremes: free-for-all AI usage or complete lockdown. Both fail.

Thinking AI-first means designing the system so teams know **when** and **how** to use AI effectively. Guardrails protect against chaos, but clarity unlocks speed.

Average Teams:
Either let AI run wild or ban it outright.

High-Performance Teams:
Treat AI like any other engineering capability: governed, measured, and applied with intent.

MIT Sloan found that companies deploying AI with strong governance saw 2.6x higher ROI than those without.

Lesson Learned:
Don't fight AI. Frame it. Guardrails turn risk into value.

Encourage Exploration:

Some stages such as early prototypes, UI sketches and idea validation benefit from fast, loose "vibe coding."

Enforce Rigor:

In critical paths such as security, finance, and core logic, AI use should be constrained and reviewed.

Define Rules:

Set standards for prompt hygiene, review expectations, and what "AI-assisted" means in your org.

Automate Safety:

Use CI checks, static analysis, and vulnerability scans as default guardrails, not afterthoughts.

Bringing It Together

This is why I believe high-performance teams today need not just great people and strong structures, but also mature AI practices.

Anything less is leaving performance on the table or worse, inviting fragility.

Leveraging AI
in High-Performance Teams

2.2 Measuring Performance: **Metrics That Matter**

Measuring performance in engineering is brutally hard. It's not just about tracking output. It's about understanding the **value we're actually delivering**, why the team is engaged in this mission, and boiling it down to the few levers we can truly control.

I've sat in rooms with beautiful charts: burn-downs, velocity, hours logged and still had to answer the question from the board: "But what did this actually achieve?" That's the reality. We can make ourselves look busy. We can even make ourselves look fast. But unless the metrics connect back to outcomes the business cares about, we're lying to ourselves.

Average teams measure what's convenient.

High-performance teams measure what matters.

Tip 1: Outcome Over Activity

I once led a team that was closing tickets at speed. Our dashboards looked green, velocity was climbing.

But sales was stalling because customers weren't adopting. We were winning sprints and losing the game.

Average Teams:
Measure busyness: story points, tickets closed, hours worked.

High-Performance Teams:
Measure business outcomes: features used, customer value delivered, cost savings

*McKinsey found that companies that shift from activity metrics to outcome metrics are **1.9x more likely** to outperform peers on profitability.*

Lesson Learned:
Activity creates the illusion of progress. Outcomes prove it.

Tip 2: Leading vs Lagging Metrics

Another trap is obsessing over lagging indicators: revenue, adoption, uptime. By the time those numbers move, the damage is already done.

The best teams I've seen use leading indicators: early signals like user feedback in beta, time-to-merge PRs, or test coverage trends. These give warning before the bottom-line suffers.

Average Teams:
Wait for lagging KPIs to show a problem.

High-Performance Teams:
Watch leading indicators to course-correct early.

*DORA's State of DevOps report shows elite teams that use leading metrics deliver **46x more deployments** than low performers.*

Lesson Learned:
Don't wait for the quarterly report to tell you performance slipped. Spot it in the daily signals.

Tip 3: Caring About the Long Term

I used to believe engagement surveys were a measure of performance. They're not. I've had teams that looked engaged on paper, happy, stable, but output was flat.

What really matters is whether the team cares. Do they care about the long-term outcome, or are they just clocking in? Do they feel ownership over the mission, or are they just running sprints?

And here's the part leaders often miss: caring doesn't happen by chance. It's our responsibility as leaders to create the conditions where people connect with the bigger picture and feel their work matters.

Some businesses chase high performance at all costs. They burn teams out, replace them, and keep moving. If that's what the business needs, fine. I might not agree with it, but I won't say it's wrong in principle.

But if you want teams to deliver high performance over time, you need them to care. Without that, performance is always temporary.

Average Teams:
Focus on the short term, delivering fast but disconnected from outcomes.

High-Performance Teams:
Care about the long term: the product, the customer, the mission, because leadership gave them a reason to.

*MIT Sloan research shows that employees who believe their work has purpose are **2.6x more likely** to remain at a company and **3x more engaged** in delivery.*

Lesson Learned:
High performance isn't just about speed. It's about whether people care enough to sustain it. And that starts with leadership.

Tip 4: AI Productivity: Uplift vs Noise

This is where I hand over briefly to Eman. AI has made metrics even trickier. It's easy to assume productivity is up just because code is being generated faster. But the question is: are we getting ***true uplift,*** *or just* ***faster noise?***

Eman's View (CTO):

We measure AI impact not by how much code is written, but by the ratio of AI-generated value that survives into production without human patching.

If a model produces 200 lines of code and 150 get regenerated or rewritten, that's not uplift; that's churn. True uplift shows up when:

- AI-generated code passes tests without human correction.
- Developers spend more time on ambiguity and less on mechanics.
- Release cycles shorten without quality dropping.

Average Teams:
Assume AI improves productivity because more lines of code are written.

High-Performance Teams:
Measure uplift against baselines: how much faster value reaches production, at equal or higher quality.

In our experiments, AI uplift ranged from ***-10% (slower, more patching) to +45% (faster, higher quality)*** *depending on whether guardrails and regeneration practices were in place.*

Lesson Learned:
Don't assume AI is helping. Prove it.

Measuring Performance: **Metrics that Matter**

Bringing It Together

Performance isn't about looking busy, feeling happy, or reporting pretty charts. It's about whether the team is delivering outcomes the business cares about, whether they can sustain that performance over time, and whether AI is genuinely compounding value instead of creating noise.

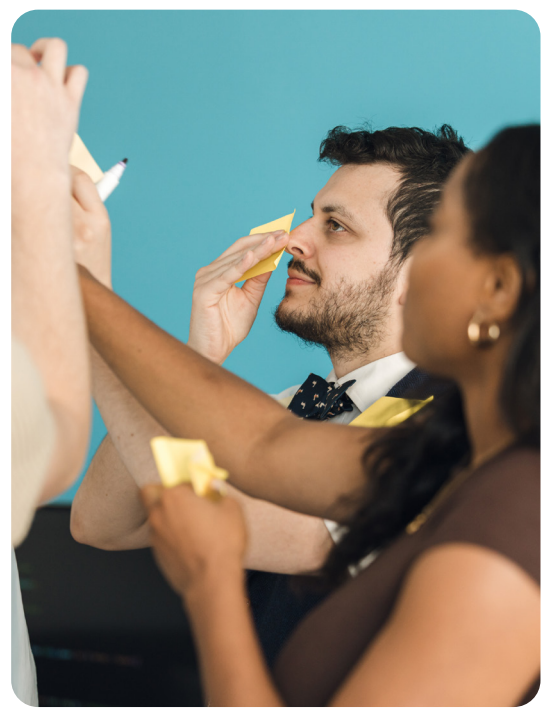
The only metrics that matter are the ones that connect back to value, and the ones the team can actually influence. Everything else is vanity.

2.3 Common Pitfalls & How to Avoid Them

Every time I've seen a team fail, it hasn't been because people weren't smart or hard-working. It's because of the same patterns repeating.

Here are the pitfalls I've run into, and what separates average teams from high-performance ones.

When these pitfalls hit, average teams get busier and more brittle: confusing activity with progress, and hoping vendors or AI won't blow up on them.



High-performance teams treat the same signals as prompts to reset: they re-anchor on outcomes, back enablers over divas, keep governance light, retain for growth, design for continuity and use AI with guardrails so issues become course-corrections, not crises.

Pitfall 1: Mistaking Activity for Progress

I've seen teams sprint through tickets, log long hours, and proudly present "green dashboards." But when we asked "what business problem did we solve?" the room went quiet.

Pitfall:
Confusing busyness with value.

Consequence:
Leaders get blindsided; teams look productive until customers or investors ask the hard questions.

Avoid it:
Measure outcomes, not just activity. Tie performance to business impact.

*McKinsey: companies that shift to outcome metrics are **1.9x more likely** to outperform peers on profitability..*

Pitfall 2: Hiring the Wrong "Rockstars"

I once hired a technically brilliant engineer who was almost always right. The problem? He loved being right. He drained energy from everyone else, and the team froze. When he left, the team found its voice and sped up.

Pitfall:
Hiring brilliance that silences others.

Consequence:
Teams lose ownership, collaboration, and speed.

Avoid it:
Hire problem-solvers who enable others, not divas who dominate.

Google research: collaboration and adaptability predict long-term success more than raw technical skill.

Pitfall 3: Over-Indexing on Leadership Style

I once worked in a company built around a "commander." When he left, the organisation fell into chaos.

Teams didn't know how to move without orders. That dependency was the risk.

Pitfall:
Building teams dependent on command-and-control.

Consequence:
Fragility occurs when the leader goes and performance collapses.

Avoid it:
Build enabling leadership.
Leaders set clarity and guardrails, but teams own execution.

*Gallup: teams with coaching-style managers are **30% more engaged** than those with directive managers.*

Pitfall 4: Governance That Strangles

I've seen rules multiply until high-performing teams were slowed to a crawl.

Governance designed to protect against weak teams ended up suffocating the good ones.

Pitfall:
Creating governance for mediocrity.

Consequence:
High performers disengage, velocity plummets.

Avoid it:
Keep governance lightweight. Automate checks, trust teams, and use rules as safety nets, not handcuffs.

*DORA report: elite teams spend **50% less time** on unplanned work thanks to lightweight governance.*

Pitfall 5: Retention for the Wrong Reasons

I once kept someone on a critical team far too long. I empathised with their personal struggles, but performance was falling. By the time I acted, the whole team was suffering.

Retention looked good on paper, but the reality was decline.

Pitfall:
Equating retention with success.

Consequence:
Stagnation, lowered standards, team frustration.

Avoid it:
Retain for growth, not comfort. High performers stay because they're challenged and cared for.

*LinkedIn: **94% of employees** would stay longer if companies invested in their growth.*

Pitfall 6: Lack of Continuity Planning

I once managed 60 outsourced engineers. They were high-performing, great people. Then the supplier was sold.

Overnight, prices spiked, culture changed, and continuity collapsed.

Exiting was painful and expensive in the busiest season of the year.

Pitfall:
Assuming great teams last forever.

Consequence:
Supplier changes, leadership exits, or resignations derail the roadmap.

Avoid it:
Bake continuity in with succession plans, documentation, contracts that protect IP and flexibility.

*PwC: **39% of CEOs** are "extremely concerned" about continuity risk, but most don't build structures to manage it.*

Pitfall 7: Misusing AI

Eman's View (CTO):

One of the most dangerous pitfalls I see today is treating AI like a magic wand. Teams paste prompts, trust outputs that "look right," and ship them without question. This leads to vibe code drift: plausible code that hides fragility.

Pitfall:
Using AI recklessly, assuming more code = more productivity.

Consequence:
Fragile codebases, regressions, false confidence.

Avoid it:
Use AI with guardrails. Regenerate instead of patching. Automate verification. Keep humans in control of ambiguity and accountability.

*GitHub: **36% of developers** admit to shipping AI-generated code they didn't fully understand..*

CLEVERBIT SOFTWARE

Common Pitfalls and How to Avoid Them

Bringing It Together

The common thread across all these pitfalls? None of them are about weak people. They're about weak systems: bad hiring choices, the wrong structures, short-termism, missing guardrails.

High-performance teams aren't built by avoiding mistakes entirely. They're built by designing systems where these mistakes don't become fatal. That's the real test of leadership.

Conclusion:

The Choice Ahead

Conclusion

The Choice Ahead

Building high-performance teams is hard. But it's not impossible. The choice is whether you accept fragility, with teams that look busy, suppliers who change the rules, codebases that drift, or whether you design for resilience.

Some companies burn people out and replace them. If that's your model, fine. But if you want performance that lasts, you need ownership, alignment, trust, continuity, and AI maturity.

The outsourcing world is broken. It locks you in, hides the truth, and leaves you exposed. That's why we built something different.

We're the underdog, and we like it that way. Because we've lived the pain of getting this wrong, and we've built a model that fixes it.

Use this playbook if you want to go it alone. And if you don't want to carry all the risk yourself, know there's another way.

The choice is yours.

Checklist:

Am I Building a **High-Performance Team?**

- Do my engineers know why they're building, not just what?
- Can engineers challenge the business without being divas?
- Do we balance psychological safety with accountability?
- Are roles clear enough that people can move with autonomy - and even fail - without chaos?
- Is retention driven by growth, not comfort?
- Is the team structure balanced, not built for comfort?
- Is governance a safety net, not a straitjacket?
- Can we survive the departure of any key leader or supplier?
- Is transparency human as well as technical?
- Do my teams care about the long term, and have I given them a reason to care?
- Are we using AI with maturity - guardrails, regeneration, human oversight - or drifting into fragility?

About the Authors

Simon Azzopardi

Partner

Simon has spent the last 15 years building and leading technology teams across startups, scale-ups, and enterprises.

He's carried the scars of missed deadlines, outsourcing failures, and hard-fought turnarounds. Those experiences shaped his conviction that high-performance teams aren't an accident. They're designed.

Eman Zerafa

CTO, Engineering & AI Leadership

Eman has led teams through high-stakes deliveries, always pushing for accountability and ownership as the core of performance. He believes the best teams act like partners, not order-takers, a principle he has carried from client trenches to boardroom strategy.

As CTO of Cleverbit, he extends this philosophy into the era of AI, exploring how new tools can amplify delivery without losing guardrails, accountability, or long-term focus.