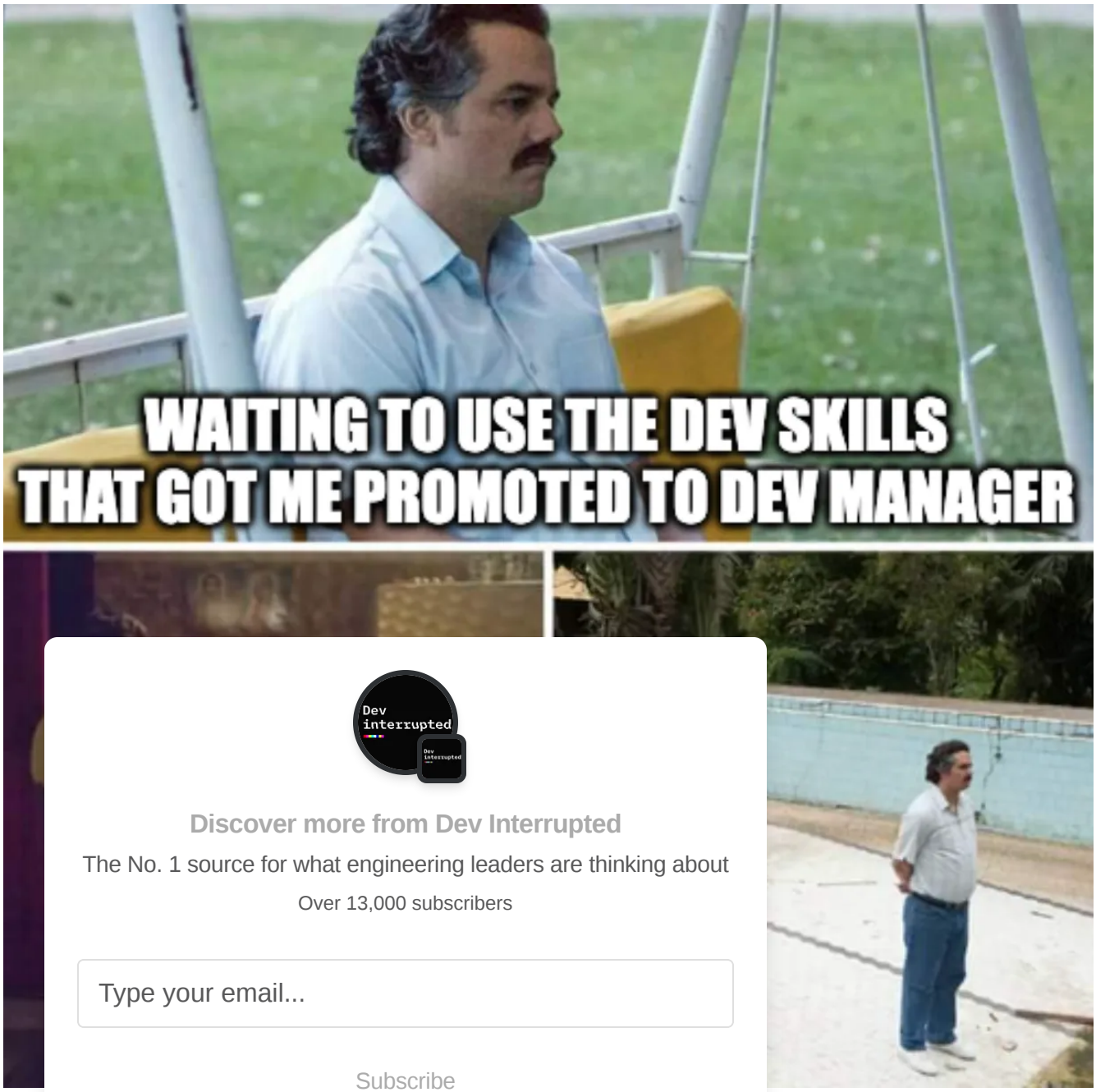
Promoted from Dev to Team Lead: 8 Things They Didn’t Tell Me

When I was promoted to Dev Team Lead, I was in over my head. Read the 8 things that I wish someone had told me when I was promoted.

DEV INTERRUPTED

MAY 18, 2023



I w

Thr and loving it.

Life was great. I lived in a small apartment in Southie (Boston) with my college roommate “Q.” I had a good job at a tech start-up called CloudLock. I hammered out code 12-14 hours a day. I worked so much that I never knew what day it was, and my bosses had to force me to go home. When I wasn’t working, I played bags (aka

cornhole) with my friends, lit people up in *Super Smash Bros Melee*, or slept. Not a care in the world.

Then a freight train hit me.

My boss, our VP of Engineering Michael Zeldich, pulled me aside one day. He explained our team was growing fast, and it was getting tough for him to have 15+ engineers reporting to him directly. We needed to put some team leads in place to scale our org and ensure everyone was getting enough attention.

Would I be interested?

Ok… I'm not going to lie. I was surprised. But it wasn't the first time I thought about it.

CloudLock was my second job out of college. My first job was working for Nuance Communications. I got a taste of what it might be like to lead a team when my boss there went on vacation for two weeks. He nominated me to be the engineering contact for tech support while he was out. In those two weeks, my network within the company expanded; I got on customer calls for the first time

and saw what it was like being responsible for more than just my own code. It was pretty fun, and I was good at it!

That experience was in my mind that day while Michael and I were talking. I wanted to say yes, but I had a million questions.

I don’t remember his exact words, but the gist of what Michael said was, “Don’t worry. You’re going to be great. I’ll help you, and we’ll make it work together.”

Michael is a really good guy. I trusted and respected him. So that was all I needed to hear.

My First Few Months as a Team Lead

Looking back on those first few months, I didn’t really understand the job.

I was going through the motions, mimicking all of the things I had seen other dev team leads do.

Don’t get me wrong. I did some good. But I also had quite a few struggles.

For starters, every time there was a problem, I went into Superman mode. Single-handedly fixing it at the speed of light. After all, I was a great coder and had an expanding set of knowledge of the entire system. I was good at helping other developers fix their problems, that is why I got promoted to team lead, right? (Kinda.)

Some of the team actually liked it at first. They acknowledged me for getting my hands dirty and being helpful and responsive.

But some of the team didn't like it. It came across as controlling. The ones who liked it initially stopped liking it because they were making the same mistakes over and over again and not getting better. Our weaker devs stayed weak and our stronger devs weren't growing.

There were more mistakes. Like when I waited too long to fire a bad developer whose negative behavior was hurting the team.

8 Things They Didn't Tell Me

I believe being a leader is a never-ending journey. There's always more to learn. Thankfully, I had great mentors who taught me a lot. And I also learned some lessons the hard way.

Here's 8 things I wish I knew back then.

1. Many of your skills don't translate.

The cruel irony is that there is a reverse connection between strong individual dev skills and dev team-lead skills. The strongest devs will have more of an uphill battle starting out as managers.

About 75% of the issues we face as a dev team lead are not technical. The job is mostly about people and processes. Once I realized this, everything changed for me.

I could fix anything in the codebase. I was great at finding creative solutions to fix problems other devs were having. Not only are those skills not super relevant anymore, there are other traits of great devs that can actually hurt you as manager...

The Superman complex. If the team has a technical problem (bug, technical blocker), great devs often have the instinct to jump in and fix it right that second. In fact, when I did that as a dev, I received praise from my peers and leaders for being a great team player. As a manager when you do that, you're the opposite of a team player.

Instead, we need to enable our people to solve the problem. Even if it takes longer the first time. Even if they make mistakes. Even if they don't do it as well we would.

By helping your team figure it out, instead of doing it for them, you'll get many benefits. Your people will see that you trust them. They'll learn more. They'll become more self-sufficient over time. And they'll also learn to help each other which will bring the team closer together.

Pro tip: *Avoid judgment at all costs. When your people feel free to get out of their comfort zone and make mistakes without fear of criticism, you'll see their true creativity come out and you'll be amazed at what they're capable of.*

Pro tip: *You can scale yourself by educating your people on your thought process. Help them understand why your instincts kicked in about a problem. Explain the process you go through to diagnose the issue. Explain your mental model for identifying fix options. Explain how you would communicate everything to the rest of the team.*

Deep focus. Another skill that did not serve me well as a manager was deep focus. As a dev, you have to get in the zone. I was good at locking in and focusing all of my energy on a single problem. That will kill you as a dev team lead.

Great leaders embrace context switching. They move around. They talk to a lot of people. If you find yourself locked in on a technical problem for hours, that's probably a sign that you need to delegate more.

When you do have the luxury of deep focus time, use it to think about strategic initiatives. Like how to propose your next big non-functional investment to your executive team or the profile of your next three dev hires.

2. Keep your instincts. Change your behavior.

Great devs have great instincts. Your intuition, which comes from your experiences, your expansive knowledge of the system, and your understanding of the end-to-end process, allows you to feel things even before you can even put your finger on exactly what it is. You sense when you went down the wrong path in your code. You sense when your team's iteration is behind schedule.

Continue to hone these instincts. Just don't act on them the same way you used to.

You need your spidey sense even more now to figure out when others need help:

- When you hear something in your daily stand-up that doesn't sound right.
- When someone can't find the root cause of a production issue.
- When you're helping out on a code review.

But if we aren't jumping in like Spiderman to save the day, then what?

(Superman? Spiderman? Make up your mind, dude! I know. Actually, Deadpool is my favorite superhero. What's the best Deadpool quote about engineering leadership you ask? "House blowing up builds character.")

First, take a breath. I process things quickly, but that can be a disadvantage as a manager. Keep listening and take extra time to process.

If you aren't hearing enough info, ask questions. Gather as much information as possible before you offer any advice. Sometimes, just asking the right question helps your dev think about the issue in a new way and come up with a solution on their own.

Pro tip: *The stand-up is an especially useful time to listen for things that might be slightly off. If I was worried someone was not on track, these questions always helped me figure out if I needed to dig deeper:*

- Has (fill in the dev or team who is dependent on this work) reviewed this?
- What are you thinking for scalability testing?
- Tell me about your feature roll-out plan?

Check out this awesome dashboard we use for our standup. It identifies who's blocked, who has too much WIP, and if our work is aligned with the business priorities. Learn more about Pulse!

3. Communicate “why” more than “what” and “how.”

As developers, we’re used to dealing with “what” and “how.” As dev managers, those are still relevant but it’s more important for us to focus on why.

There are four reasons for this:

Customer alignment: Product managers are hopefully delivering stories that clearly explain the customer problem and use case. But it’s still easy for devs to get in the weeds. If you constantly remind your team to come back to the problem and user experience, you’ll deliver a higher quality product more often.

Pro tip: *Another question I like to ask when a dev is stuck: “Can you describe what your user is going to be doing before, during, and after using this feature?” If they can’t, they need more info.*

Pro tip: *Encourage your devs to listen to customer support calls and sales prospects calls. In my experience, lots of teams say they are going to do this then they don’t. At LinearB we have a*

rule (voted on by the team) that every dev attends two customer calls minimum every month.

Business alignment: A big part of being a dev leader is translating executives to engineers and aligning your team's work to business objectives. I believe there's a business decision behind every line of code. Is the ultimate goal to acquire more customers and generate revenue? Or are we trying to increase customer satisfaction and drive renewals? Are we making a strategic investment in non-functional work to save money and drive higher profit? The more your people understand the big-picture impact of what they are working on, the more they'll be able to think strategically and make better decisions about how to write their code.

Mission and motivation: Most people want to feel part of something bigger. Sharing the "why" with your team helps connect them to the rest of the company. Also, regardless of what your company does, you have customers that rely on your product. Real people. Sharing

“why” helps developers buy-in to something bigger than just what you’re doing in a particular sprint which will make them happier and lead to stronger team culture.

Career path: The best bosses I had were setting me up for success for my next job. Whether your dev wants to continue down a technical path or move to a manager path, they’ll need to learn to see the big picture and be able to communicate “why” to others. I had mentors teaching me this and I try to pay it forward as much as possible.

4. Culture is a real thing. And you’re responsible for it.

Speaking of culture… it’s a real thing and you’re responsible for nurturing it.

Can I define culture? It’s easier to describe than to define, but that doesn’t mean it’s not there. Culture is like gravity. It’s the invisible thing that stops us from floating away from each other.

One thing I know about culture… It helps to have shared values. And it helps when you write them down, use them in your day-to-day work, and iterate on them over time.

Another thing I learned about culture… it flows from bottom-up and top-down. Your devs will know if you don’t believe it. They’ll see if you’re not living it. So it has to be authentic.

For me, when it comes to building culture, three ideas have served me well:

- Be kind and empathetic toward everyone.
- Follow through on what you say you’re going to do.
- Stay humble and keep everything focused on the team.

5. Clear a path for your people.

One thing I struggled with as a manager… computers operate differently than people. Seriously. Bear with me for a second.

Code is deterministic. You tell it to do something. It does it. If the code does the wrong thing you just fix it and it does the right thing. As developers, this is how our brains are wired.

But that's not how people work! You can't tell them what to do. It doesn't work. One day you'll get one thing. The next day you might get something completely different.

With people, all you can do is set them up for success and trust them to do the right thing. If you have great devs working for you, the best thing you can do is give them a clear path and get out of their way.

As a manager, I knew I was spending my time wisely if I was unblocking my team. There are three types of blockers that will come up every week:

Technical: We know all about this one. Try to avoid saving the day yourself. You're better off empowering them to fix it themselves

or to help each other to fix it.

Dependencies: We're used to dealing with this one too, but now we're in a better position to predict when issues will arise. We can use our visibility of what's happening across our team and other teams to anticipate problems and start communication sooner rather than later.

Priorities: After a while in my role as lead, I figured out that the #1 reason my team missed an iteration was that either my devs did not truly understand what their #1 priority was or they knew what it was, but they were still getting pulled into other things. This is a silent killer. Eventually, I focused a lot of my time throughout the week making sure every dev was working on the biggest things that would help us ship our iteration on time. Learn more about the LinearB shadow work detector.

Pro tip: *Instead of spending time in your daily talking about status updates that could easily be a Slack message, consider talking about priorities instead. Give your devs a chance to say what their #1 priority is and if anything is pulling them away from working on it. I bet you'll see a new level of engagement from your team and you'll deliver your iteration on-time more often.*

6. Priority setting and estimation takes up 50% of your time.

Speaking of priorities… as a dev, you do get asked for updates a lot. As a dev team lead, I found the requests for status updates increased by 10x.

The process of determining priorities and coming up with estimates for the business took up at least half of my time. I found it to be a fun puzzle and I enjoyed working on it.

Many articles have been written about how to estimate so I'm not going to get into detail on that now, but I will share some tips I learned:

- When it comes to estimation, be honest and don't play games. Don't tell them what they want to hear. If your business says "I'm sure you can do this faster", stand your ground. If you don't, it will catch up with you.
- Listen to your own devs as much as possible. Try to prioritize their non-functional requests because, in the end, those items will enable you to deliver more of what everyone else wants. Again, you'll get a lot of pressure from the CEO and other leaders to focus on customer features. Use data to show them you'll deliver twice as many features later if you do the non-functional now.
- Make friends with your product leaders and educate them on how the non-functional work you want can help get them what they want faster. If you have a united front with product management, you're much more likely to convince others.

7. Invest in your ecosystem.

When we get promoted, our circle expands by 3-5X. We're used to working with the devs on our team, our product owner and sometimes some devs on other teams.

All of a sudden we're regularly interfacing with all of the other dev managers, many of their devs and all of the leadership from product, tech support, sales, and HR. Not to mention the senior leaders in our own department.

These relationships really matter. You can't be successful in your new job without their help and you definitely won't get promoted to the next level without their sponsorship. So invest in those people. Learn what's important to them and go out of your way to make their life easier.

For example, with the sales team, ask them what would help their demos run better and work on something for them in your next hackathon.

8. Translate engineering to execs with data.

Your new boss (the Director, VP, or CTO) cares about different things than when you were a dev. They are probably going to be a lot more focused on data and metrics because that's all their boss cares about. So give it to them.

This is why I started LinearB. At CloudLock we lacked a clear set of metrics for communicating engineering progress to the business. It was always just "are we on track to deliver feature XYZ by the deadline?"

If you aren't sure what metrics you should share, I put this guide together with 17 metrics that we use at LinearB now to track and communicate dev team performance.

Do you know a Team Lead who needs to read this article?

Great Memories

I loved being a dev team lead. The gratification was no longer instant like it can be when you're directly building things. But it felt really good to help other people reach their potential. To watch them grow. And you're in one of the most powerful positions in the company. The perfect place to tactically make things happen and also influence strategy.

My little crew at CloudLock became a powerful force. We built features that thousands of security pros use every day. We helped lead the company to success from the bottom-up. Many of those devs went on to be architects, managers, and directors.

After two years, I got promoted to Director. Then later, VP of Engineering. I started this podcast (Dev Interrupted) talking to other engineering leaders. I was hungry to progress and experience new challenges, but I always think back fondly on those years. The higher up you go as a dev leader, the more lonely it is.

I miss problem-solving with the team. The late nights working side by side to hit a deadline. The inside jokes. There is nothing quite like a small, close-knit team.

So enjoy it. It won't last forever.

By Dev Interrupted Host Dan Lines - Originally published at LinearB

If You Liked My Blog, Come See My 3-part Summer Workshop Series for Engineering Executives

Engineering executives, register now for LinearB's 3-part workshop series designed to improve your team's business outcomes. Learn the three essential steps used by elite software engineering organizations to decrease cycle time by 47% on average and deliver better results: Benchmark, Automate, and Improve.

Don't miss this opportunity to take your team to the next level - save your seat today.

12 Likes · 1 Restack

Comments

Write a comment…