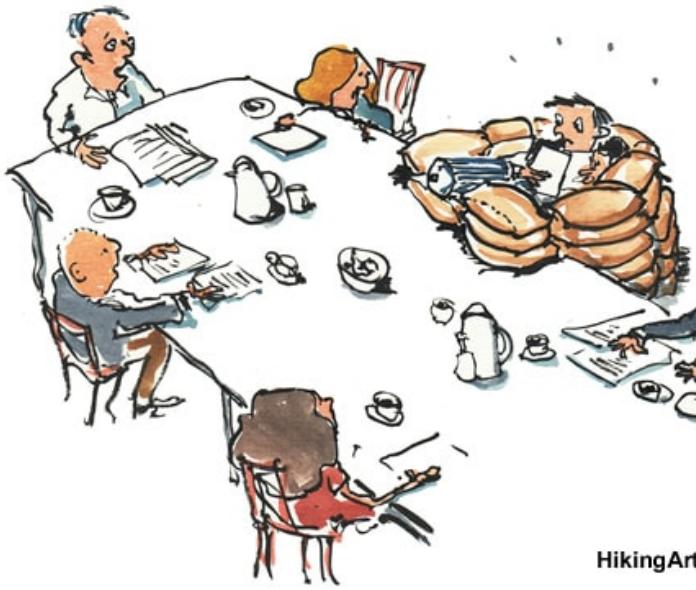
Becoming a CTO

October 13, 2015

If you ever find yourself writing a blog post on why PHP sucks, you are not ready.

Being a CTO doesn't mean you are the craziest hacker on the team. In actuality, writing code is probably the last thing you will be doing. Instead it's a person who is able to communicate the technology to others and lead them through execution of the project. In some ways, it's a person who protects the technology team from outside interference and when needed, takes responsibility for mistakes.

It's easy to accidentally become a CTO by getting hired by a small company with no other developers, like you would see in most startups. But while the title might say "Chief Technology Officer," it is probably just a fancy acronym for an "overworked developer." This is then about how to realize what the position needs and how to get there.



I hate environments where the technology department is simply implementing what was envisioned by others. It might as well be an outsourced development team because it becomes independent of the decision making process. As Camille Fournier puts it in 'On the role of CTO,' "The CTO must protect the technology team from becoming a pure execution arm for ideas without tending to its own needs and its own ideas."

If a company needs a CTO then that person defines the vision for a long-term impact on technology. More and more businesses today are not using technology, but instead are defined by it. Everything is built on technology, from retail ecommerce to mobile apps and it all benefits greatly when someone inside is pushing the technology to make it greater. It's a fight a CTO can't give up and allow the technology team to be the implementers.

I've sat in a few roundtables with fellow CTOs and the majority of the discussions were about how to get the other parts of the company to respect the development processes. For example, how to sell unit-testing to the board. No, no, no! Speaking for myself, I'm not going to sell you on anything. I'd say "This is the direction we are going and this is how it's going to work." In short, whatever the CTO says, that's how it is going to be. If a CTO doesn't have that sort of power for technical decisions, they are not a CTO, but more of a lead engineer, in a suit. Suit being optional.

CTO is a job in business strategy, one which defines the direction of technology inside a company. It's not the right job if you hate meetings, dealing with non-technical people and think that all managers sit all day and do nothing. All of this can be learned though especially since there are many great books to help. Any meeting is a discussion on the direction the business is going and how technology can help with that, or sometimes how technology can create new opportunities for growth. This all needs to be explained in plain English, in a manner which is understandable to everyone.

Thus it is essential to understand the needs of the business and the customers. From my experience, a lot of technical people like to distance themselves from the "business stuff". Yet it's the number one thing to know as a CTO. No technical decision should happen in a vacuum of purely software or hardware concerns. Most of the time the CTO should be in sync with product managers so the strategy for products is consistent with the developmental operations.

Ultimately CTO creates an environment which allows the team to achieve great things. Today a big part of the problem is hiring. Everyone is looking for developers but there is only so many of them, hence the environment should be as welcoming as possible. This is something the CTO should know how to do very well, since they were once one of them. If the team wants to do TDD, or pair programming, or staging servers - they get it all approved by the CTO. It's up to them to consider the bigger impact of those changes.

Thinking about the financial impact of technology operations is important. A startup might allow itself to jump on whatever is greatest and latest, but a bigger organization can't afford to. Everything should be weighed on a return-on-investment basis, asking how much value does it deliver to the customer. Therefore in most cases, it's about balancing the advancement and updates of existing infrastructure, while not falling through the rabbit hole of massive rewrites. Finding what gives 80% of return with only 20% of the cost, following the 80-20 rule, is a big part of it.

I've done interviews with future CTO candidates where they would ask, "Why are you stuck with this old version, why haven't you rewritten it with React.js?" Sorry to burst your bubble, but that's not a great idea. Sometime down the road legacy applications become expensive to manage, but rewrites almost always deliver zero value to the customer. It's about balancing development efforts. Building a team of people who only want to touch the freshly baked technologies is not sustainable.

It builds a stage to implement the things you find most important. For example, I think reliability and security are the two most important features of any software. Thus any changes in the business goals are weighed against that. Privacy of course is right there as well, a topic which is still hard to grasp for non-technical people. Yet it sometimes limits what a company can pursue simply because things might fall outside of respecting privacy. Part of my job is to handle this and make sure they don't.

While all I do is technology, I think technology should be invisible to the user. So when it comes to discussions on whether or not PHP sucks, I think it's an irrelevant question. It is a stimulating

question to some, but for an organization it shouldn't matter. I'd like to believe that this is a stepping stone to becoming a CTO - not caring about these sort of things, not overly focusing on technical details but instead looking out for the team and everything else on top of that.

Becoming a CTO is about realizing the bigger picture of technological impact, which is how it defines the business and how it helps the customer. It helps to be amazing at understanding the technology itself, but it's much more than that.

« The Ludicrous Tesla

I Make Enterprise Software »