International Journal of Computer Science Engineering Techniques – Volume 6 Issue 5, September - 2022

Scaling Data Engineering Teams: Leadership Models and Organizational Design

Kuladeep Sandra
Independent Researcher
kuladeepsandra90@gmail.com

Abstract*—Most data engineering managers learned technical skills, not management. The transition from senior individual contributor to manager of a 30-person team is non-obvious, poorly documented in the literature, and frequently painful for everyone involved. This paper is a personal case study of that transition, documenting lessons learned over several years of growing and restructuring a data engineering organization across 6 business units and three time zones in banking, insurance, and manufacturing environments. The paper addresses three organizational questions: what organizational structures enable effective management and individual contributor growth in data engineering teams of 30 to 50 people; how hiring practices, conflict resolution, and promotion transparency correlate with retention and performance; and at what team size particular management structures break and what the warning signs are. The honest contribution is the failures: starting out of depth at age 38, two bad hires that took nine months to exit, a span-of-control crisis that required restructuring from eight direct reports to five, and a technical disagreement that turned out to be old promotion resentment. The thesis is that management is a learned skill rather than an innate talent, that the patterns that work at 10 people break at 30 and the patterns that work at 30 break at 100, and that organizations whose engineering managers learn these lessons fastest produce substantially better outcomes.

***Index Terms*—engineering leadership, team scaling, organizational design, data engineering, management structure, hiring practices, team dynamics, organizational behavior**

I. INTRODUCTION

The author became a data engineering manager at 38, after roughly twelve years as an individual contributor and senior engineer. The promotion was the standard one: promoted from within because that was the path of least resistance, despite having no management training. The first four months were genuinely out of depth, going through the motions of meetings that were not understood and making decisions on instinct. This paper is the retrospective that the author wishes someone had handed to them at 38, based on personal case study and lessons learned from growing and restructuring a 30-person data engineering organization over several years across banking, insurance, and manufacturing contexts.

The team now managed spans 6 business units and three time zones with technology stack including Apache Iceberg, Spark, Kafka, Flink, Trino, Kubernetes, and Ceph. The operational outcomes are documented in companion papers, including 1.5M in annual storage savings, query latency reduced by over 80 percent through topology optimization, 720K files compacted to 3K, the data fabric implementation, and compliance automation. The outcomes are real and the team's, not the manager's alone. What this paper documents is the management work that made the team capable of producing those outcomes, including the management mistakes that almost prevented them.

This paper addresses three research questions: (RQ1) What organizational structures and leadership models enable effective management and individual contributor growth in data engineering teams of 30 to 50 people? (RQ2) How do hiring practices, conflict resolution, and promotion transparency correlate with team retention and performance in technical organizations? (RQ3) At what team size do particular management structures break, and what are the warning signs that a structure is no longer working? The audience is experienced engineers considering their first management role, or new managers struggling with scale.

II. LEADERSHIP MODELS AND ORGANIZATIONAL STRUCTURES

Four organizational structures recur in data engineering teams of the 30-to-50 person range, and each has a regime where it works and a regime where it breaks.

A. Functional Structure

In the functional structure, all engineers report directly to the manager. This works well at 5 to 8 people because the manager can know every engineer well, attend every code review, and make every architectural decision. It breaks above around 8 people because the manager runs out of hours in the day and starts making decisions without context.

B. Matrix Structure

In the matrix structure, engineers report to domain leads who report to the central manager. The central manager focuses on platform direction and the domain leads focus on the day-to-day. This works well at 15 to 30 people. It breaks when the matrix produces ambiguity about who decides

what, when the engineer reports to two people with different priorities and serves neither well.

C. Layer-Based Structure

In the layer-based structure, junior, senior, and principal engineers are organized as career layers, with seniors leading projects and principals setting technical direction. This works well when the team has enough seniority distribution to populate the layers and breaks when there are not enough principals to set direction, or when the layer boundaries become sources of resentment rather than clarity.

D. Domain-Aligned Structure

In the domain-aligned structure, each business domain has its own data engineering capacity, embedded with the domain rather than reporting through a central data team. This is the data mesh organizational pattern. It works well when the domain teams have engineering maturity and breaks when they do not. The author's team evolution went from functional to matrix to a hybrid that mostly resembles domain-aligned but retains a central platform enablement squad. Each transition was forced by the previous structure breaking, not by foresight.

III. THE PERSONAL JOURNEY

A. The First Four Months

The first four months as a manager were dominated by problems the author did not know they were supposed to solve. People had compensation questions they did not have authority to answer. Engineers were unhappy in ways that could not be diagnosed. Architectural decisions came with the expectation of an opinion, and the opinions given were the wrong shape—reasoning as an engineer when they should have been reasoning as a manager about how the engineers would experience the decision. They were working 60-hour weeks and accomplishing less than at 40 hours as an individual contributor.

The single most useful thing that happened during this period was a more senior manager telling them that management is a learned skill and that nobody is good at it the first year. That sentence reframed what was happening. They were not failing at something they should have known how to do; they were learning a job they had never been trained for. The framing did not make the job easier but it stopped them from interpreting every difficulty as a personal failing.

B. Span of Control: 8 to 5

By month nine, the team had grown and the author had eight direct reports. Each one needed a weekly one-on-one. Each one had goals to set, performance to review, conversations about career development, conversations about compensation, conversations about whatever was on their mind. Eight one-on-ones a week sounds manageable until you add the meetings owed to the manager's own manager, the cross-team meetings owed to peer managers, the architectural review meetings, the planning meetings, and the time needed to actually think about anything.

The author was missing one-on-ones, rescheduling them to next week and then rescheduling next week. They were showing up to the ones made without preparation. The engineers noticed. The ones who needed coaching the most were the ones whose meetings were being canceled, because they were the ones with the hardest conversations.

The fix was to restructure by creating two senior engineer roles with formal team-lead responsibility, transferring three direct reports under each, and going from eight to five directs. The transition was awkward—the engineers who got moved felt demoted, the engineers who got promoted to lead were anxious about their new responsibilities, and they felt like they were abandoning their team. None of those concerns turned out to be warranted. Within a month, they had recovered roughly four hours per week. The team leads were actually closer to the day-to-day than they had been, because they were peers of the engineers they led rather than three steps removed. The engineers had better coaching than had been able to be given.

The lesson: span of control above six is unsustainable for direct technical management. The number is not arbitrary; it is roughly the upper bound on how many people a manager can know well enough to coach. Above six, something has to give, and what gives is the quality of the coaching.

C. The Two Bad Hires

In year two, the author made two bad hires. Both were technically competent. Both were the wrong fit for the team. Both took nine months to exit through the formal performance management process. The cost of those nine months was substantial and not just financial—the team's morale, the time spent on the performance conversations, the energy that should have gone into other work and instead went into managing the situation.

The hiring mistakes had a common shape. Both candidates interviewed well on technical skills. Neither candidate was assessed seriously on values or on how they would handle disagreement. Both came in with a confident style that masked an inability to collaborate. By the time the patterns became clear in their day-to-day work, they had already alienated the engineers around them, and the recovery path was either firing or restructuring around them. The decision was made to fire both in formal processes that took nine months.

The lessons from those two hires shaped how hiring is done now. Technical screening is necessary but not sufficient. Values and collaboration assessment must happen early in the interview process, not late. Disagreement questions ('tell me about a time you disagreed with a senior colleague and what you did') are more revealing than technical questions, because they expose how a candidate handles conflict, which is the single most important predictor of how they will fit into a team.

D. The Kafka vs. Pulsar Conflict

The second instructive failure was a technical disagreement that turned out to not be about technology. Two senior engineers had been arguing for weeks about whether the messaging infrastructure should standardize on Kafka or Pulsar. The arguments were detailed, technical, and apparently unresolvable. Multiple architecture review sessions were spent trying to mediate them on the technical merits, with no progress.

What was missed for too long was that the two engineers had a history. One had been promoted to senior eighteen months earlier; the other had not. The not-promoted one had concluded that the promotion was unfair, had never voiced the grievance directly, and was now expressing it as opposition to every technical position the promoted engineer took. The Kafka vs. Pulsar argument was a proxy for 'I should have been promoted instead of you.'

The fix was not a technical decision. The fix was a one-on-one with the not-promoted engineer that surfaced the actual grievance, followed by a conversation about the promotion decision and what would need to happen for the next one to go differently. The technical conflict evaporated within two weeks of that conversation. The lesson: technical disagreements that resist resolution often mask personal issues, and the most useful thing a manager can do is to look for the personal issue rather than to spend more time debating the technical merits.

IV. HIRING AND FIRING PRACTICES

A. Hiring

The hiring practice that emerged from the bad-hire experience has four elements. First, values screen first: a 30-minute conversation focused on collaboration, disagreement, and previous-team dynamics, before any technical screen. If the values screen is bad, no amount of technical talent makes the candidate the right fit. Second, scenario-based technical questions: not 'what is the time complexity of merge sort' but 'here is a system with these constraints; how would you design it, and what would you measure to know if it was working.' Scenario questions reveal how a candidate thinks rather than what they have memorized. Third, disagreement question: 'tell me about a time you disagreed with a senior colleague on a technical decision.' The follow-up 'and what happened next' is the revealing one. Candidates who have never disagreed with anyone are either lying or are not senior enough. Candidates who describe disagreement in terms of 'winning' are red flags. Fourth, reference conversations focused on collaboration: not the standard reference questions about technical capability, but pointed questions about how the candidate handled conflict and how they treated junior colleagues.

B. Firing

Firing is the part of management that nobody enjoys and that most managers do badly. The principles that emerged from doing it badly the first few times: document early and consistently, so that when the conversation has to happen there is a written record rather than a he-said-she-said; be clear and direct, so that the person being let go does not leave the conversation confused about what happened; be respectful of the person, because they are still a person and they will have a next role and how they remember this organization matters; make the process fair and not arbitrary, because the rest of the team is watching and will calibrate their own behavior to whether they trust you to be fair.

The nine-month exit processes for the two bad hires were too long, and most of the length came from inadequate documentation early on. Better documentation in the first two months would have made the path to exit faster and would have spared everyone (including the people being managed out) months of difficulty.

V. CONFLICT RESOLUTION AND PROMOTION

A. Conflict Resolution

The Kafka-vs-Pulsar lesson generalizes. Most of the technical disagreements that have escalated as manager have turned out to have a non-technical root cause: an old grievance, a status concern, a resource allocation that someone perceived as unfair, a promotion decision that someone is still angry about. The technical surface of the disagreement is real, but it is not where the resolution lives. The resolution lives in addressing the underlying issue, after which the technical surface usually resolves itself.

The practical pattern is to spend more time in one-on-ones with the people involved than in joint meetings about the technical question. The one-on-ones surface the personal context that the joint meetings cannot, because the joint meetings are inherently performative.

B. Promotion Transparency

Promotion decisions are the highest-stakes decisions a manager makes for the people on the team. They affect compensation, career trajectory, and sense of fairness. Opaque promotion decisions corrode trust faster than any other management failure observed.

The practice that works has three elements. First, documented criteria: what does it take to be promoted from level X to level Y, written down and shared with the team in advance, so that engineers know what they are working toward. Second, documented justification: when a promotion happens, the justification is written down and is available to the engineer being promoted. When a promotion does not happen, the reasons are documented and shared with the engineer who was considered. Third, predictable cadence: promotions happen on a known schedule rather than at the manager's whim, so that engineers know when their next opportunity is.

The practice does not eliminate disappointment, but it eliminates the worst category of disappointment, which is

the engineer who feels that the decision was arbitrary or unfair.

VI. ORGANIZATIONAL PATTERNS AT SCALE

A. When to Create Layers

Layers (junior, senior, principal) become useful when the team is large enough that career progression needs visible structure. Below around 10 people, layers are bureaucratic overhead. Above around 20 people, the absence of layers produces ambiguity about who decides what and who is responsible for what.

B. When to Distribute to Domains

The domain-aligned structure works when the domain teams have engineering maturity. Pushing data engineering capacity into a domain team that lacks the engineering culture to absorb it produces low-quality outputs and frustration on all sides. The signals that a domain team is ready: senior engineers within the domain who can mentor the embedded data engineers, a domain product owner who treats data as a product, and a willingness to take on the operational responsibility that comes with ownership.

C. The Platform Enablement Squad

A small central platform team is essential even in a fully domain-aligned model. The squad's job is to maintain the shared infrastructure that all domains use—the lakehouse, the streaming infrastructure, the orchestration, the governance tooling—and to provide consultative support when domain teams encounter problems beyond their expertise. The squad is small (2 to 4 engineers) and is the highest-leverage role in the organization because everything the domain teams do flows through the platform.

D. On-Call and Operational Responsibility

On-call is the part of operational responsibility that surfaces every weakness in the organizational structure. Engineers who are on-call for systems they did not build and do not understand resent it; engineers who are not on-call for systems they did build do not feel the consequences of their design decisions. The right pattern is that the team that builds a system also operates it, and the on-call rotation is shared across that team. This is harder than it sounds because it requires that every team be staffed with the operational expertise to handle their own systems, and developing that expertise takes time.

E. Cross-Functional Partnerships

Data engineering does not exist in isolation. The teams that matter most for the partnerships are infrastructure (who run the underlying compute and storage), security (who care about access control and audit), finance (who care about cost), and the business unit data consumers (who care about the actual data). Investing in the relationships with these teams is not glamorous and it does not feel like engineering work, but it determines whether the data engineering team can ship anything.

VII. CONCLUSION

Returning to the three organizational questions: (RQ1) Organizational structures that enable effective management and individual contributor growth in 30-to-50-person teams are matrix or domain-aligned structures with team leads handling day-to-day coaching, a small central platform squad maintaining shared infrastructure, and explicit career layers for the engineers. Functional structures break above about eight directs. (RQ2) Hiring practices that screen for values and collaboration as rigorously as for technical skill, firing practices that are documented and respectful, and promotion practices that are transparent and predictable correlate strongly with retention and performance. The opposite practices correlate with the bad outcomes documented earlier. (RQ3) Functional structure breaks above approximately eight direct reports. Matrix structure breaks when accountability becomes ambiguous. Domain-aligned structure breaks when the domain teams lack engineering maturity. The warning signs are usually visible months before the structure formally fails: missed one-on-ones, repeated conflicts that do not resolve, attrition concentrated in one part of the team, and the manager's own sense of being perpetually behind.

The closing observation is that most of what was learned about management was learned by getting it wrong first. The two bad hires, the span-of-control crisis, the Kafka-vs-Pulsar conflict, the early months of feeling out of depth—these are not embarrassing details to be hidden in a retrospective; they are the curriculum. The managers who get good at this are the ones who pay attention to their failures and update their practice in response. The managers who do not are the ones whose teams suffer. The learning continues because the team keeps changing and the problems change with it.

REFERENCES

[1] T. DeMarco and T. Lister, Peopleware: Productive Projects and Teams, 3rd ed. Addison-Wesley, 2013.

[2] F. P. Brooks, The Mythical Man-Month, Anniversary Edition. Addison-Wesley, 1995.

[3] P. F. Drucker, The Effective Executive. Harper Business, 2006.

[4] A. S. Grove, High Output Management. Vintage, 1995.

[5] C. Fournier, The Manager's Path. O'Reilly Media, 2017.

[6] W. Larson, An Elegant Puzzle: Systems of Engineering Management. Stripe Press, 2019.

[7] N. Forsgren, J. Humble, and G. Kim, Accelerate: The Science of Lean Software and DevOps. IT Revolution Press, 2018.

[8] N. Forsgren, M.-A. Storey, C. Maddila, T. Zimmermann, B. Houck, and J. Butler, "The SPACE of Developer Productivity," Communications of the ACM, vol. 64, no. 6, 2021.

[9] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Eds., Site Reliability Engineering. O'Reilly Media, 2016.

[10] A. C. Edmondson, "Psychological safety and learning behavior in work teams," Administrative Science Quarterly, vol. 44, no. 2, 1999.

[11] D. Coyle, The Culture Code: The Secrets of Highly Successful Groups. Bantam, 2018.

[12] P. Lencioni, The Five Dysfunctions of a Team. Jossey-Bass, 2002.

[13] M. Armbrust et al., "Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics," in Proceedings of CIDR 2021, 2021.