// ARTICLE · PUBLISHED 29 OCTOBER 2026

Engineering Team Turnarounds: Nobody Gets to Blame History

Leading broken, acquired and divided engineering teams back to delivery

Jason Newell · MAX Research Collective · Article ART-BIZ-006 · Published 29 October 2026
Licensed under the Creative Commons Attribution 4.0 International License (CC BY 4.0), https://creativecommons.org/licenses/by/4.0/
Original: https://www.maxresearchcollective.com/articles/engineering-team-turnarounds.html
Cite as: Newell, J. (2026). Engineering Team Turnarounds: Nobody Gets to Blame History. MAX Research Collective, ART-BIZ-006. https://www.maxresearchcollective.com/articles/engineering-team-turnarounds.html

The conditions break the team

Over 25 years I've been handed a lot of engineering teams that were described, usually in the first meeting, as a problem. Some came with an acquisition. Some were scattered across four states or five countries. Some had stopped shipping. In others, "us" meant the people who built the product and "them" meant whoever had just bought it, reorganized it or shown up to manage it.

These teams are rarely short of good people. What breaks them is the conditions around them: unclear ownership, a budget nobody has actually mapped, a decade of deferred investment, and leaders who walked in and started by explaining everything that was wrong. The way back runs through servant leadership, the idea Robert Greenleaf put a name to in 1970, that a leader's first job is to serve the people doing the work. It starts with respect for what was built, it modernizes in slices that customers can feel, and it pays for itself once the real costs are written down.

Start by respecting what was built

The fastest way to lose a legacy team is to criticize its code. Every system you inherit was built by people making reasonable decisions with the skills, tools and

budget they had at the time. A monolith from 2012 was the right call for its moment, and so was a hand-rolled deployment script. The moment has simply passed.

So I set a rule early. We leave old design alone, and we don't criticize history. The people who wrote that code are often sitting in the room, and the ones who left are still somebody's former teammate. Our job is to decide what comes next.

This matters more than it sounds. A team that believes the new leader thinks its work is garbage will defend that work, and every modernization conversation turns into a fight. A team that believes you respect what it built becomes your best guide to where the real risks are buried.

Don't reinvent the wheel

When a system is slow, brittle and expensive, the instinct is to rewrite it. It's the most common choice I see. It's rarely the best one. Big rewrites take longer than planned, freeze feature work while customers wait, and tend to rediscover the edge cases the old system already handled.

The better path is usually incremental. Martin Fowler described the strangler fig pattern in 2004: wrap the old system, route traffic through a layer you control, and replace one capability at a time until the old core can be retired. These are the methods I lean on most:

- Add a layer. Put an API gateway or facade in front of the legacy system. An API gateway is a single front door that every request passes through, so you gain one place for logging, metrics, security and gradual routing without touching the core.
- Simplify before you replace. Remove dead features, consolidate duplicate services and retire unused integrations. Less surface area means cheaper hosting and fewer incidents.
- Adopt modern tooling around the code. Continuous integration and delivery (CI/CD), which builds, tests and ships every change automatically, can transform delivery speed long before a single line of business logic is rewritten, and so can automated tests, infrastructure as code and observability.

- Finish the cloud move. Many teams I inherited had been lifted into the cloud and then left there, running cloud-hosted servers the way they ran the data center. Picking that work back up and moving to native cloud services, such as managed databases, queues, serverless functions and autoscaling, often delivers the biggest gains in speed and the biggest cut in spend.

Each slice ships value on its own. The team sees progress in weeks rather than years. So does the business.

Modernize in ways customers can feel

Modernization earns its budget only when existing customers notice. Every improvement I prioritize has to land in one of three buckets: performance, usability or new capability. In practice that means faster page loads, fewer outages, cleaner workflows and the features customers have been asking for.

This is also what turns the team around. Engineers who have spent years absorbing complaints about a slow product get energized when the support queue goes quiet and customers start saying thank you. Happier customers and a healthier team tend to arrive together.

Be honest, and put everyone in the room

Acquired and dispersed teams are usually starved for information. Rumors fill the gap, and rumors almost always assume the worst. The cure is honesty, delivered early and often, even when the news is uncomfortable.

The most effective tool I've found for breaking down "us versus them" is simple: include the whole team in meetings. That means everyone, well past the leads, headquarters and whichever time zone is convenient. When remote engineers, acquired engineers and long-tenured engineers hear the same message at the same time and can ask the same questions, the factions start to dissolve.

Google's Project Aristotle, which studied 180 of the company's teams, found that psychological safety was the strongest predictor of team effectiveness. Psychological safety is the shared belief that it's safe to take interpersonal risks in

front of the team. Half a team can't build it while learning about decisions secondhand.

Being honest also means admitting what you don't know yet. "I don't have an answer on that, and here's when I will" builds more trust than a confident guess that turns out to be wrong.

Cross-train and eliminate single points of failure

Every struggling team I've inherited had at least one person who was the only one who understood a critical system. That person is usually exhausted, often quietly resentful, and a serious business risk. If they leave, get sick or take a vacation, delivery stops.

Cross-training fixes three problems at once. It reduces risk, it spreads knowledge across the "us versus them" lines, and it gives the overloaded expert room to breathe and grow. Pair programming, a rotating on-call schedule, documented runbooks and a second engineer shadowing every critical deployment are cheap and effective ways to start. A runbook is the written, step-by-step procedure for running or repairing a system, so the knowledge outlives the person who had it.

Have the honest conversation about pay

Many engineers on legacy teams feel underpaid, and after the last decade many of them have a point. Salaries ran hot for years while their raises stayed flat. The market has since changed: hiring has cooled, budgets are tighter, and the compensation gap won't close on its own or all at once.

I tell teams the truth about that. Pay can't be fixed overnight without a budget that almost certainly doesn't exist yet. What I can work on are the things people say matter to them: meaningful work, flexibility, a path to learn modern technology, recognition, a manager who advocates for them, and a credible plan to improve compensation as the business case improves. A surprising number of engineers stay for those things when they believe the leader is being straight with them.

Newell, J. (2026). MAX Research Collective, ART-BIZ-006. CC BY 4.0

Right-size the team, and plan for turnover

In any turnaround, plan on one or two people leaving. Some won't want the change. Some will find a better offer. That's normal and healthy.

Sometimes you need to help it along. Larger, older teams often carry a few seat warmers who stopped contributing years ago and got by because nobody looked closely. Ask honestly whether the team can function without them. If it can, those dollars are better spent stabilizing pay for the people who carry the load. Handle these exits with dignity and fairness. Make them anyway. Keeping people who don't contribute is unfair to everyone else on the team, and the team always knows who they are.

Ask how people are doing as well as what they're doing

Status updates tell you what someone is working on. They leave out whether that person is burning out, looking for another job, or sitting on an idea that could save the product. You learn that only by asking how they're doing and actually listening to the answer.

The data backs this up. Gallup estimates that managers account for at least 70 percent of the variance in team engagement, and that employees who have regular one-on-ones with their manager are almost three times as likely to be engaged as those who don't. On distributed teams, a consistent one-on-one is often the only real human connection someone has with leadership. Protect it.

Mentor, and help people mentor each other

Look for every chance to mentor. Pair senior engineers with junior ones. Have the legacy expert teach the cloud-native newcomer how the business actually works, and have the newcomer teach the expert the new tooling. Mentorship that crosses the old boundaries is one of the fastest ways to turn two camps into one team.

Newell, J. (2026). MAX Research Collective, ART-BIZ-006. CC BY 4.0

Spend the budget that exists, and spend it on people

Training budgets on struggling teams often go unused because nobody had time or nobody pushed. Find that money and make sure it gets spent on certifications, courses, conference tickets and cloud credits for hands-on labs. Training is also one of the most affordable retention tools available when salary increases are limited.

Set aside room for spot bonuses. A small, timely bonus for the engineer who stayed up to fix a production outage, or who documented a system nobody else understood, says clearly that the contribution was seen and valued.

Learn the real budget and costs, and write it down

A credible case for change starts from what things cost today. Learn the budget in detail: salaries and contractors, cloud and hosting, licenses, support contracts, and the hidden costs of incidents and manual work. In many inherited organizations this has never been documented, or the documentation is years out of date.

Writing it down is one of the most valuable things a new leader can do. It almost always surfaces waste, such as idle servers, overlapping tools, licenses for people who left, and contractors doing work the team could own. It also sets the baseline every future decision will be measured against.

Build the pitch with the team

Collect improvement ideas from the team and talk through them together. Tie each one to a benefit for existing or new customers, such as faster performance, fewer outages, better usability, new features or a lower cost to serve. Engineers who helped shape the plan will fight to deliver it.

Then make the pitch for a new, realistic budget. A wish list won't survive the room. A plan will, when it allows for meaningful change and is built around real deliverables the business can see and justify, with clear milestones, measurable outcomes and the savings that help pay for it. When leadership can see the cost baseline, the

customer impact and the path, the conversation shifts from "why does engineering cost so much" to "what do you need to keep going."

What this looked like in practice

These patterns sit behind results I've been part of:

- Post-acquisition turnaround at Trimble. I stabilized a team after layoffs and an acquisition, restored delivery, and realigned roughly $1M in projected staffing cost. I also started and planned a production multi-agent AI orchestration architecture, and used VS Code and Cursor across multiple LLMs to understand and upgrade complex software libraries and versions, work that called for surgical changes.
- Platform turnaround at AMN Healthcare. I led an organization of roughly 120 people across four states that improved platform performance by 200 percent and delivered about $240K in annual savings.
- Scaling through growth and acquisitions at Clinicient. Engineering grew roughly tenfold through multiple funding rounds and two acquisitions, while cost per engineer fell 18 percent.
- Operational efficiency at Cambia Health Solutions. Operational efficiency improved roughly 50 percent across the technology operations of a multi-brand health plan.
- Distributed leadership. The teams spanned the United States, Canada, Europe, India and Mexico, where including everyone in the room was a requirement from the first week.

Why it matters

In every one of those situations, the turning point came when the team believed that leadership respected what they had built, was being honest with them, and was invested in their future. Frameworks and rewrites came later — when they came at all.

The cost of skipping that step lands on everyone. A leader who opens by explaining what's wrong spends the first year fighting the people who know where the risks are buried, while the one expert who holds the critical system quietly updates a résumé and customers wait on a rewrite that keeps slipping. Servant leadership is the harder version of the job, and it's the most reliable way I know to turn a broken team into one that ships.

Sources

- Praerit Garg, "The one leadership approach that transforms engineering teams," Smartsheet.
- Google re:Work, "Understanding team effectiveness," the Project Aristotle findings.
- Gallup, research on managers' share of the variance in team engagement and on regular one-on-one meetings.
- Martin Fowler, "Strangler Fig Application," martinfowler.com, 2004.
- Robert K. Greenleaf, "The Servant as Leader," 1970.
- Addy Osmani, "Leading Effective Engineering Teams."
- LeadDev, "How to be the best servant leader for your team."
- Baylor University Human Resources, "Manager Best Practices: One-on-One Meetings," citing Gallup.
- Atlassian, "1-on-1 meeting tips," citing Gallup.
- "Servant leadership and the Scrum team's effectiveness," Leadership & Organization Development Journal.
- vFunction, "The Strangler Architecture Pattern for Modernization."

Glossary

TERM	DEFINITION
API gateway	a single front door placed in front of one or more services, so every request passes through one point. It's where logging, security and gradual rerouting get added without changing the system behind it.
Autoscaling	a cloud service that adds or removes computing capacity as load changes. It cuts the cost of servers sized for the busiest hour.
CI/CD (continuous integration and delivery)	automation that builds, tests and ships every code change. It turns a risky release day into a routine event.
Facade	a simplified interface wrapped around a complicated system. It lets new code talk to old code without learning its internals.
Infrastructure as code	servers, networks and settings written as files that can be reviewed and rerun. It makes an environment reproducible instead of hand-built.
Managed database	a database the cloud provider runs, patches and backs up. It moves routine operations work off the team.
Observability	the logs, metrics and traces that show what a running system is doing. It's how a team finds a fault before customers report it.
On-call rotation	a schedule that shares responsibility for answering production problems out of hours. Rotating it spreads both the load and the knowledge.
Psychological safety	the shared belief that it's safe to take interpersonal risks, such as asking a question or admitting a mistake, in front of the team. Google's Project Aristotle found it was the strongest predictor of team effectiveness.
Queue	a service that holds messages between two parts of a system until the receiver is ready. It keeps a slow component from stalling the rest.

Newell, J. (2026). MAX Research Collective, ART-BIZ-006. CC BY 4.0

TERM	DEFINITION
Runbook	the written, step-by-step procedure for running or repairing a system. It keeps critical knowledge from living in one person's head.
Servant leadership	Robert Greenleaf's 1970 idea that a leader's first job is to serve the people doing the work. It's the approach this piece argues for.
Serverless function	code the cloud provider runs on demand, with no server for the team to manage. It suits small jobs that run occasionally.
Single point of failure	one component, or one person, whose loss stops everything that depends on it.
Spot bonus	a small bonus paid soon after a specific contribution. It recognizes the work while it's still fresh.
Strangler fig pattern	Martin Fowler's 2004 name for replacing a legacy system one capability at a time behind a layer you control, until the old core can be retired.

CAREER & LEADERSHIP ART-BIZ-006

// ABOUT THE AUTHOR

Jason Newell leads organizations and people in technology, and is a researcher, deep technologist and builder — from a CRAY to agentic AI. He founded MAX Research Collective to build AI for regulated industries that runs where its owner has control, with healthcare as its strongest focus. About Jason → Connect on LinkedIn →

Newell, J. (2026). MAX Research Collective, ART-BIZ-006. CC BY 4.0