Check for updates

Split Your Overwhelmed Teams

TWO TEAMS OF FIVE IS NOT THE SAME AS ONE TEAM OF 10

THOMAS A. LIMONCELLI

Queue welcomes the return of columnist Thomas Limoncelli! You may remember Tom from his column, "Everything Sysadmin" which ran from 2015 to 2020. His new column, "Operations and Life" begins with this issue and will focus on DevOps and IT operations from a uniquely personal perspective.

A friend came to me asking for advice. His SRE team was suffering from low morale. People were burning out. Attrition was high. People leaving often cited high stress levels as the reason. There was concern that their backfill replacements wouldn't last.

Then Todd (not his real name) told me the biggest shocker: The team is overworked, yet his boss won't let him hire more people.

The way he explained this gave me the impression that he thought this was the first time in the history of the world that a request for more staff was denied. I broke the news to him as gently as I could.

After wiping the tears from his eyes, I tried to refocus the conversation the best way I knew how. I asked the most powerful question you can ask an engineer: "What problem are you trying to solve?"

He responded, "How to convince my boss to hire more people!"

"Yes," I replied, "but what problem are you trying to solve?"

He thought for a moment and said, "Low morale? High stress? The fact that everyone is so overloaded that nothing gets done?"

Yes! Now we're talking. Hiring more people is a solution, not a problem. By restating the problem as one of morale and stress, we open the door to many possible solutions.

THE PROBLEM

I learned a lot about the team by discussing the problem.

His team was responsible for managing six systems: the physical infrastructure (wires, cables, and physical computers), as well as cloud infrastructure, plus a series of applications that ran on top of that infrastructure.

Onboarding new people took many months as they learned all the various systems, technologies and processes, policies, and procedures. Todd said they had tried different ways to train people, manage documentation, and so on. All that helped, but it was still a lot of information to keep in your head.

It was clear to me that the main cause of the team's stress was being responsible for so many wildly different systems. The team members weren't stressed by risky, high-stakes work, as you might expect. There weren't a lot of outages. However, people were stressed because they felt incompetent. Six major areas of responsibility meant that no matter how good you became at one thing, there were other areas that you always felt embarrassingly ignorant about.

In simple terms, the team was feeling overwhelmed. This stresses people out and leads to low morale.

Todd said, "But that's all in their heads!" and I agreed. Overwhelmed is a feeling, not a physical problem. If it were a physical problem, it could be surgically removed.

Psychologists' term for this is *cognitive load*: the amount of working memory resources used. Like a computer running slowly because it is running too many programs at the same time, your brain is overwhelmed.

When I talked with members of the team, I often heard phrases such as "I feel stupid" or "At my last job, I was the expert, but I've been here a year and I still don't know what I'm doing."

These were highly intelligent, experienced engineers, yet they frequently put themselves down. They weren't just being humble; they really felt inadequate.

He pointed out that it is normal to feel inadequate while learning a new skill, but every time he got to use that skill, he "got the dopamine hit of a job well done."

One team member told me something that explained this better than any book on management theory could. He said that with six areas of functional responsibility, he never did a single task long enough to get good at it. At previous jobs, he learned a task and then applied that learning to literally hundreds of projects. He pointed out that it is normal to feel inadequate while learning a new skill, but every time he got to use that skill, he "got the dopamine hit of a job well done."

On this team, with six major areas of responsibility, there was constant context switching. Every task would bring this team member back to the "feeling stupid" phase. There wasn't enough repetition to get to the "feeling of accomplishment" phase. Nobody wants a job that makes them "feel stupid" every day.

Yes, if he waited long enough, he would be assigned a task that let him use a skill he learned earlier. Skills fade if you don't use them, however, so he would go back to "feeling stupid," compounded by the fact that he would be kicking himself for "not taking good enough notes."

Engineers often say they enjoy learning new things, but I believe what they really enjoy is demonstrating that new knowledge. To achieve a feeling of accomplishment, they need to be able to demonstrate the new skill soon after learning it. Constant context switching among six areas of responsibility meant the dopamine hits were few and far between.

It wasn't always this way. Years earlier. the team was half the size and had half the responsibilities, and everyone was happier. There was no talk of burnout. Everything just worked better.

My suggestion was simple: Split the team in two and give each half as many responsibilities.

PUSHBACK AND CONCERNS

Todd initially pushed back against this idea. Why split the team when he could just have certain people specialize? I pointed out he had already tried that, and it wasn't working. Without the hard boundary that comes from having distinct teams, the engineers felt obligated to stay informed and be involved in all the decisions of the team. The specialists must attend meetings outside their specialty and hold the cognitive load of knowing all other functions.

No. To make this work, they needed the hard limits that come from organizational boundaries.

After some discussion, we decided to try reorganizing as two teams of five, each being responsible for three related areas. This would lessen the cognitive load, reduce context switching, and increase the likelihood of the kind of repetition that would lead to more "demonstration of skill" and less "feeling stupid."

HOW TO SPLIT A TEAM

The book *Team Topologies* (Manuel Pais and Matthew Skelton) has advice about how to divide teams using the concept of a fracture plane. This is a natural seam that allows the system to be split easily into two or more parts. These typically fall along lines of business domains, regulatory compliance, change cadence, team location, risk isolation, types of technology, or user personas.

In this case, there was an obvious natural seam along business functions: application versus the platform on which the application ran. Basically, this was splitting the team along architectural layers.

Having smaller teams would mean team members would be invited to fewer pro forma meetings.

The split would have several benefits:

- *Fewer communication paths.* Instead of 10 people trying to communicate with 10 people, most communication would happen within each five-person team, with the tech leads of each team bridging the two.
- *Easier to achieve consensus.* It is easier to get five people to agree than 10. The people making the decision would be more focused on the issue. It would remove from the decision-making process people that are unaffected or simply don't care.
- *Meetings that are more efficient.* The more people invited to a meeting, the more difficult it is to schedule; the more likely someone will be late; the more likely someone isn't paying attention and needs to ask for information to be repeated when asked for an opinion.
- *Fewer meeting hours per person.* Smaller teams would mean team members would be invited to fewer pro forma meetings.
- *Increased likelihood of repetition.* As described earlier,

we work better when we can learn a task and demonstrate that skill frequently. This requires a certain level of repetition in the type of work.

➡ *More clearly defined boundaries and separation of duties.* With one large team, the "big ball of mud" design pattern seems to come more naturally. What's a little layering violation among friends? On the other hand, two separate teams are forced to be more intentional about boundaries in technical areas, as well as in policies and processes. Separation into two teams would become a forcing function for well-defined interfaces between technical systems (APIs), as well as personal interfaces (rules of engagement).

➡ *Lighter-weight leadership responsibilities.* Instead of one overworked tech lead, now there would be two tech leads, each with a more manageable domain of responsibilities.

➡ *More leadership opportunities.* Some of the attrition was the result of a perceived lack of room for advancement. More teams would mean more opportunities.

SHARED ONCALL

One concern Todd had was how this would affect the on-call rotation. Being on call for one week out of 10 was great. The team liked being on call for no more than one or two weeks per quarter. With a five-person team, however, a team member could be on call four times each quarter, or 20 percent. That frequency is detrimental to project productivity.

Instead, the team decided to have a shared on-call rotation. They would cross-train. Each team makes a procedure book that covers any first-on-the-scene tasks for most alerts and issues. If you are on call for the other team's responsibilities, you would be trained in basic tasks but could escalate to the other team. To reduce cognitive load, you are not expected to memorize every task. Instead, all alerts and common issues are documented with a checklist.

You follow the steps of the checklist. If you get to the end of the checklist and the alert is still firing, you can escalate to the responsible team.

The checklists are treated like software: You could file a bug if you find something wrong, confusing, or incomplete. Their code-review process (github pull requests) would be used to suggest changes. If one team feels they are getting too many escalations, they could improve the documentation or the process. Thus, a feedback loop exists to ensure that documentation and training stay fresh and useful. Training new members basically involves reviewing the most-common checklists.

If you must do a task many times each day, you're going to make time to improve the process.

The increased likelihood of repetition not only helps morale, but also has some unexpected other benefits. For example, repeating the same or similar tasks provides more opportunity to make potential improvements to the process. If you do a task once a year and see how the task could be improved, there's little incentive to make those improvements. It isn't worth it. If you must do a task many times each day, you're going to make time to improve the process. Repetition leads to better, more efficient processes.

CHALLENGES AND OPPORTUNITIES

There are also many challenges with the split-team approach. People have to let go of old responsibilities. When you are good at something, it can be difficult to leave the task to someone else. Generalists have to pick one team or the other and give up tasks where they had developed expertise and comfort.

At the team level, many existing practices they had grown comfortable with changed. Meeting schedules were reworked. Policies and procedures changed.

On the other hand, there have been unexpected new opportunities. Managing two smaller teams can be less overwhelming than managing one large team. There has even been discussion about hiring a different manager for each team. Each kind of team might need a different kind of manager. For example, one team might do better with a manager who has operational management expertise, while the other team might need a manager with more hardware experience.

SUMMARY

This team's low morale and high stress were a result of the members feeling overwhelmed by too many responsibilities. The 10-by-10 communication structure made it difficult to achieve consensus, there were too many meetings, and everyone was suffering from the high cognitive load. By splitting into two teams, each can be more nimble, which the manager likes, and have a lower cognitive load, which the team likes. There is more opportunity for repetition, which lets people develop skills and demonstrate them. Altogether, this helps reduce stress and improve morale.

If your team is suffering from low morale and high stress, look at the cognitive load on the team, review its sources, and look for substantive changes that will have the desired impact. The solution might not be splitting the team, but that could be exactly what is needed.

Thomas A. Limoncelli *is an internationally recognized author, speaker, and DevOps advocate. He is a Technical Product Manager for SRE at Stack Overflow, Inc. He previously worked at small and large companies including Google, Bell Labs/Lucent, and AT&T. His books include* Time Management for System Administrators *(O'Reilly),* The Practice of System and Network Administration *(3rd edition) and* The Practice of Cloud System Administration *(Pearson).*