www.phparch.com

January 2021
Volume 20 - Issue 1

php[architect]

Newfangled Views

Encore, Encore, Bravo, Bravo!
Combining Front End Development with PHP Using Symfony and Vue

Evolving to Modern Front-End with ReactJS

Six Things I Learned as a Lead Developer

Free Sample Article

ALSO INSIDE

Six Things I Learned as a Lead Developer

Gabor “Mefi” Nadai

Being a Lead Developer is a fun, engaging, and responsible adventure. Becoming one is not something that happens from one day to the next. It is something you grow into by raising your voice, taking action, and responsibility. I’ve learned many things working as a Lead Developer. Still, I’d like to share six lessons, which were the most important or sometimes the hardest for me.

From Contribution to Coordination

First of all: this is not a deep-dive tech article. I won’t say a word about code writing, developer tools, frameworks, performance tweaks, or security. Don’t get me wrong: these are relatively essential topics, especially for a lead developer, but right now, I’d instead like to share my thoughts around the phrase: soft skills.

If you are a lead dev now or are soon to be one, you’ll quickly realize that your work’s real impact is no longer only measured by writing code. At this point, you’ve already proved your hard skills. However, with your shiny new title, people will expect different types of contributions from you. Of course: you still have to write code! You are still making an impact on that, but it is not your singular responsibility.

Becoming a lead developer also means that your responsibility will shift from an individual contributor towards a collaboration coordinator. You’re going to start making an impact by having conversations with fellow developers, project managers, product owners, or sometimes even users and non-technical folks.

You also will realize that many tech leaders stated that most of the job is done way before writing the actual code when making software. User stories, architectural plannings, API specifications, whiteboard discussions, data-driven decisions, monitoring, alerting, etc., are all important pieces of software engineering.

How To Improve Soft Skills?

To have meaningful conversations and achieve results, you need soft skills. And the funny thing is that these are only soft in the name, but I find it pretty hard to learn or practice them.

Soft skills or interpersonal skills are the non-technical aspects of how you are doing your work. These skills—like communication, conflict resolution, or time management—are just as necessary as hard skills like writing code, architectural designing, or creating tests.

You can improve your soft skills by learning and practicing new things as well, but you don’t always have the proper situations to exercise or even the proper materials to learn. You can learn how to develop a high traffic website by tutorials and pet projects running in a Docker container.

But imagine you’re going to have a difficult conversation with someone. You can rehearse it in front of the mirror, and you should do it, too. But it’s way more different from what it’s like with a real human in front of you.

Have you ever worked with a horrible boss? In my experience, that can happen because many companies promote people for a leadership position based mostly only on hard skills and without proper training. And sometimes it happens because “hey, you are the most senior member, so you are the leader now, let’s lead”.

Half of all Americans have left a job because of their manager, not the company or job itself, as found in “The State of the American Manager”[1], a broad study made by Gallup in 2015. So a vast majority of people actually leave managers, not jobs.

> *You manage things, you lead people*

This is my favorite quote, and it’s from Grace Hopper. A woman in technology. A woman in the US military. A woman who created the fundamentals of modern programming languages. She believed that you manage things, and you lead people. An excellent manager I worked with once told me that to become a good leader of people, I must become a great manager of things first. This was my very first experience in learning soft skills.

What Does “Lead Developer” Even Mean?

A title doesn’t matter, right? Still, here’s just a few examples of how many exist for this role:

> *Lead Programmer, Lead Software Engineer, Team Lead, Development Lead, Technical Lead, Lead Software Developer, Software Engineer Lead, Software Development Manager, Software Manager, Lead Application Developer, and so on.*

A lead developer’s exact responsibilities tend to vary from company to company. According to Wikipedia[2], in general, they are responsible for the underlying architecture for the software program and for overseeing the work performed by any other software engi-

1 *“The State of the American Manager”:* *http://phpa.me/state-american-manager*

2 *Wikipedia:* *http://phpa.me/wikip-lead-programmer*

neers working on the project. They also typically act as a mentor for new team members or software developers beginning their journey and all members of the team.

The most important thing is that a lead dev is responsible for both the process of execution and the executed job itself. The question is: how?

By Joining Forces

A bunch of people working together is just a group, not a real team. Teamwork is pretty essential, and you have a crucial role in achieving that.

Making a real team out of a group of people

A group of people together is not a team. A team is when people move, work, think together, and therefore they achieve results together. As a lead developer, you have a key role in defining and shaping the team's identity.

Having a deep understanding of the business model.

You don't need to have a deep understanding of everything about your project. But you need to have a deep understanding of the core business model. Both on a code-level and a business-level perspective. Which is the most business-critical component of your project? How does it work? What are the risks?

Speaking honestly, openly, and respectfully.

This advice, of course, shouldn't be limited to engineers working in a lead developer role. I don't need to explain the reasons deeply. I only have a question for you: would you rather work with someone who communicates honestly or with someone who isn't?

Talking in a way both stakeholders and teammates can easily understand.

Software engineering is a business. We create and maintain software to deliver value to our users. To avoid taking on technical debt and creating silos, it is vital to understand one fundamental idea: we're here together to achieve some kind of business goals. To do that, you need to collaborate and work together as a team with stakeholders continuously. But this can be tricky as your stakeholders most of the time won't be software engineers.

Making new connections.

Networking and making new connections is not the easiest thing to do. However, if you continuously grow your network, you can discuss ongoing topics, challenges with others working in the same area. And you never know: you might find your new team members over beers or coffee.

By Motivating People

To be honest, I believe that getting your paycheck is not a motivational source, more like a hygiene factor. It's a mandatory thing like having water in the office. In the long-run, you can't motivate anyone with money. Motivation is something coming from the inside, and if someone has this internal motivation, you can boost that and build on that in many ways.

Being compassionate about team health.

A team health check can be a great start and great help in being on the same page and having actual data about how your team's health and mood change.

Understanding what motivates people.

Moving Motivators can be a great start. For this, you mostly need to have a valuable conversation with your teammates and understanding what excites them.

Knowing when to go or when to slow.

Sometimes you need to push the team to meet an important goal, reach a deadline, etc. But sometimes, you need to stop and evaluate your current situation.

Leading by example in many ways.

In any leadership role, you will be a role model. Do everything in a way that people around you are watching what and how you are doing. A small example: it's not cool to always late from meetings, but in a leadership role as a role model, it's an absolute red-flag to always late from meetings.

By Having A Plan

Of course, your plan will change a lot, but that's not a problem as everything is always changing.

- Knowing what's next.
- Having a long-term vision for the codebase.
- Knowing why and how to get there.
- Being prepared for both the best and the worst-case scenarios.

By Keeping Things Tidy

You are kind of like the glue which connects people as a team. But you are also a hammer that strikes on obstacles cruelly.

Moving obstacles away before the team gets there.

I think a lead developer should think in advance and plan what's coming for the team. By having an eye on the discovery and the strategy, you can eliminate blockers for your teams, which results in speeding up delivery.

Maintaining good relations with colleagues outside the team.

This includes engineers and lead developers from other teams, but also people from marketing, product, sales, etc. Doing so helps you understand what's going on in the company and the challenges other teams have.

Saying no to spooky user stories.

If something is not ready to work on, you have a great responsibility to raise your voice and, moreover, help anyone make a better user story out of it.

And saying no for unacceptable deadlines.

Again, you have a responsibility in this. Your team should not be committed to something they obviously can't fulfill.

By Helping Others To Grow

One of your many responsibilities is making seniors out of juniors and leaving no-one behind.

Helping juniors on their way to be seniors.

You can help them with material things like books, training, meetups,

and conferences. But you can also help them, for example, with pairing, mob coding, and deep-dive code review sessions.

Helping POs, PMs, or any non-technical members of the team.

It is useful to take an ambassador role to help non-technical team members understand technical aspects, like why something is more complicated than it appears to.

Making decisions together with stakeholders.

You can go in this direction and have an empowering mindset in your teams. Or you can go the other way around, and you will feel that your group receives orders from "above."

Answering questions.

Answer questions both from inside and from outside of your team.

The Six Lessons

Here are the six most challenging and most important lessons I learned along the way

1. I Am The Boss... Of My Own Calendar

So you are a lead developer. Do you know what everyone wants from you?

Your time.

Do you know what is your most valuable resource of which you have the least?

Your time.

So yes, it is *your* time, so you are the one who decides when to do things. Certainly, there are deadlines you can plan with, but on-call duties or incidents could make it hard to manage your time. And there are those "Hey, you've got five minutes?" non-meetings fragmenting your time.

It is important to have slots for focusing on problems and working without interruptions. You need at least two hours of working without any interruption considered as "focused time". You need a decent amount of focused time. It is not easy, but I've learned a few habits along the way that can help you achieve this.

Assess Your Current Situation

It starts with this step: look at your calendar and try to measure a few weeks. Look for both the "good weeks" and "bad weeks," look for interruptions making your day or week bad.

Try to answer these questions:

1. How much focused time do you need to complete your tasks on time? In my experience, at least 40% of my time should be for focused work.
2. Are there any recurring meetings you shouldn't participate in?
   - Can you be an optional guest? It might save you time.
3. How many fragmented time slots do you have?
   - Fragmented time is when you only have 30 or 40 minutes between two events.
   - Rescheduling events could help you avoid this.
   - Also, you can have a rule like mornings for meetings, afternoons for focused work.
4. What are the most time-killing activities?
   - Is there anything you can delegate to someone else?
   - Is there anything you can automate?

This is a worthwhile first step. Even with this, you might have a more practical schedule. If you have the results, you can start planning your day.

Daily Planning

I do this thing each day.

At the end of the day:

- I check out my calendar and update it.
- If there were any over-run meetings, I update the invite.
- If there any meeting that initially wasn't in my calendar, I create an event to track it.
- I make a list of the "5 minutes questions" I've answered. I might create a wiki page which could help others to find some answer later.
- I check out my next day. Do I have to prepare for anything?

At the beginning of the day:

- I check out my day, looking for any changes since the last day.
- I prepare for any upcoming meetings, gathering info, taking notes for questions, etc.

You can start practicing these simple habits today. After a week, you should begin to feel that you are the boss of your own time. If you've successfully built these things into your days, you can go to the next level and start planning your weeks.

Weekly Planning

The purpose of doing weekly planning is to have some kind of agenda for your week answering the question: what will I deliver this week?

For me, Fridays work the best for this. After lunch, I create a new document for my next week, and I spend at least one hour to figure out "what can I finish now?" and "what should I finish next week?"

Review all the meeting notes:

- Is there an action item assigned to me?
- If I can do it quickly, I complete it.
- If I can't, I schedule it for next week.

Checking out Slack and email:

- Do I have action items?
- Do I have snoozed reminders?
- Have I starred anything to bother with it later?
- The same goes here as well: I do it, or I schedule it for next week.

Checking out my to-do list:

- I write down lots of things to remember during a week
- Some items go to meeting notes; some just get archived
- Some are actual action items that I either do or schedule for next week

Checking out my calendar

- I lock down all two hours or more free space as "Focused time—please don't disturb."
- If I don't have enough focused time for the week, I try to reschedule events
- I check out all meeting invites to see if I have to prepare for something
- I send out invites if I need to discuss something

Of course, I am somewhat flexible with these booked focused time events, but it helps me avoid fragmented time.

Things are always changing, so do the calendar events, but these habits have helped me have well-organized weeks.

2. A Team's Always Smarter Than an Individual

This might be a bold statement, but I've experienced this many times in my career. At the end of the day, a team always makes a better decision than an individual one.

There are many books, blog posts, and workshops about the importance of teamwork, but as you are responsible for your team, it is your duty to involve everyone in the decision-making process.

Primarily involve those who recently joined the team, any junior or mid-level member of your team, or someone not experienced with the topic. Now is an excellent opportunity:

- for them to learn,
- for you to empower them,
- for the team to grow together.

Ask them, involve them, trust them. Leave no-one behind.

There's also an important lesson here if you became a lead developer because you were the senior or had the most experience or domain knowledge within the team. In this case, you need to learn how to let go of your solutions. Sure, you need to make a final decision many times, but otherwise, you need to support and encourage other team members.

3. Saying No Without Hard Feelings

For me, it is *so easy* to be a yes-man. I like my work, I like challenges, and I want to prove I'm good at the job. These are quite essential things in achieving any form of success.

So saying no was a hard lesson for me. Because I've always felt that by saying no, I let people down, or if I'm not as impactful, maybe I'd be less likable.

You don't need to run circles around this one as I did. I'm here to tell you: sometimes you must say no. Saying no doesn't mean you let people down. And you are going to be as much or even more impactful and likable.

Say no to:

- Impossible deadlines. If it's impossible, this can't—or shouldn't—get you into any trouble or more trouble than a missed deadline.
- Things you don't believe in. A workplace often isn't a place for saying no, but if there's something fundamentally against your values, you should raise your concerns.
- Things that would hurt your team, your product, or your users. I think this goes without saying, but your team, your product, and your users should be the most important folks for you to represent.
- Gossip. Everyone likes to gossip, even me. If it's harmless, I do nothing. Otherwise, I put a large amount of effort to stop it.

After saying no, it is a good practice to explain the reasoning as well. It is also totally cool if you don't know the answer right away and say, "I don't know, let me check that" or "I'm not sure, let me think about that" if you'd like to sleep on it.

Trust me, these are simple rules, but when you are in the middle of a conversation, it is damn hard to say no. But you can start this one in small steps as well.

4. I Work For My Team and My Users

And I definitely don't work for my boss.

So many people do tasks because "their boss told them to do so." Sometimes that's reasonably good because multiple roles exist for this purpose. You have to work with many people around a company who's job is exactly to tell you what to do: seniors, managers, architects, even the CTO.

But you should remember: they must involve you as well in the decision-making as you are the one responsible for your team. That's why you are a lead developer. It is not your job to satisfy your boss, though. If you ever feel you have to do stupid things for your boss of any kind, give honest feedback and if things are not changing, quit. I'm serious. Just quit. Plenty of great companies are looking for experienced engineers like you are, so you don't have to waste your time on rubbish things.

Always keep in mind and speak for what's the best for your team, what's best for the product you are building, and what's best for the users who will use it and who are going to pay your bills by the end of the day.

5. Prepare For The Fickle Finger Of Fate

I know it's hard not to fix some things: Legacy code or technical debt. Code is just hard to work with. There are so many of these.

But there's always an important question to answer here: do you *need* to fix it?

Is this something that is preventing the success of your team in any way? Is this something that could break and cause incidents or on-call work? Is this something that would make users have a hard time using your product?

If there's a yes to any of these questions, you should work with your team on fixing it. And if the answer is no to all of those questions, you shouldn't fix it but plan to do so later.

It could be a good practice to work on those things before, for example, adding a new feature that could affect any risky areas. But if something is working right now and isn't causing any more trouble than hurting your eyes, it should not be your mission to fix it.

Prepare for both good and bad things to happen, but don't let these things prevent you from moving forward.

6. I Must Be Open

Well, first of all: be open to learning from your mistakes. It is absolutely OK to make mistakes, as long as you learn from them and grow from their lessons. Don't be afraid to break things. That's part of your job both as a developer and as a lead developer.

Also, please be open in general. Open to new solutions. Open for upcoming changes. Open to ideas. Open to discussions. And most of all: open to people.

Being open and diverse is not a fancy Silicon Valley startup thing to do. It is the way everyone should do their job. It doesn't matter whether you work with a woman or a man. With an older or a younger colleague of yours. With someone who speaks your language or with someone who doesn't. And so on.

The only thing matters are someone's knowledge, passion, and willingness to do and learn. You must only evaluate their performance by these values and the work they've done, nothing else.

Key Takeaways

Wrapping up all the things above:

- Don't ignore soft skills. Those are important and hard to learn.
- You should help your team create value for your users.
- Start it in small as it is easier to win one step at a time.
- Fixing everything should not be your mission.
- Time is always really the most valuable thing.
- Sometimes, you must say no.
- You must be open.

And for a final thought: don't be afraid of mistakes. Remember what Theodore Roosevelt, the 26th President of the United States, said:

> *The only one who makes no mistakes is the one who never does anything.*

Gabor Nadai, but you can call him Mefi. Engineering Manager, currently on an intergalactic mission working with a talented group of engineers and managers at Bitrise. Former VP of Engineering at ingatlan.com, Hungary's market leading real estate listing portal. His mission is building awesome communities and teams creating great software which truly helps people. @mefiblogger

Related Reading

- *Defining Project Metrics* by Terri Morgan, September 2020. http://phpa.me/defining-project-metrics
- *Up to My Eyeballs in Technical Debt!* by Steve Grunwell, May 2018. https://phpa.me/grunwell-technical-debt
- *The Dev Lead Trenches: Simple Project Management* by Chris Tankersley, November 2017. https://phparch.com/magazine/2017-2/november/

Borrowed this magazine?

Get php[**architect**] delivered to your doorstep or digitally every month!

Each issue of php[**architect**] magazine focuses on an important topic that PHP developers face every day.

We cover topics such as frameworks, security, ecommerce, databases, scalability, migration, API integration, devops, cloud services, business development, content management systems, and the PHP community.

Digital and Print+Digital Subscriptions Starting at $49/Year

http://phpa.me/mag_subscribe

php[**architect**]